Analysis of Single Event Effects on Embedded Processor

electronics

MDPI

Article

Analysis of Single Event Effects on Embedded Processor

Sarah Azimi *, Corrado De Sio, Daniele Rizzieri and Luca Sterpone

Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, 10129 Turin, Italy; corrado.desio@polito.it (C.D.S.); daniele.rizzieri@polito.it (D.R.); luca.sterpone@polito.it (L.S.)
* Correspondence: sarah.azimi@polito.it

Abstract: The continuous scaling of electronic components has led to the development of high-performance microprocessors which are even suitable for safety-critical applications where radiation-induced errors, such as single event effects (SEEs), are one of the most important reliability issues. This work focuses on the development of a fault injection environment capable of analyzing the impact of errors on the functionality of an ARM Cortex-A9 microprocessor embedded within a Zynq-7000 AP-SoC, considering different fault models affecting both the system memory and register resources of the embedded processor. We developed a novel Python-based fault injection platform for the emulation of radiation-induced faults within the AP-SoC hardware resources during the execution of software applications. The fault injection approach is not intrusive, and it does not require modifying the software application under evaluation. The experimental analyses have been performed on a subset of the MiBench benchmark software suite. Fault injection results demonstrate the capability of the developed method and the possibility of evaluating various sets of fault models.

Keywords: embedded processor; reliability; single event effects

Citation: Azimi, S.; De Sio, C.; Rizzieri, D.; Sterpone, L. Analysis of Single Event Effects on Embedded Processor. *Electronics* **2021**, *10*, 3160. https://doi.org/10.3390/electronics10243160

Academic Editor: Alexander Barkalov

Received: 19 November 2021
Accepted: 14 December 2021
Published: 18 December 2021

1. Introduction

The continuous downscaling of transistors is leading to more performant devices due to smaller device sizes with higher operating frequencies. However, smaller size and higher operational frequencies lead to systems that are more vulnerable to soft errors. When a charged particle interacts within the silicon matter of a circuit and releases its energy, it might lead to a bitflip in the sequential logic of the circuits such as memory and registers, known as single event upset (SEU). An SEU affecting the sequential logic of a processor system executing an application can result in wrong outcomes or even in the crash of the system with a dramatic effect on the application [1].

There are several methodologies for investigating the impact of soft errors affecting a processor system. Performing radiation tests and exposing the system to radiation particles while the application is running on the processor provides the most realistic behavior of the physical phenomena [2,3]. However, most of the time, the final system is not available during the design phase and, therefore, it is necessary to rely on emulation platforms. Many alternatives have been proposed in order to overcome the difficulties of radiation test experiments [4]. Simulating or emulating the radiation-induced fault through fault injection environments introduces an efficient alternative for analyzing the effect of soft errors on the execution of an embedded processor.

In this work, we focused on the processing system integrated into all programmable-system-on-a-chip (AP-SoC), analyzing the effect of single event effects (SEEs) on the behavior of the application executed on the embedded processor. To do so, we have developed a fault injection platform for emulating different SEEs and detecting the effects they produce on the running application. Differently from the state-of-the-art solutions, we did not only focus on the evaluation of the SEU effects, but also expanded the analysis covering other fault models such as single event multiple upsets (SEMUs) and clear and present content (affecting either the memory resources or registers of the processor system). We evaluated

the effects of the faults during the execution of different software applications on an ARM Cortex-A9 platform embedded within a Xilinx Zynq-7020 Reconfigurable AP-SoC device. To the best of our knowledge, it is the first work that exploits the operating system mapped on the embedded processor for executing a fault injection environment which leads to a fast and comprehensive fault injection execution.

Moreover, we performed a deep investigation on the outcome of the software applications while soft errors affect the memory and registers of the processor system.

Please notice that the developed platform is not targeting the software-level fault injections but targeting the hardware faults and their impact on the execution of the software faults.

The paper is organized as follows. Section 2 illustrates previous related works. Section 3 describes the evaluated fault models, while Section 4 describes the proposed radiation analysis platform. The obtained results are reported in Section 5. Finally, Section 6 contains a conclusion and discussion on future works.

2. Related Works

Several works have been dedicated to evaluating the soft errors affecting embedded processors through both radiation test experiments [5]. However, due to the challenges in terms of costs and availability introduced by radiation tests, fault injection techniques are widely exploited to investigate the impact of soft errors on embedded systems [6–8]. In [9], the authors developed a simulation-based fault injection platform for fault injecting bits of the in-memory resources, CPU registers, and interconnection infrastructure. The authors in [10] investigate the impact of soft errors on the operations of a microprocessor-based architecture by injecting bitflips at a random time and in a random location on the resources of an application running on an 80C51 microcontroller and a 320C50 Digital Signal Processor. A fault injection technique is proposed in [11] which allows injecting bitflips at the LLVM compiler's intermediate representation (IR) level.

The evaluation of transient faults occurring in processors' computational units, memory units, and CPU registers is not considered. CAROL-FI is a fault injection platform proposed by [12], which allows the injection of several fault models in a randomly chosen variable during the execution of an application under the test, focusing on Xeon Phi devices and NVIDIA GPUs as the main supercomputer cores. The EFIFT tool, which has been proposed by [13], is a GDB-based fault injection platform that corrupts the execution of the application based on the chosen fault model, such as corruption of variable data, registers, and memories. Instead, the platform proposed by [14], called PROPANE, acts by stopping the execution in order to replace a fault-free portion of the code with a corrupted one and resume the execution after. On the other side, instead of targeting the application under the test, a different approach has been followed by the authors in [15]. They have proposed a framework for injecting in real-time multicore embedded systems, focusing on monitoring the whole system through the running operating system and accordingly, injecting faults in locations that include the OS, specific resources such as hardware counters but without specifically addressing memory or processor registers.

The main contribution of this work is the development of a python-based fault injection platform capable to perform hardware-oriented fault injection on the memory and register resources of embedded microprocessors. The platform supports a larger set of fault models and an in-depth analysis of the outcomes resulting from the fault injection campaign, considering not only the behavior of the application but also the exception generated towards the operating system. The use of a python environment running on the operating system of the embedded microprocessor results in a fast fault injection platform with respect to the state-of-the-art.

