Microelectronics Reliability 137 (2022) 114667

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

Observation framework of errors in microprocessors with machine learning location inference of radiation-induced faults

Sébastien Thomet [a,b,*], Fakhreddine Ghaffari [b], Serge De Paoli [a], Jean-Marc Daveau [a], Fady Abouzeid [a], Olivier Romain [b]

[a] STMicroelectronics, Crolles, France
[b] ETIS Laboratory, CY University, Cergy, France

ARTICLE INFO

Keywords:
Single-event upsets
Single-event functional interrupt
Fail-reason capture
Fault injection
Hardware assertions
Machine-learning
Error diagnosis
Beam testing

ABSTRACT

This paper presents a non-intrusive failure reason capturing approach for processor-based system-on-a-chip (SoC) integrated as an intellectual property (IP). It provides diagnostic information about the origin of single-event upsets (SEU) in the context of technology qualification to radiations. Thanks to the combination of trace events buffering and error detection with triggering mechanisms, the module is able to capture an execution trace containing the error propagation, using only 1 KB of memory. The execution trace is supplemented by a configurable set of pipeline registers. For single-event functional interrupt (SEFI), we also propose a technique based on a machine-learning algorithm to find from which register the SEU comes from. The captured CPU trace is processed by a classification algorithm, trained on a fault-injection campaign database and providing an accuracy score up to 87 %.

1. Introduction

The growing demand in performance of processor-based systems for safety-critical application, constitutes a challenge for the qualification at mission profile. Spatial application domains require a specific hardening to prevent system failures due to radiation effects. Hardening by design techniques used in application specific integrated circuits (ASICs) involve silicon area, power overhead and frequency limitations, that have an impact on the performances [1,2]. Such devices suffer from several generation lag [3,4]. Software hardening techniques are very efficient to mitigate faults or to detect errors. Nevertheless, the performance degradation is significant, the execution flow may be much more sophisticated if it relies on the replication of procedure [5], of variables [6], or on every instruction added to control the flow [7]. As a result, regular architectures based on reconfigurable logic may become more sensitive with hardened software [8]. That is why, many approaches rely on hybrid techniques combining software hardening with either hardened designs for ASICs [2], redundant architectures for multi-cores systems on a chip [9,10] or hardware control flow checking [7,11–14].

Such designs require to be qualified using accelerated soft error rate (ASER) techniques in irradiation facilities providing particle beam that emulates a specific environment. This study addresses spatial applications requiring heavy ion beam experiments, which can be carried out in particle accelerator facilities, this includes *RADEF*-Finland [15] or *CYCLONE*-Belgium [16] for instance. We are focusing on single-event upsets, as in complex logic designs, a major part of the die area is covered by memory.

To experiment such a complex system under beam, it requires to detect a failure and to have a sufficient hardware observability. This information may help to ensure that the setup is working as expected and errors come from the device under test (DUT) by radiation effects. Techniques related to the post-silicon debug and the software validation [17,18] try to overcome the issue of error propagation in designs with limited observability. Design for debug (DFD) approaches, presented in [19–21], provides a very detailed logical monitoring over a circuit for either JTAG standard or from device dedicated pinouts. But such approaches are design dependent and may be difficult to exploit with processors for the purpose of online error detection, indeed they are quite similar to core debug. Processors are working with a large data bandwidth, and as we can see in [22] addressing raw inputs/outputs with hash function, even with embedded online dedicated hardware to calculate signatures, the bandwidth stay terribly high. Therefore, several authors in literature propose to process the data online to check the flow [23–26].

* Correspponding author at: STMicroelectronics, Crolles, France.
E-mail address: sebastien.thomet@gmail.com (S. Thomet).

https://doi.org/10.1016/j.microrel.2022.114667
Received 21 December 2021; Received in revised form 27 May 2022; Accepted 19 July 2022
Available online 18 September 2022

In this paper, we propose a new approach based on the buffering of CPU trace supplemented by execution related information, and error detection mechanisms. It addresses the context of high reliability requirements for spatial processor-based applications exposed to radiations, which need hardened implementation and beam qualification. In our approach, both trace buffering and error detection features, combined with a triggering system, allow capturing a history of the last executed instructions. This history gives an insight of what was executed before a single-event upset (SEU) led to a crash. Such a diagnostic information help to identify the source of the bit flip and how the running application has been affected. We propose to process the captured execution trace with a machine-learning (ML) algorithm to infer the location of the SEU in the design leading to a single-event functional interrupt (SEFI). We have developed an intellectual property (IP) embedded within a system-on-a-chip (SoC) that provides trace encoding and recording in a buffer, and error detection features based on program counter (PC) checks.

The rest of this paper is organized as follows. The Section 2 gives an overview of the state-of-the-art techniques related to processor observability. The Section 3 describes the test platform, the proposed IP including how it has been designed, integrated and validated. It also explains the use of ML classification algorithms for execution trace processing. The Section 4 presents the results of a fault injection campaign carried out in simulation and the accuracy scores of ML classification. Finally, the Section 5 concludes the paper, and we discuss what is planned to do with the upcoming test chip to validate our approach especially with beam testing in irradiation facilities.

2. Background

Observation in processor is an issue that has been addressed for a while. To tackle down complex logic debugging, or software development, many techniques are proposed with different levels of intrusiveness proportionate to the suspected threats. Some of them are become standard debug infrastructures like CoreSight© [27] for ARM processor cores, trace and debug specifications for RISC-V cores [28], Nexus 5001 Standard [29], etc. The MIPI Alliance consortium [30] proposes many protocols for the exportation of trace data flow outside the chip to perform online system monitoring and to challenge the lake of SoC physical connections. Such protocols may rely on any interface like Parallel Trace Interface, USB, microSD card, HDMI, DisplayPort, etc. Furthermore, Lauterbach [31] provides an external hardware support to debug and trace infrastructures embedded in SoCs. These of-the-shelf products and IPs provide a full debug access and an online trace exportation. Otherwise, some papers described thereafter present more specific techniques either to ensure a consistent system behavior or to export as information as possible for observability enhancement.