3. Radiation-Induced Fault Models

When a radiation particle interacts within the silicon device and releases its energy, it can generate different phenomena. In order to perform a deep investigation of the

SEE effects on embedded microprocessors, we have considered single and multiple effects within memory and register cells, and on the functional memory module activating memory page preset and clear.

3.1. Single Event Upset in the Memory

The executable binary of a software application is stored in memory. When a radiation particle hits this memory and releases its energy, it can change the state of a single memory element, referred to as single event upset (SEU), resulting in a malfunction or eventual abnormal termination of the application.

3.2. Single Event Multiple Upset in Memory

By transistors scaling down, the distance between adjacent memory cells is drastically reduced. The cells with small separation distances can have multiple sensitive junctions collecting the charge that is released by a single interacting particle. Therefore, as an effect of one single incident, multiple events (bit value corruption) can arise. Results of many radiation tests indicate the occurrence of single event multiple upset in memories. Previously, we have performed a radiation test on Xilinx Kintex-7 SRAM-based FPGA with the ultrahigh energy heavy ion beam at the CERN facility [3]. By elaborating the readback data file of the configuration memory and analyzing the location of SEUs in the configuration memory, different upsets patterns have been found as multiple upsets occurred close to each other forming a cluster, which is known as SEMU occurrence. Similar multiple-bit upsets (MBUs) have been seen in reports of previous radiation tests with lower energy as well [16,17]. By analyzing the readback file, we have found different cluster patterns suffering from MBU. Figure 1 represents different identified cluster configurations.

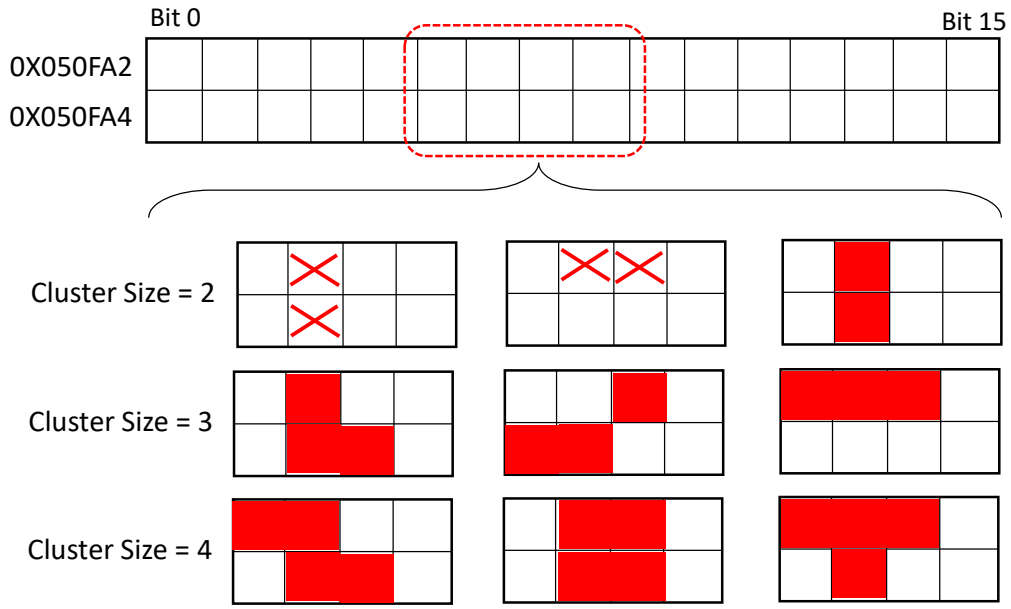

Figure 1. The observed Cluster SEMU pattern during the radiation test.

The sensitivity rate of each cluster configuration, represented as cross-section sensitivity has been calculated and represented in Figure 2. As it can be observed, the cluster configuration with the two adjacent bits corrupted by a single particle is the case with the highest cross-section.

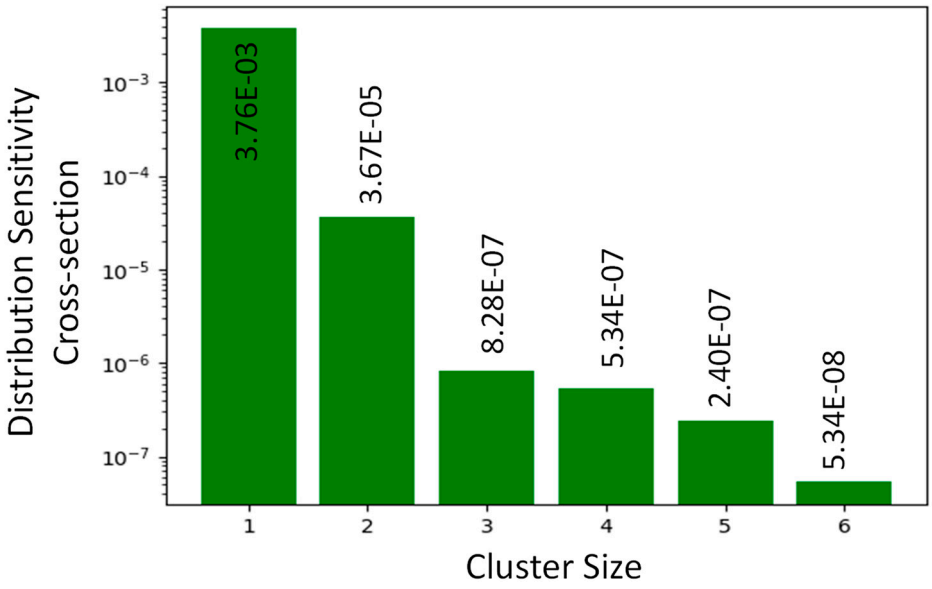

Figure 2. Cluster distribution cross-section for different cluster sizes.

Considering that the technology of the processor integrated into the Zynq-7000 as the device under the test is 28 nm, the same technology of Xilinx Kintex-7 used for the radiation test performed at [3], we have referred to the same SEMU occurrence for evaluating their effects within the processor system memory. Therefore, we considered the occurrence of two bitflips in the adjacent location as the condition with the highest occurrence among SEMU phenomena. Figure 3 represent the general scheme of the processing system of AP-SoC under the study.

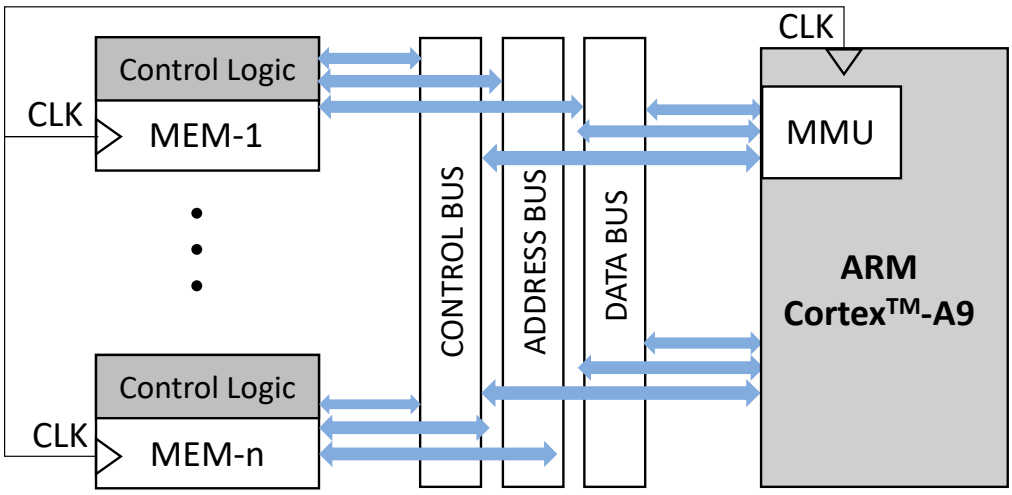

Figure 3. The general scheme of the processing system of AP-SoC.

3.3. Clear Content in the Memory

The clear content in memory resources refers to the situation in which the value read in a memory is 0 due to the particle strike [18]. This effect is principally due to failure happening within the control logic resources. Control logic is one of the most important parts of the memory, with the role to manage several signals and decode instructions coming from the processor. The particle hitting the control logic can cause SEU/MBU that results in clear content in the affected memory bank [19]. This effect may be extremely critical. Considering that the memory management unit (MMU) is responsible for translation between the virtual address, computed and processed by the PS, and the physical address. As an effect of SEU/MBU in MMU, a virtual address can be translated to the

wrong physical address, leading to the PS to request for a physical address with the content of 0 [20].

Moreover, as an effect of particle incident in the clock signal of the memory element, a disturbance voltage pulse or glitch can be generated known as single event transient (SET). This clock glitch could lead to a false rising or falling edge, leading to the transmission of an erroneous data value or clear content in memory [21].

3.4. Single Event Upset in Processor Registers

Registers of the processing system are one of the most important resources since they can provide fast and efficient calculations and data manipulation. They can be implemented in two ways: the former, as a fast static RAM (SRAM) block, having dedicated input and output ports in order to result in minimum reading and writing latency; and the latter, as 'flip-flop' structures suitable for data storing and manipulation.

The ARM Cortex-A9 Processor, as the embedded processor under the study, has 17 available 32 bits core registers, 13 general-purpose data registers, 3 special-purpose registers (stack pointer (SP), link register (LR), and program counter (PC)), and a status register, the current program status register (CPSR), containing information about the current state of the core and its operating modality.

If the particle strikes the registers of the processor during the execution of the application SW, it can modify the state of the affected register, change the content of the register bits, leading to SEU, and eventual malfunction of the application software. For example, if the particle strike causes SEU on one single bit of the PC, it can lead to the execution of the wrong instructions or to jump to a forbidden memory segment provoking a segmentation fault error.

3.5. Single Event Multiple Upset in Processor Registers

The same as the SEMU on the memory resources of the processor system, due to a single highly energetic particle strike, depending on the strike angle and linear energy transfer (LET) of the particle, multiple bits of the registers might be changed, leading to SEMU effect in the registers. Based on the experiments performed at [3] and elaborated in subsection II.B, we investigate the occurrence of SEMU where two adjacent bits are corrupted and changed state with a single particle strike.

3.6. Clear/Preset Content in Process Registers

The clear content fault model addresses the situation in which during the execution of the application SW, the content of a register is forced to 0 while the preset refers to a case in which all the bits of a register are forced to 1.

In the case that the registers are implemented using SRAM blocks, a particle affecting the control logic can cause a clear content or preset content situation. On the other hand, if the registers are implemented using the FF logics, specific control signals, named *preset* and *clear* are dedicated to each FF which are responsible for manipulating the content of the whole register. If a particle hits one of these two signals, it can release its energy and create an undesired voltage glitch. This voltage glitch known as single event transient (SET) can cause a faulty rising or faulty edge for the clear/preset signals and force the registers bit to all 0/1.

4. The Developed Fault Injection Environment

The developed fault injection environment is based on a python overlay executed within the operating system (OS) able to perform physical resources modification and classify the failure effect considering the behavior of the application and the exception generated at the OS level. We have developed a python-based environment, which is able to emulate different radiation-induced SEE effects, on both the memory resources and registers of the processing system of the AP-SoC, during the execution of the application SW.

Moreover, the outcome, as well as the execution status of the application SW, is elaborated in detail to investigate the cause of the failure of the application.

4.1. Single Event Effect in Memory

In order to evaluate the impact of the different SEE affecting the memory resources of the processing system, where the binary of the application, executable and linking format (ELF) is stored, we have developed the python-based workflow represented in Figure 4, executing on the Linux operating system of the ARM Cortex.