Online error detection through the CoreSight trace infrastructure using a dual-core ARM Cortex-A9 implemented on Zynq7000 SoC family has been presented in [23–26]. Error detection combines the use of program trace macrocell (PTM) to track the PC with the instrumentation trace macrocell (ITM) to export calculation results. The proposed program trace data checker (PTDC) IP is implemented in the FPGA and monitor online the program execution during tests. PC values are monitored with a range checking and the ITM data are checked with assertions. Both PC ranges and data assertions are software driven. Authors provide results of proton irradiations [23,24], neutron irradiations [25], and LASER fault injection [26]. They tried to use the embedded trace buffer (ETB) from the CoreSight to catch last executed branches [24,25], but as the buffer is also exposed to the beam, results were often corrupted. Moreover they also mentioned a certain sensitivity of the trace infrastructure.

The technique presented in [32] takes advantage of the unused trace infrastructure of SoCs to track the execution flow. Authors have implemented a control flow checker (CFC). It analyzes the value of the PC and IR registers that are supposed to be available on a debug port at each clock cycle. At each cycle, the CFC decodes the instruction and compacts the machines code to perform code signature calculation and jump verification. If a branch instruction is detected, the next PC value should be either the following one or the one corresponding to the jump. And at each branch instruction the code signature is checked. This approach proposes a dynamic signature table with automatic filling. The IP is basically composed of a memory table including gates for the final state machine. As beam experiments reveal the sensitivity of observation means, such design may be prone to false detection due to fault in the signature table.

First Failure Data Capture (FFDC) presented in [33] is intended to be used in final products to provide relevant data to enable the support in case of operating system crashes. Diagnostic events are generated on exception faults and contain a snapshot of CPU registers. Events are ranked according to their priorities. A smart agent filters low priority event, to keep only high priority ones, in case of memory congestion. This aspect is explained as errors are often followed by many events that fill the buffers, erasing the relevant diagnostic information about the origin of the failure. Such approach to capture information about the failure by catching CPU registers is appropriate in the context of beam testing. The only difference is when an error occurs, the memory is reloaded and the SoC is reset. Such failure-capture technique has been reused in [25,26] to export information about the memory management.

As most of the above techniques are dealing with processors running at hundreds of Megahertz, they must be embedded within the device. Therefore, during beam testing, they are also exposed to radiations. As we need a measurement tool allowing to identify the origin of an error and to calculate an error rate, it must be hardened. Making such structures hardened involves using hardened-by-design cells or a hardened design with redundancy. This also involves more power consumption, a silicon area overhead, high timing constraints, and may affect the running frequency. For this reason, the observation framework must be designed as minimalist as possible. Non-intrusive existing trace architecture is a fair tradeoff as it allows both SoC debug and fail-reason understanding. However, online trace exportation needs a specific connector between the DUT and an adapter connected to the host PC, like CoreSight, or Lauterbach. This adapter must be closed to the DUT and will also be exposed during beam testing. Furthermore, such infrastructures are either proprietary, quite massive designs, or processor dependent. All in all, to tackle down observability needs in such context with high reliability, we propose a processor-agnostic IP which gathers watchpoints triggering events and online trace recording, with a protected memory and a hardened implementation.

3. Contribution

The proposed approach consists in identifying the failure reason of the firmware running on the processor. In harsh environment, devices are prone to SEUs that may affects the application causing either insignificant calculation errors or functional interrupts. A SEFI in a CPU can be due to illegal instructions, incorrect jumps, unexpected infinite loops, watchdogs no longer refreshed, double errors in a memory cell not recoverable by its error correcting code (ECC), and so on. In the end, a failure is often handled by machine trap, either triggered by hardware or by a software interrupt. This doesn't always provide sufficient diagnostic information to understand what has happened.

To detect failures, we have implemented a hardware assertion block that covers errors which lead to make the execution flow to diverge or to leave the expected application. As commonly done in the literature, a circular buffer is used to record the CPU activity and provides to the user a backtrace upon a breakpoint reached in the application during debug sessions. The principle of this buffer is to keep in memory the most recent events and erase the oldest ones. Indeed, the size of the memory only allows to save a very restricted section of the program execution. In our IP, the trace is supplemented by values of results arising out of the arithmetic-logic unit (ALU), and values of the stack pointer (SP) address.

The error detector is composed of a set of configurable hardware assertion checkers monitoring the PC and the data access addresses, as the CPU is based on a Harvard architecture. Triggers can be configured to make selected assertions either to stop the trace buffering, or to freeze the pipeline. Freezing the pipeline is a way of stopping the execution in order to recover further diagnostic information in case of a crash during experimentations in the SoC.

In this manner, when an event is detected by one of the assertions, it often means that a SEFI occurs, as presented in the Table I. In normal operation mode, the probed registers are recorded continuously in the buffer. Therefore, within event captured, the execution trace buffer provides the CPU trace, including the probed architectural registers, composed of a thousand instructions before the SEFI. In a first time, this information enables to ensure the setup is working as expected. Then, they provide an insight in the error propagation through the design allowing to infer where the SEU comes from. The execution trace can be interpreted with regards to golden run or the disassembled code. We will also present a method to analyze traces with machine learning algorithm, including the training part on data from simulation-based fault injection campaigns.

Our module is also providing an un-resettable bank of memory mapped registers where the application running on the SoC can log information, typically operating system context switches, counters, execution reports, error codes, CPU status registers, etc. All resources of the modules are accessible via JTAG debug access. They are mapped on a dedicated debug bus and can be read or written at any time without any intrusiveness on the running firmware.