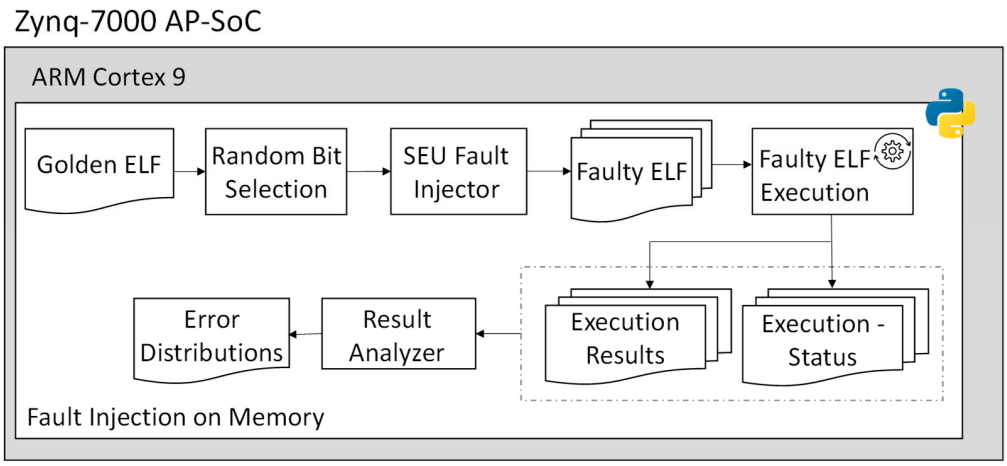

Figure 4. The Developed environment for SEE injection on memory.

As a first step, the golden ELF file consisting of the non-faulty executable binary of application SW has been provided as an input to the developed fault injection environment in order to generate the fault-free (golden) output of the application. As a second step, the fault injection acts on the generation of the faulty ELF file. It reads the golden ELF, selects a random bit in the golden executable binary, performs emulation of the different SEE effects by injecting a bitflip in the selected bits, and creates a faulty version of the binary executable file. Based on the injection number selected by the user, the bitflip injection is repeated, generating the different faulty ELF files. Please note that the executable binary of the software application running on the processor is loaded in the memory of the processing system. Therefore, injecting a bitflip in the executable ELF file is modifying the value stored in the memory of PS, injecting bitflip on the memory element of SoC, leading to emulation of SEU effect on the memory.

We have instrumented the developed python tool to exploit multi-processing features, dedicating each process for generating and executing the faulty ELF or executable binary of application SW. Moreover, in order to observe the effect of SEU on the memory used for storing the ELF of the application SW, the execution status or exit code of the process which executes the faulty ELF file is collected and classified by the main process. Moreover, the outputs generated by executing the faulty ELF file are observed and compared with the golden output in order to provide a more detailed investigation of the impact of SEE affecting the memory of the processor system on the output of the application SW.

4.2. Single Event Effect in Registers

The second analysis workflow is dedicated to emulating the radiation effects within the processor registers. We enhanced the environment to mimic the SEE effect during the runtime of the application on the PS, represented in Figure 5. It is based on a python extended GDB tool, executing through the Linux operating system, running on the target ARM microprocessor.

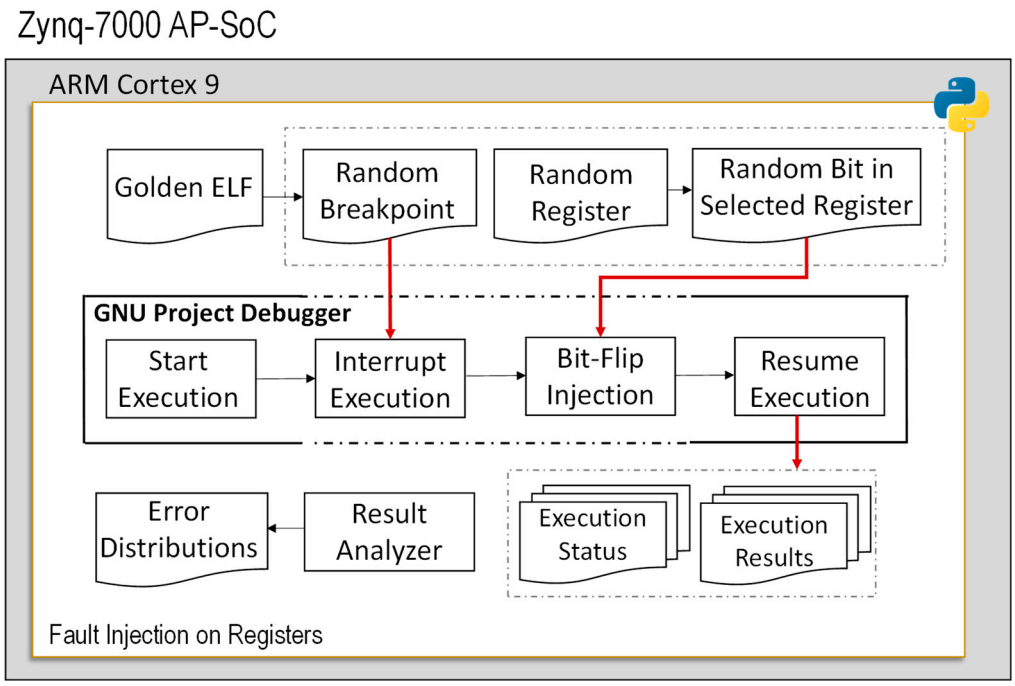

Figure 5. The developed environment for SEU injection on register.

The flow starts by running the golden (fault-free) application and obtaining the golden outputs of the application software. It moves forward by emulating the SEE effect on the software accessible registers during the runtime of the application. To do so, a register among software accessible registers is selected randomly as well as a bit among the 32 bits of the selected registers is chosen randomly for performing a bitflip. Please notice that the PS under the study is based on ARM-v7 architecture. Therefore, the software accessible registers include the floating-point status and control register (FPCR), the 64 NEON technology registers, the general-purpose registers (R1 to R13), and the three special-purpose registers named as program counter (PC), link register (LR) and stack pointer (SP) [22]. After the bitflip selection, the fault injection tool computes a random breakpoint node. The breakpoint node is selected in two steps: firstly, a random instruction number among the source instructions of the application, starting from zero to the total number of instruction lines (LOC) of application SW is selected. Secondly, exploiting GNU project debugger (GDB) features, a random assembly instruction offset starting from the selected LOC baseline is chosen. This defines the breakpoint node for performing bitflip injection. Please note that choosing a random breakpoint with this methodology overcomes the limitation of time-based breakpoint selections for applications with short execution time.