Although our IP is processor agnostic, it has been interfaced with a RISC-V core by probing some of its registers. It has been integrated in a test chip in 28-nm fully-depleted silicon-on-insulator (FD-SOI) technology. As the physical properties of this technology ensure a significant hardening, the circuit is only prone to SEUs and not to MCUs. Clock and reset trees are hardened by design using low soft-error rate (SER) inverters. The cross section is up to 3 times lower than flip-flops [34] and inverters represent only 7 % of the number of flip-flops in the processor. So, we assume that SETs events in the clock tree are highly unlikely, and we focus on SEUs in this study. Whereas unexpected resets are captured by the application. The implementation of the IP uses cells hardened by triple modular redundancy (TMR), for the configuration registers and the register bank, to ensure the reliability. The circular buffer uses error-correcting code (ECC) on each memory cell to ensure the consistency of the recorded trace, enabling single error correction and double error detection (SECDED). The data persistence in the buffer is very limited, as it is continuously refreshed. In case of a crash the data are recovered few seconds later from the SoC. Moreover, the SER remains low during beam testing to ensure a correct observability on errors. Considering the previous operating conditions, such a correction is enough to keep data safe in most of the case, especially to prevent double errors. As the primary application domain is spatial, especially geostationary orbit (GEO) applications, the efficiency of the module will be evaluated by exposing the test chip to one of the heavy-ion beam facilities mentioned in Section 1.

Table I
Simulation-based classification results with observation IP coverage and capture rate.

Test status	Rate (%)	Trigger coverage (%)	Buffer capturing (%)
PASSED	87.7	2.0	99.0
FAILED	12.3	57.5	83.1
Calculation	37.5	11.4	64.9
Software trap	6.1	100	100
Fault	27.0	100	90.0
Exit	0.8	62.2	70.2
Timeout	24.4	80.6	72.6

3.1. Hardware module implementation

The module is integrated in a SoC implemented a single test chip. This SoC is based on two 32-bits RISC-V *SCR1* [35] Microcontroller cores designed by Syntacore. The SoC is clocked at 400 MHz, it is mainly composed of a high-speed system bus, 128 KB of ECC protected SRAM, and a low-speed peripheral bus with some IPs. In our test plan, cores are not intended to work together. Only one of them is partially implemented with cells hardened by TMR. The SoC is equipped with a debug unit based on JTAG access, that allows to access the system bus, a dedicated debug bus and the core debug unit. The observation module is mapped on the debug bus of the SoC. It is accessible both from JTAG port and from CPUs themselves, thanks to self-debug features. This allows the firmware to configure the IP at startup, and a JTAG remote controller to interact with it without any intrusiveness at system level. The trace buffer can be connected to one core at a time, the switching is operated by the interface. Fig. 1 illustrates the architecture diagram of the SoC and the mapping of the module.

The JTAG link and some GPIOs configured as a parallel port are the only interfaces used during radiation tests. The JTAG link is also used to load the program in the memory and provides a peek/poke support to any resource of the SoC while system is running. The debug unit of the processor is also accessible through the JTAG port, but we do not yet use this capability when doing radiative tests.

During a run, the firmware should periodically increment a counter in the module memory bank at each benchmark execution. The remote controller is polling this register to check the status. When the status becomes wrong or is no longer refreshed, a failure has occurred. Then the remote controller can dump the content of the circular buffer and the memory mapped registers for a temporal view of the error propagation. Richness of the analysis depends on probed registers. At first, a check of recorded values allows to identify critical errors, while further analysis permits to better understand the cause of the error.

3.2. Dedicated software for beam testing

A dedicated bare-metal application has been developed for the beam experiments. It includes a benchmark algorithm, the fast Fourier transform (FFT), fed by pre-recorded signals. If a single error is detected in the memory, it raises an interrupt to correct the data, and increment a counter which attests the user of the harshness environment. Whereas a double error detected means the end of the execution to avoid memory errors in the experiments. The whole unused memory is filled at startup with error capturing instructions (ECI) [36]. A software memory scrubbing is implemented to ensure that the memory which doesn't contain executable code is not corrupted. Cyclic-redundancy check (CRC) algorithm is used to verify the integrity of the code during the run with either pre-stored syndrome or a syndrome computed at startup as reference. In this manner, the configuration of the IP, the read-only code section and the FFT results are periodically verified thanks to CRC, as well as the memory with the scrubbing. The application is composed of three steps:

- Initialization: Boot procedure, SoC initialization including IP configuration, CRC reference value calculation.
- Benchmarking: Computation of FFT algorithm on pre-recorded signals followed by result checking with CRC.
- Integrity checking: Verification of IP configuration, read-only code thanks to the CRC, and memory scrubbing.
- Status update: Finally, the error code is updated with regards to the previous checks.

Two versions of this application are proposed. One is dedicated for the simulation-based fault injection campaign and the other for the board using the DUT. In the campaign version, only the benchmark and the status update steps are processed one single time. Whereas, in the board version, the benchmark and the Integrity checking steps are executed in a loop until an error is detected.