The injection phase is carried out exploiting the GDB, an open-source debugger that uses low-level system calls such as ptrace to monitor and modify the value of the core resources as well as to control the application execution flow. To elaborate more, thanks to the features provided by GDB, the fault injection platform starts the execution of the application with the golden binary, interrupting the execution when it reaches the randomly chosen breakpoint node, injecting bitflip by modifying the value of the selected bit of the selected register and resuming the execution of the application until the termination. The same as the previously developed platforms, we have exploited the multi-process features of python for executing single register bitflip injection. Once each process is terminated, the python environment evaluates each application's results and the computation exit code and compares the result with the golden execution in order to define and classify the error distribution.

4.3. Error Analysis

As it has been described in the previous sections, in both of the developed platforms, fault injection in the memory, and registers, we have exploited the multi-process features provided by python. To elaborate, the binary of the faulty application is passed to the python subprocess for the execution, and the status of each process executing the faulty application is monitored through the execution by the developed platform. The fault injection platform evaluates the execution status of the application and classifies faulty applications in different groups:

- Silent Dara Error: If the injected fault does not affect the normal execution of the application SW and the application terminates the execution without any risen exception, then the termination status of the application SW does not indicate any error. However, to perform a detailed analysis we move forward by investigating the output of the application SW while the normal termination status is received. We compared the output of the faulty applications with the golden one in order to evaluate the case in which the injected fault leads to the normal termination of the application SW but leads to the incorrect output of the application SW labeled as silent data corruption (SDC). On the other side, if the injected fault model leads to the risen of exceptions by the operating system during the execution of the application SW that could not be masked or safely handled by the OS, the execution will be interrupted before the end of the application and the termination status represents the exception signals raised by the OS.

We have analyzed the impact of the injected fault model by performing a deep investigation on the cause of each raised exception and classified them as follows:

- Segmentation Fault: Due to the injected fault, the application SW requests for accessing a memory segment out of its own mapped memory area. It occurs when the injected fault modifies the address provided to the instruction operand or modifies the content of registers such as PC and SP.
- Illegal Instruction: In the case that the injected faults corrupt the OPCODE of instructions, the process tries to execute an illegal instruction which leads to an interruption of the application's execution.
- Bus Error: The bus error is classified as the case in which the process executing the application requests to access a memory location that is not accessible physically.
- Abort: This refers to the condition in which the injected fault leads to the interruption of the execution of the application, not due to an exception risen by the operating system but due to an exception risen by the process executing it, itself which tells the process to terminate.
- Breakpoint Trap: It is defined as the case in which the injected fault causes the process to enter the debug state and execute the breakpoint hook instructions.
- Arithmetic Operation Error: If the injected fault corrupts one of the operands, the process tries to execute an erroneous arithmetic operation which might lead to execution interruption.
- Hang: It is defined as the condition in which the injected fault leads to the occurrence of deadlock in the execution of the application.

5. Experimental Results

In order to confirm the effectiveness of the developed radiation analysis workflows, we have chosen the MiBench benchmark suite as an application to run on the ARM Cortex-A9 Processor embedded within the Zynq-7020 AP-SoC.

5.1. Application Benchmark

As a set of applications running on the PS under the test, the MiBench benchmark suite has been chosen [23]. Mibench contains different application classified on the basis of the target application domain such as automotive, network, security. Between the provided

benchmarks, we chose the *basicmath*, *bitcount*, and *FFT* as the benchmarks dedicated to mathematical and signal processing operations, highly used in space application. However, the developed platform is applicable to any other applications. Some specification about the chosen benchmarks is provided in the following:

- *Basicmath* carries out basic math operations such as cubic functions solving and angles conversions which are highly used in space application systems for autonomous control.
- *Bitcount* is dedicated to bit manipulation as a key part of the computing system.
- *FFT* carries out Fast Fourier Transform, composed of pseudorandom sinusoidal components having variable amplitude and frequency, used in most of the systems that require transmitting and receiving signals.

For each application, there are four groups of instructions: control (unconditional and conditional branch), integer, floating-point, and memory (load and store) [23]. Figure 6 represents the distributions of the selected applications.

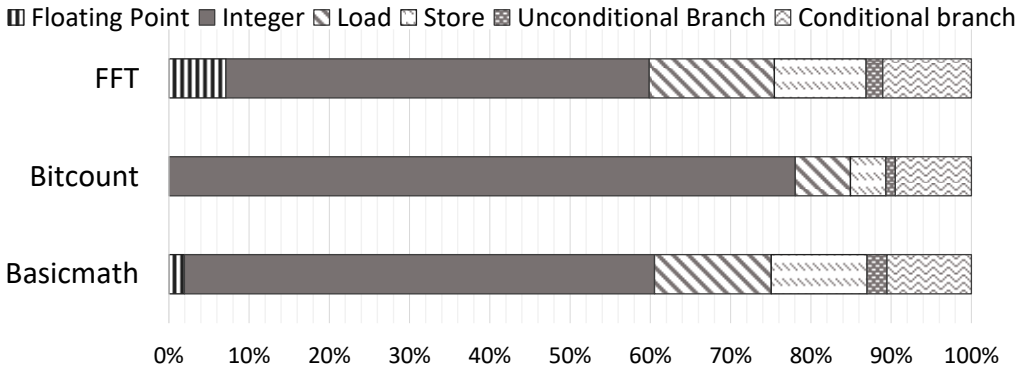

Figure 6. The dynamic instruction distribution for selected benchmarks.

5.2. Experimental Setup

For the purpose of this work, we have used PYNQ-Z2 board with the embedded Zynq-7020 AP-SoC. It consists of a dual-core ARM Cortex-A9 Processor, with a maximal working frequency of 1 GHz, mated with 28 nm SRAM-based programmable logic, equivalent to the Xilinx Artix-7 FPGA. The ARM core allows the SoC to execute bare-metal applications but also full operating systems such as Linux or real-time operating systems such as FreeRTOS. In particular, Xilinx provides a default operating system installation running Ubuntu 18.04, based on version 5.4 of Linux Kernel which has been exploited for executing the developed analysis environment.

5.3. Static Radiation Sensitivity Analysis

In order to perform an accurate radiation analysis, we performed the radiation characterization of the memory resources and registers of the processing system in terms of cross-section, defined as the radiation sensitivity of the cell with respect to the physical characteristic of the technology. We have developed the electrical model of both SRAM cell and Flip-Flop representing the memory and register resources of the PS, exploiting the FreePDK physical library tuned for 28 nm, as the technology under the study, and adopting the electrical Predictive Technology Model (PTM) for bulk CMOS. Using the commercial K-layout tool, the layout description of the SRAM and FF cells have been extracted in terms of Graphic Data System-II (GDS-II). Based on the netlist and layout of the cells, we have performed a radiation analysis, using our in-house Monte Carlo-based simulation tool, described in detail in [5], using the Heavy Ion Profile related to the Université Catholique de Louvain (UCL) facility [24]. We have performed a simulation of 10,000 particles for each cell and we reported the obtained cross-section in Figure 7.

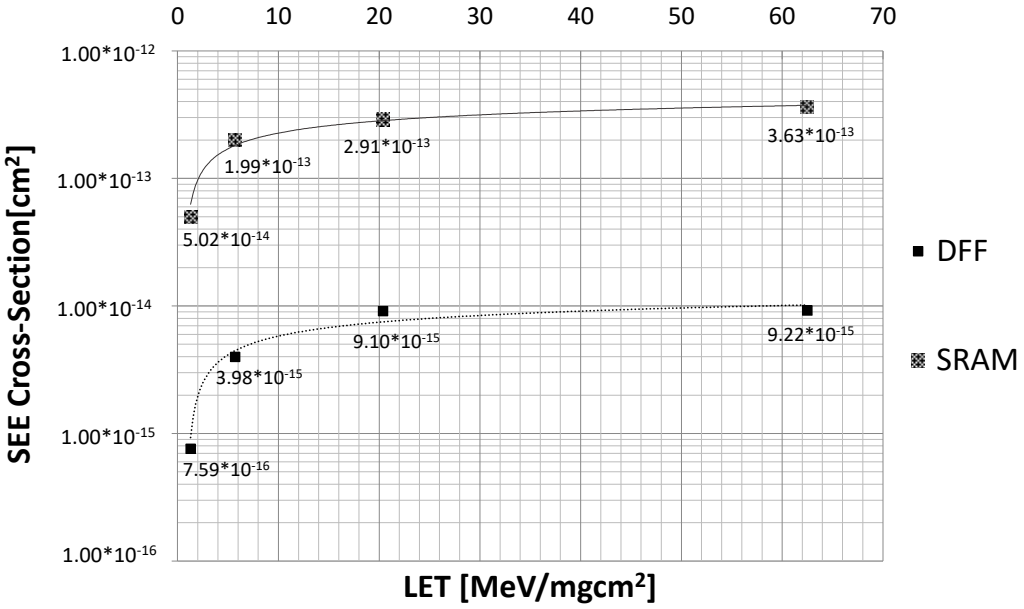

Figure 7. SEE cross-section [cm^2] for static radiation analysis of SRAM and FF in 28 nm.

5.4. Fault Injection Experimental Results

In order to investigate the radiation sensitivity of the selected applications running on the ARM processor, we have performed 10,000 fault injections, both in the memory resources and registers of the processing system, emulating different effects such as SEU, SEMU, clear content, and preset content. Table 1 represents the percentage of the error in terms of total error rate which includes the cases in which the application SW leads to an unsuccessful execution. Table 1 represent the error rate when there is a fault occurring in the memory while Table 2 represent the total error rate while the fault occurs in the registers. An unsuccessful execution is defined as a case where the application terminated its execution successfully, but the generated output of the application is different from the golden one (defined as faulty output or if the execution is interrupted before the end of the application). Moreover, we have performed a detailed investigation in order to identify the effect of the injected faults which leads to the execution status being different than normal. Figures 8 and 9 represent the distribution of the total error rate, elaborating the contribution of each termination status and risen exception in the total error rate. Figure 8 is dedicated to the faults occurring in memory while Figure 9 is representing the faults happening in registers. As it can be observed, most of the errors are dedicated to the case in which the application terminated successfully, without interruptions, but led to a faulty output of the application. Please notice that by exploiting the python-based overlay executed on the host OS, we are able to perform a fast and accurate fault injection for different SEE effects. As an average, for performing 10,000 fault injections on memory resources, approximately, 10 min is required. This value increased to 3 h for the fault injection campaign in register resources.

Table 1. Error rate classified per application and fault model in memory.

Application	Total Error Rate for Faults in Memory		
	SEU	SEMU	Clear Content
Basicmath	7.01	6.89	6.89
Bitcount	1.65	1.67	1.39
FFT	3.94	4.10	4.33

Table 2. Error rate classified per application and fault model in register.