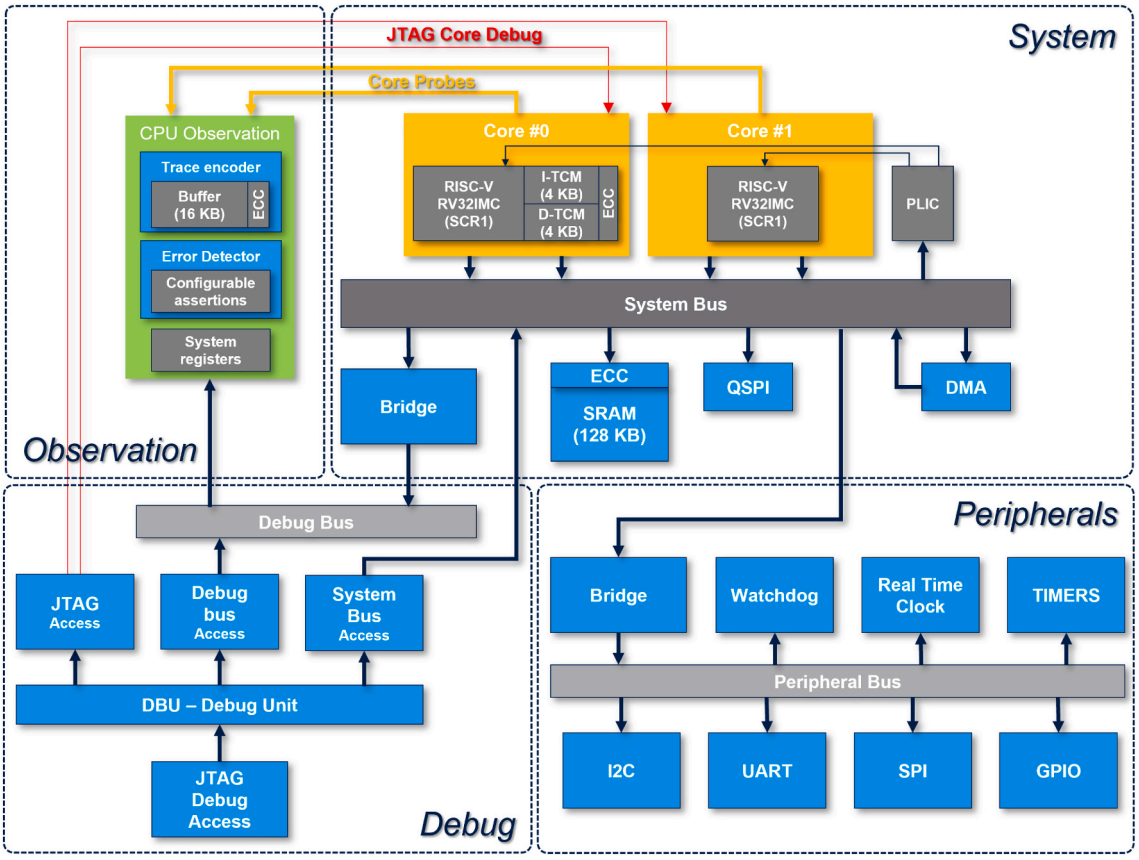

Fig. 1. System on a chip architecture diagram.

When a handled error case is encountered, the memory bank of the IP is filled with CPU status registers, error code, and information related to the error (address and data of corrupted memory, unexpected interruption type, etc.). Then, the CPU is halted to prevent diagnostic data corruption. Finally, the execution trace buffer and the memory bank is dumped from the JTAG remote controller. Here is the list of handled error cases:

- Fault exception: Triggered by CPU error detection mechanisms (instruction address misaligned, instruction access fault, illegal instruction, load/store address misaligned, load/store access fault, etc.)
- Breakpoint: Called when an ECI is erroneously encountered.
- Memory error: ECC unrecoverable memory error. (ECC simple error are corrected and counted).
- Unexpected interrupt: Either hardware or software, information about the interruption cause are dumped in the memory bank.
- Standard exit: Called if a software assertion detect an error and the execution can no longer continue.
- Unexpected reset: More than one reset is not expected in the experiment.
- Code section error: Read-only code section corrupted.
- IP configuration error: IP configuration memory corrupted.

Deadcode sections are also inserted within the benchmark to enhance the error detection capability in case of flow divergence inside the application. This technique is derived from obfuscation techniques [36]. A deadcode section is a macro composed of a series of Error Capturing Instructions (ECIs) started by a jump to a label placed at the end of the macro. Its means that ECIs are not supposed to be executed unless an error occurs. Breakpoint interrupt instruction is used as ECI, as it triggers an interrupt caught by a watchpoint.

To achieve each functionality of software diagnosis, only a modification of the exception handler is required. It doesn't affect the performance. Nevertheless, the deadcode sections spread throughout the benchmark produce a significant number of jumps and reduce the performance up to 10 %.

3.3. *Failure reason capturing IP*

The Fig. 2 presents in detail the architecture of capturing IP, the probes implemented in the RISC-V core as well as the different blocs of the IP: Trace encoder, Error detector and interface.

The failure reason is captured thanks to a **circular buffer** recording the CPU trace and the probed registers at processor clock speed. This buffer works as a FIFO buffer erasing the oldest data in case of memory congestion. The buffer is 1024-events depth protected by SECDED ECC, the size corresponds to 16 KBytes standard trace buffer. As a first implementation on a test chip, the buffer size is large. We propose an optimized size in the Section 4 with still high observation capabilities.

The **trace encoder** is based on the RISC-V Trace specification [37]. It rebuilds the execution flow thanks to a final state machine and probes on PC, instruction register and signals in the pipeline. A jump filter can be activated to avoid tracing all instructions inside basic blocks. Resulting trace events are timestamped by counting clock cycles between each event. Trace events are supplemented by two 32-bits registers that can be chosen from a set of three probed pipeline registers. These registers have been chosen following the fault injection campaign results, described in III-D, in order to monitor resources that are most likely to handle the propagation of errors.

The **error detector** is based on programmable assertions focusing on the monitoring of the PC being aware of the application program, and on the data access addresses to monitor the addressed IPs. It is composed of:

- *Watchpoints* flag an error if the PC matches a given value enabling to catch every fault exception traps.
- *Checkpoints* count when the PC matches a given value and flags an error when the counter reaches a given threshold, enabling to monitor