Application	Total Error Rate for Faults in Register			
	SEU	SEMU	Clear Content	Present Content
Basicmath	7.76	8.09	8.43	9.53
Bitcount	4.16	4.50	61.55	1.33
FFT	6.11	11.01	6.49	8.20

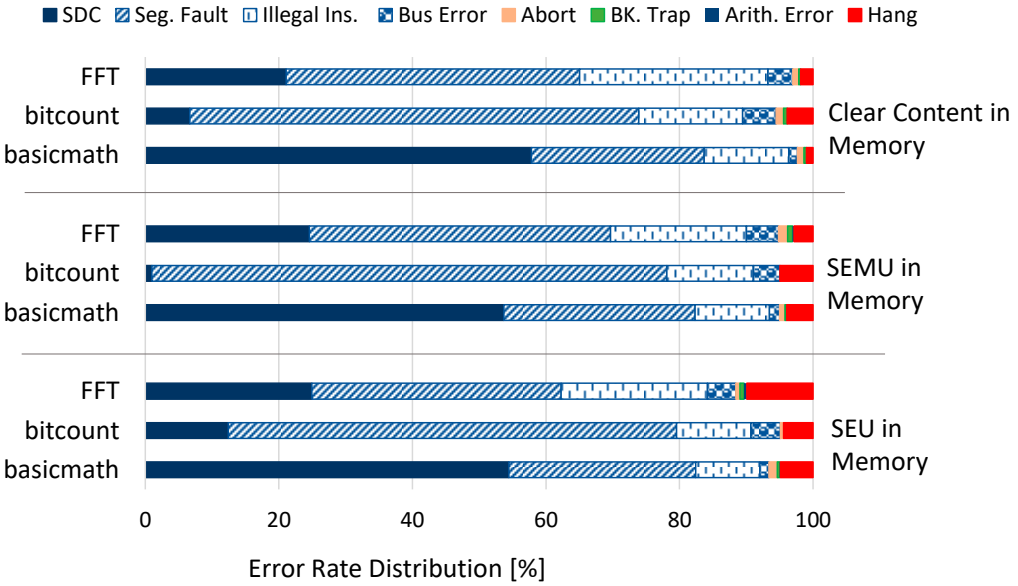

Figure 8. The error rate distribution for the SEE fault injection campaigns in memory.

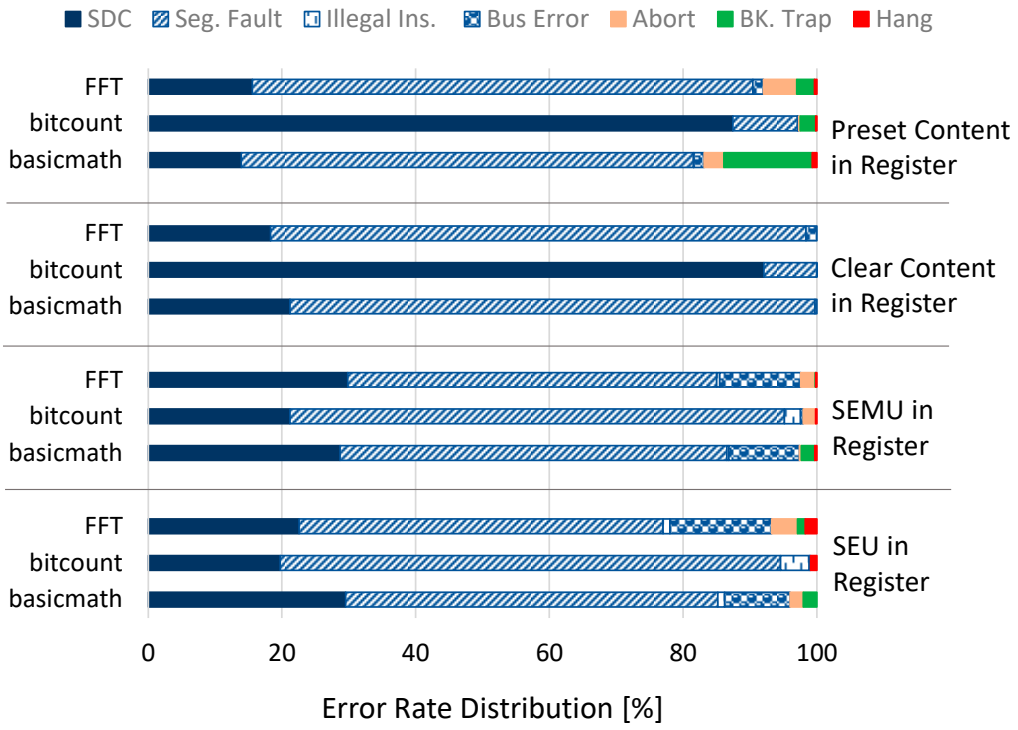

Figure 9. The error rate distribution for the SEE fault injection campaigns in registers.

Please note that different applications have different scopes and thus different assembly instructions distributions. This leads to different sensitivity to faults appearing during

the execution. For example, if an application is more oriented towards bit manipulation instructions (e.g., MiBench.bitcount) then it will be more susceptible to register injection since registers value are heavily used by the application through the whole execution and a single SEU in the registers can compromise the whole functional outcome. Please note that, in this way, one could also assess the sensitivity of different implementations of the same application, analyzing how different choices regarding the instructions and the data structures can affect the overall fault tolerance of the system

6. Conclusions and Future Works

In this paper, we developed a radiation analysis environment capable of emulating different SEE faults on both the memory and register resources of the processor system of AP-SoC while an application SW is executing on the embedded ARM Cortex-A9 of PS. Moreover, an in-deep evaluation of the fault injection campaign has been performed in order to investigate the effects of injected faults and failure causes of the application execution. In future work, we have planned to exploit the outcome of the developed fault injection environment in order to provide an efficient mitigation strategy of embedded nodes in HPC applications.