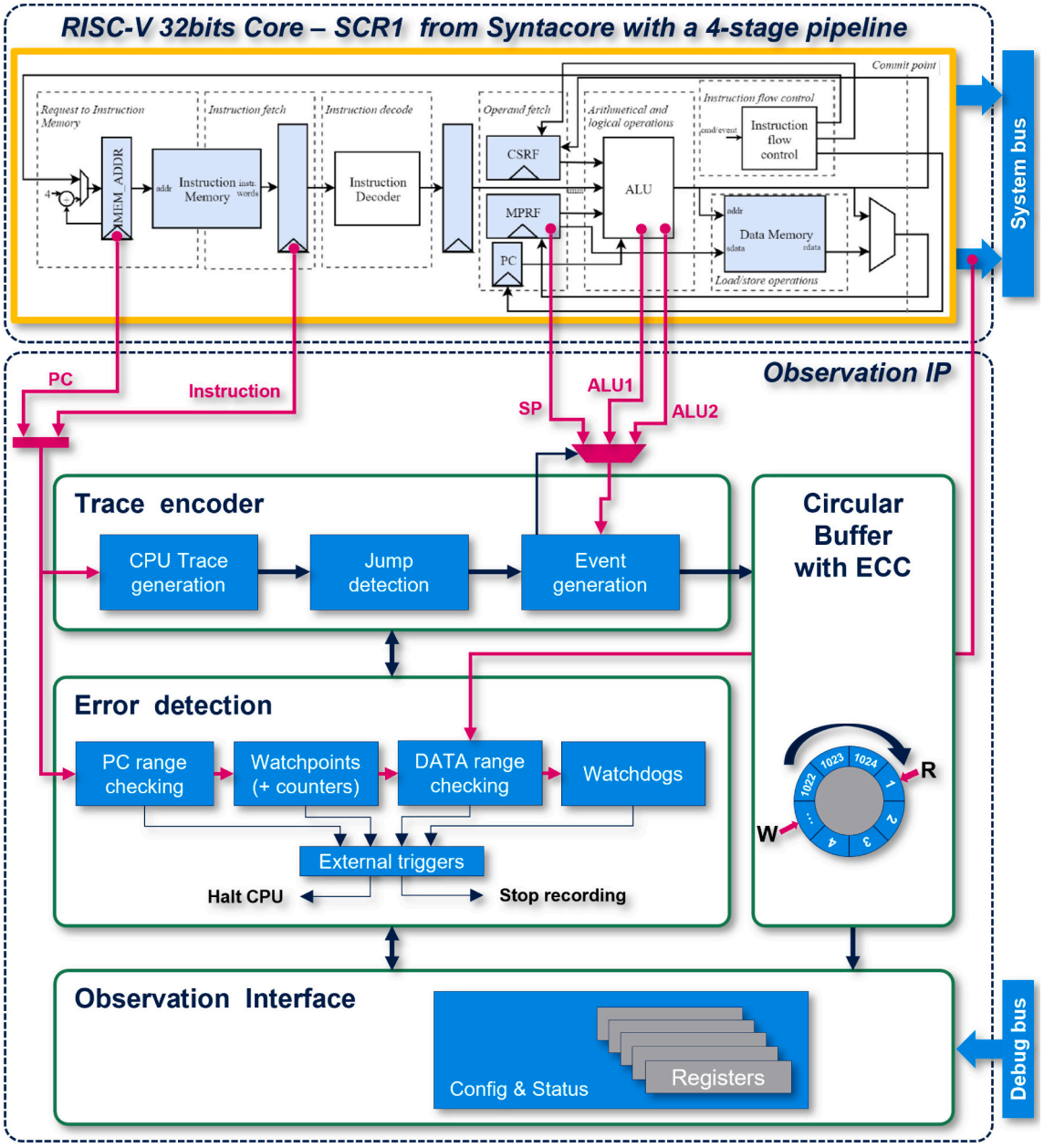

Fig. 2. Functional diagram of the failure reason capturing IP.

the number of accesses to functions.

- *PC range checkers* in which are defined the value ranges, for the purpose of excluding initialization functions. If the PC is in this range, the assertion flags.
- *PC range checker scope* composed of a selection of range checkers to defined a set of ranges of allowed values, it flags an error if the PC is outside this zone, in order to focus on the running application.
- *Data range checker scope* monitors the addresses of the read/write data accesses the same way as for the PC, applicable to a Harvard-type CPU architecture.
- *Data range checker scope* same as PC range checker scope, used to defined allowed address ranges to address certain IPs.
- *Watchdogs* count the clock cycles and flag if the counter reaches a given threshold.

The **external trigger** can be configured to make selected assertions to trigger actions like stop the trace recording or freeze the pipeline to get a snapshot of the error propagation and/or to dump further diagnostic information in the SoC.

3.4. IP development using simulation

Fault injection campaigns carried out in simulation with SEU-like faults injected in the CPU allowed to get metrics about the coverage of the IP in terms of error detectability and capture capability. This type of phenomenon is chosen according to the technology of which the final test chip is made. Some failure modes are highlighted, linking an error detected by the IP with a fault injection point. Simulations are based on the Register Transfer Level (RTL) model of the SoC.

A simulation campaign consists in running simulation runs including a bit flip at a random time. The locations of bit flips cover every flip-flop inside the target. To get better statistics, hundred faults were injected in each flip-flop. The fault injection time window only targets the benchmark application that is run in a loop during beam experiments.

The software executed is based on a bare-metal test harness that monitors calculation tasks. It is composed of a SoC initialization, including the configuration of the IP, then a benchmark is run. This benchmark has been adapted to check the result consistency at the end.

From a run, the following parameters are extracted:

- Execution timing: completed on time, prematurely, delayed or timed out.
- Software outputs: test succeeded, failed or execution handled by a machine trap resulting in a specific error code.
- Error detectability: which probes allowed to detect the error.
- Probe reactivity: time for the injected error to be propagated to the probes.

These results have been exploited to choose best set of core probes that are the most likely to see the propagation of a fault in order to enhance the observability of the error pattern.

From these results, the depth of the buffer can also be estimated as the fault injection time is known, and the fault injection campaign covers almost all failure scenarios. This way, dimensioning the buffer to be able to record the propagation of faults that leads to a detected error, returns to optimize its size.

1) Core probes for error capturing: The first goal of the simulation campaign is aiming at determining the most relevant CPU resources to probe, taking into account the fault detection coverage and the delay of the propagation until the detection. While having a significant error coverage, selection of probes also considers the ability to identify the cause of a failure, that is why we focused on registers whose contents can be interpreted.

The fault injection campaign is preceded by a golden run that creates reference signals from the chosen model probes. For this matter, model probes are spread in the entire core to cover each CPU register as a future probe. During simulation runs with a fault injection, a comparison with a golden run highlights errors introduced by fault injection on the probes. The relevance of observation points in simulation is discussed in Section 4 to choose a set of probes to be recorded into the buffer. Considering a fixed buffer size, if there are more probes, fewer events can be stored. The buffer size is parameter that also needs to be optimized.