Author Contributions: Conceptualization, L.S. and S.A. and C.D.S.; methodology, D.R.; software, D.R.; validation, C.D.S.; writing—original draft preparation, S.A. and D.R and C.D.S.; writing—review and editing, S.A. and D.R; supervision, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The developed platform is available on the GitHub repository. The repository includes both of the developed platforms, Single Event Effect in Memory and Single Event Effect in Registers. Each platform has a dedicated readme file that elaborates and guides the readers through the steps to implement and use the python platform. The link to the Github repository is the following: https://github.com/danirizziero/SEE_injection_framework (accessed on 7 December 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Azimi, S.; Sterpone, L. Digital design techniques for dependable high performance computing. In Proceedings of the IEEE International Test Conference, Washington, DC, USA, 1–6 November 2020; pp. 1–10.
2. Sterpone, L.; Azimi, S.; Bozzoli, L.; Du, B.; Lange, T.; Codinachs, M.D. A novel error rate estimation approach for UltraScale+ SRAM-based FPGAs. In Proceedings of the IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Edinburgh, UK, 6–9 August 2018; pp. 120–126.
3. Du, B.; Sterpone, L.; Azimi, S.; Codinachs, M.D.; Cavrois, F.V.; Polo, B.C.; Alia, G.R.; Kast, M.; Martinez, F.P. Ultrahigh Energy Heavy Ion Test Beam on Xilinx Kintex-7 SRAM-Based FPGA. *IEEE Trans. Nucl. Sci.* **2019**, *66*, 1813–1819. [CrossRef]
4. Azimi, S.; Sterpone, L.; Du, B. On the Analysis of Radiation-induced Single Event Transients on SRAM-based FPGAs. *Microelectron. Reliab.* **2018**, *88*, 936–940. [CrossRef]
5. Sterpone, L.; Luoni, F.; Azimi, S.; Du, B. A 3-D Simulation-Based Approach to Analyze Heavy Ions-Induced SET on Digital Circuits. *IEEE Trans. Nucl. Sci.* **2020**, *67*, 22034–22041. [CrossRef]
6. Cho, H.; Mirkhani, S.; Cher, C.; Abraham, J.; Mitra, S. Quantitative evaluation of soft error injection techniques for robust system design. In Proceedings of the ACM Design Automation Conference, Austin, TX, USA, 29 May–7 June 2013; pp. 1–10.
7. De Sio, S.C.; Azimi, S.; Portaluri, A.; Sterpone, L. SEU Evaluation of Hardened-by-Replication Software in RISC- V Soft Processor. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Athens, Greece, 6–8 October 2021; pp. 1–6.
8. De Sio, C.; Azimi, S.; Sterpone, L. On the analysis of radiation-induced failures in the AXI interconnect module. *Microelectron. Reliab.* **2020**, *114*, 1–6. [CrossRef]
9. Rosa, F.; Kastenmidt, F.; Reis, R.; Ost, L. A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Amherst, MA, USA, 12–14 October 2015; pp. 211–214.
10. Velazco, R.; Rezgui, S.; Ecoffet, R. Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection. *IEEE Trans. Nucl. Sci.* **2000**, *47*, 2405–2411. [CrossRef]

11. Lu, Q.; Farahani, M.; Wei, J.; Thomas, A.; Pattabiraman, K. LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults. In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, Vancouver, BC, Canada, 3–5 August 2015; pp. 11–16.
12. Oliveria, D.; Frattin, V.; Navaux, P.; Koren, I.; Rech, P. CAROL-FI: An Efficient Fault-Injection Tool for Vulnerability Evaluation of Modern HPC Parallel Accelerators. In Proceedings of the ACM Computing Frontiers, Siena, Italy, 15–17 May 2017; pp. 295–298.
13. Tian, N.; Saab, D.; Abraham, J. ESIFT: Efficient System for Error Injection. In Proceedings of the IEEE International Symposium on On-Line Testing and Robust System Design, Platja d'Aro, Spain, 2–4 July 2018; pp. 201–206.
14. Hiller, M.; Jhumka, A.; Suri, N. PROPANE: An Environment for Examining the Propagation of Errors in Software. *ACM SIGSOFT Softw. Eng. Notes* **2002**, *27*, 81–85. [CrossRef]
15. Horstmann., L.P.; Frohlich, A.A.; Horstmann, L.P.; Frohlich, A.A. A Fault Injection Framework for Real-time Multicore Embedded Systems. In Proceedings of the Brazilian Symposium on Computing Systems Engineering, Florianopolis, Brazil, 24–27 November 2020; pp. 1–8.
16. Keller, A.M.; Whiting, A.T.; Sawyer, K.B.; Wirthlin, J.M. Dynamic SEU Sensitivity of Designs on Two 28-nm SRAM-Based FPGA Architectures. *IEEE Trans. Nucl. Sci.* **2017**, *65*, 280–287. [CrossRef]
17. Reorda, S.M.; Sterpone, L.; Ullah, A. An Error-detection and Self-repairing Method for Dynamically and Partially Reconfigurable Systems. *IEEE Trans. Comput.* **2017**, *66*, 1022–1033. [CrossRef]
18. Herrmann, M.; Grumann, K.; Gliem, F.; Kettunen, H.; Cavroi, F.V. Heavy ion SEE test of 2 Gbit DDR3 SDRAM. In Proceedings of the IEEE European Conference on Radiation and Its Effects on Components and Systems, Seville, Spain, 19–23 September 2011; pp. 934–937.
19. Bougerol, A.; Miller, A.; Guibbaud, N.; Leveugle, R.; Carriere, T.; Buard, N. Experimental Demonstration of Pattern Influence on DRAM SEU and SEFI Radiation Sensitivities. *IEEE Trans. Nucl. Sci.* **2011**, *58*, 1032–1039. [CrossRef]
20. Wang, X.P.; Kohler, P.; Hermann, M.; Fichna, T.; Pouget, V.; Saigne, F. SEL/SEU/SEFI/TID Results of the Radiation Hardened DDR3 SDRAM Memory Solution. In Proceedings of the IEEE Radiation Effects Data Workshop (REDW), Waikoloa, HI, USA, 16–20 July 2018; pp. 1–5.
21. Balasch, J.; Gierlichs, B.; Verbauwhede, I. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography, Nara, Japan, 28–28 September 2011; pp. 105–114.
22. ARMv7-A Architecture Reference Manual, ARM. 2018. Available online: https://developer.arm.com/documentation/ddi0406/latest (accessed on 25 October 2021).
23. Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; Brown, R.B. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of the IEEE International Workshop on Workload Characterization, Austin, TX, USA, 2 December 2001; pp. 3–14.
24. Akhmetov, A.O.; Bobrovsky, D.V.; Tararaksin, A.S.; Petrov, A.G.; Kessarinskiy, L.N.; Boychenko, D.V.; Chumakov, A.I. IC SEE comparative studies at UCL and JINR heavy ion accelerators. In Proceedings of the IEEE Radiation Effects Data Workshop, Portland, OR, USA, 11–15 July 2016; pp. 1–4.