2) Error detector configuration: The final goal of the simulation campaign is to estimate the efficiency of the detection capability, and a buffer size enough to capture a majority of errors aiming at optimizing its size. For this purpose, watchpoints are monitoring interrupts expected to occur only in case of errors. Checkpoints are counting the number of accesses to some functions of the benchmark to ensure a coherent behavior of the application. PC checkers are configured to only allow benchmark application and exclude start-up configuring functions. And data-address checkers monitor accesses within the SoC excluding unused IPs.

The external trigger is configured to make every assertion to stop the buffering. Fault exception handler and breakpoint handler are filled with a function to notify that an error has occurred by filling the custom register bank, and definitely stop the processor. Then the user can recover all diagnostic data without risk of corruption due to a running corrupted program.

3.5. Execution trace analysis using machine-learning algorithms

During the fault injection campaign, all the CPU flip-flops are targeted. Only information available from the DUT in real testing condition are recovered. Moreover, as memory is protected by ECC and periodic memory scrubbing, almost all SEUs come only from the CPU, the same ones simulated in the fault injection campaign. Thus, the error cases simulated are very similar to those obtained using beam testing.

To be able to interpret all information on the error propagation recorded in the execution trace buffer, we propose to compare experimental error cases to simulated ones, in order to infer where SEU comes from in the design. For this study, injection points are either CPU architectural registers, or isolated flip-flops. Therefore, bits of registers are grouped to make a single injection point for the inference of the origin of the SEU which makes the CPU to crash. Hence, injection points form classes that a classification algorithm can process. Having a such exhaustive set of CPU failure modes with the campaign enables to train a machine-learning (ML) model to classify failure modes thanks to the execution trace according to the injection point. With regards to the classification accuracy of the model on the training data, it constitutes a reliable metric on the observation capability provided by the IP.

1) Model training using fault injection data: The model training and the accuracy estimation of the model is performed with data from the fault injection campaigns carried out in the same conditions as the experimental setup. Failure modes captured by the IP are selected using the software error code and the status of the embedded error detector. We provide data from several hundreds of faults injected in all flip-flops that compose the CPU to make the training set.

2) Error origin inference: During beam testing, failures provoked by SEUs are also captured following the same error cases handled by the IP and the software. Therefore, in using the captured execution trace and the information from the memory bank, the model is fed by consistent data in relation to the training to perform the classification. Finally, the model is able to infer from which register or isolated flip-flop the SEU that causes a SEFI.

4. Results

4.1. Error propagation and probe specifications

The RISC-V CPU targeted for the SEU fault injection campaign is composed of 2827 flip-flops. To get relevant statistics, 1200 bit-flips have been injected per flip-flops for the campaign, that represents 3.4 M of simulations.

The Fig. 3 depicts the coverage rate and the reactivity of each probe. Indeed, every probe error detection is logged during simulation runs, so the coverage rate for a probe is the number of detections over the number of runs with at least one error detected.

The reactivity depends on the time an SEU takes to propagate errors through the design to the probes. Reaction time is expressed in terms of a rate of the propagation delay to the runtime duration. This reaction time is represented on the Fig. 3 by the orange line with the scale on the right. The coverage is represented by the clustered column with the scale on the right. The probes are sorted in reactivity from highest to lowest. Both probes with red bars are used to generate the CPU trace and the three other probes with green bars can be stored in the buffer by selecting a set of two.

The probe on the result register of the second ALU block (*CPU.PIPE.i_pipe_exu.ialu_sum2_res*) has a high reactivity and coverage because this block is used to calculate PC jumps. The one on the result register of the first ALU block (*CPU.PIPE.i_pipe_exu.ialu_res*) reflects every computation of the CPU, it also provides high coverage and reactivity. CSR registers provide a high reactivity but only cover few error cases corresponding to their functionalities. Finally, the probe on the SP (*CPU.PIPE.i_pipe_mprf.mprf_int* [2]) doesn't have as coverage or as reactivity as other probes but it reflects how the stack is behaving.

1) Failure-mode coverage: The simulation outcomes have been classified to identify the failure modes, illustrated in the Table I. In this study, a run with fault injection is considered as failed if the result is not correct, otherwise we consider that the fault has not affected the application or has been masked. The timeout limit is set to 300 % of the normal execution time.

Description of simulation outcomes in the Table I:

- Calculation: They are identified by the running benchmark and reported at the end of the test. As the circular buffer recording capability doesn't cover the benchmark execution time, in addition to be difficult to detect with our assertions, this case is difficult to capture. However advanced software data checking feature as [32] may detect these errors and trigger the module during execution.
- Software trap: An ECI in a deadcode section or in unused memory area has been executed due to a flow divergence.
- Fault: An error detected by the CPU that triggers an exception

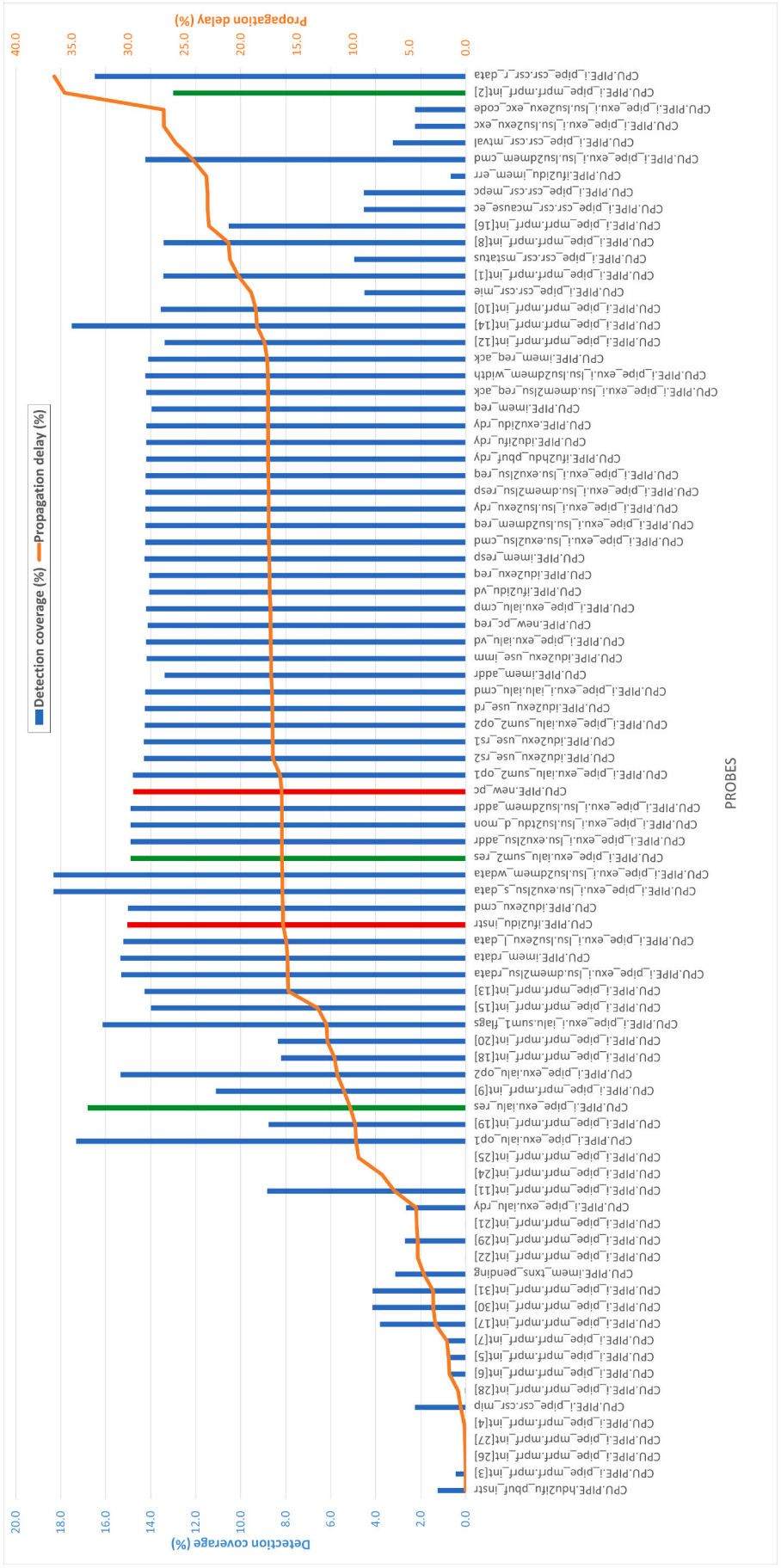

Fig. 3. Probes error coverage rate (%) and reactivity (number of cycles).

because the execution can no longer continue.

- Exit: The standard function exit is called by the running application if an assertion is false. This case is very rare
- Timeout: A case quite difficult to handle because the origin is often unknown. A typical case is an infinite loop, but the origin is often too far from the detection for the circular buffer recording capacity. Watchdogs can also detect this failure.

Table I also highlights the ability of the observation IP to detect errors and capture the fault propagation in the design through the core probes recorded within the buffer. An error is considered as captured if the recording, once stopped due to a violated assertion, covers a period including the first divergence propagated to a probe. Whereas the origin is not always captured, 100 % of the back traces recorded differ from the golden run, thus contains information related to the cause of the crash. Such results constitute the first approach to find an efficient tradeoff of the buffer size.

By achieving such capture rates may imply that the buffer size can be optimized. In many cases the recording is deeper than necessary to capture the fault propagation until its origin. The Fig. 4 shows the capture rate while reducing the buffer depth. As shown on the chart, timeout and calculation error types are the most difficult to detect and to capture. The fault error type is also difficult to capture as it may also result of an error propagation or a corrupted data in the register file. Whereas software traps shall almost all due to a jump directly corrupted by the SEU.

To illustrate the efficiency of the assertions, the Table II depicts the coverage of each assertion which has detected an error and stopped the buffering during the fault run.

The Table II reports that PC scope checking assertions are ahead in terms of coverage compared to other assertions, although data-address checking assertions have also a significant coverage in case of calculation errors or timeout. Checkpoints are more dedicated to the behavior of the application while counting the number of accesses to functions, that is why it gets the best coverage in case of calculation errors and test passed. Watchpoints do not appear in this table as the PC scope overlap the detection reasons. As legal interruptions are not used in our software, they are not allowed by the PC scope.

4.2. Backtrace analysis

The data captured in the buffer are collected at the end of each simulation to enable the investigation of quite exhaustive failure cases and form the fault dictionary database. This database is used as a training set for the ML model. Through the model accuracy estimation, performed on a part of the training set, it provides an observation metric towards the IP implementation.

1) ML model comparison: The Table III present an accuracy comparison of different models. The environment used for this study is the *Scikit-Learn* [38] library in Python.

This table shows that the models Bagging Classifier, Extra-Trees Classifier and K-Nearest Neighbors Classifier are the most suitable to process the data as they provide the best mean accuracy on each class.

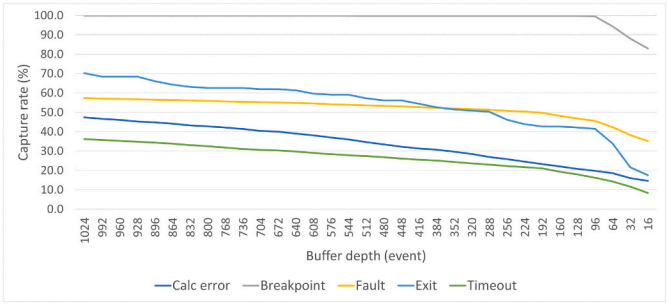

Fig. 4. Capture rate (%) in reducing the buffer depth.

Table II
Assertions efficiency (in %).

Test status	Checkpoints	PC out of scope	Data out of scope
PASSED	1.9	22.4	75.6
FAILED	30.4	37.3	31.8
Calculation	68.1	6.7	25.2
Software trap	0.0	99.1	0.9
Fault	4.1	49.6	46.3
Exit	63.2	33.3	3.5
Timeout	67.0	7.6	24.0

Table III
ML-model accuracy comparison.

ML classification model	Accuracy score
Bagging Classifier	87 %
C-Support Vector Classifier	74 %
Linear classifier (SGD training)	70 %
AdaBoost Classifier	25 %
Extra-Trees Classifier	84 %
Gradient Boosting Classifier	80 %
K-Nearest Neighbors Classifier	82 %

For the following results the Bagging classifier is used.

2) IP parameter optimization: The Fig. 5 highlights the effects of key parameters in order to propose a resource-optimized implementation of the IP with specific settings from which the IP can provide the best performances. On this figure, the accuracy of the model is represented according to the three different set of probes.

First, we can see that the probe sets with which the model performs better are *SP-ALU_RES* and *ALU_RES-ALU2_RES*. Indeed, according to the Fig. 3 the probe *ALU_RES* provide a high error coverage rate with a low mean propagation delay.

Then, we can see that the accuracy of the model is stable until a buffer size of 64 events. That means that the buffer can be reduced to 1 KB without affecting the observability on captured events due to SEU, while providing an accuracy of 87 %.

5. Conclusion and future work

In this work, we present a technique to address microprocessor errors due to radiation induced SEU faults. It helps the investigation of the failure cause in providing detailed diagnostic information from a software and a hardware perspective, without any intrusiveness towards the application. We also use machine-learning classification algorithm to take advantage of both collected data from experiments and a fault dictionary from simulation-based fault injection. This correlation enables the algorithm to find the CPU register where the SEU fault comes from, in case of single-event functional interrupt. Such approach is

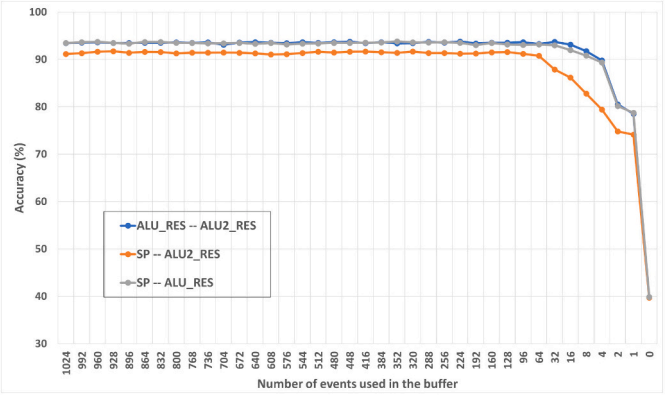

Fig. 5. Bagging classifier model accuracy (%) in reducing the buffer depth.

intended to be used in the context beam testing and device qualification to help the analysis of irradiation campaign data in an efficient way to find potential design weaknesses.

The approach is based on a hardware IP embedded within a SoC, running at the processor frequency, recording continuously its activity. To achieve this functionality, the module combines a trace encoder and an error detector to capture the fault propagation. When an error occurs, the recording is stopped, and the processor halted to keep internal state for further diagnostic. Thereby, it provides a back trace supplemented by CPU architectural register as diagnostic information in case of a crash. The trace is recorded in an embedded circular buffer protected by SECDEC ECC with an optimized size in relation to the application and the processor. The error detector is based on configurable assertions that consist in program counter and data access address range checkers, supported by software traps, to detect flow divergence.

This module is processor agnostic, it is designed to be scalable in terms of buffer size and register probe locations. It has been interfaced to a RISC-V core by probing several key registers and signals, whose location was chosen according to a fault propagation study. In such a configuration, we were able to achieve an accuracy score up to 87 %, with a buffer size of 1 KB. The software requires to handle error cases with dedicated exception handlers and implements software traps to capture legal erroneous jumps.

The developed SoC is implemented in 28 nm Fully-Depleted Silicon-On-Insulator (FDSOI) hardened technology in a test chip embedding a dual RISC-V core. One of them has been partially implemented with cells hardened by TMR and will be used as a reference for the machine-learning SEU location inference.

The chip has been manufactured and will be soon operable with our testbench. To evaluate the approach and validate the setup, we plan to perform Electromagnetic Fault Injection (EMFI) and X-rays irradiations at ID09 beamline of ESRF-France [39]. The ID09 beamline provide a high-energy pulsed X-rays beam of 5 μm in diameter, that allows to scan the SoC and target specific zone to generate SEUs.

As the technology is addressing spatial applications, the test chip is going to be irradiated with heavy-ion particles, this will allow validating the observation module in real situation and identify potential weaknesses of the SoC.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors wish to thank the Syntacore designers for providing the SCR1 core as an open-source and free to use RISC-V core.

References

[1] Norbert Seifert, Sh.ah. Jahinuzzaman, Jyothi Velamala, Ricardo Ascazubi, Nikunj Patel, Balkaran Gill, Joseph Basile, Jeffrey Hicks, Soft error rate improvements in 14-nm technology featuring second-generation 3d tri-gate transistors, IEEE Trans. Nucl. Sci. 62 (6) (2015) 2570–2577.
[2] Vinay Vashishtha, Lawrence T. Clark, Srivatsan Chellappa, Anudeep R. Gogulamudi, Aditya Gujja, Chad Farnsworth, A soft-error hardened process portable embedded microprocessor, in: 2015 IEEE Custom Integrated Circuits Conference (CICC), 2015, pp. 1–4.
[3] R. Ginosar, Survey of Processors for Space, European Space Agency, (Special Publication) ESA SP, 2012, 701, 01.
[4] M. Pignol, Dmt and dt2: two fault-tolerant architectures developed by cnes for cots-based spacecraft supercomputers, in: 12th IEEE International On-Line Testing Symposium (IOLTS'06), 2006, p. 10, pp.–.
[5] N. Oh, E.J. McCluskey, Error detection by selective procedure call duplication for low energy consumption, IEEE Trans. Reliab. 51 (4) (2002) 392–402.
[6] George A. Reis, Jonathan Chang, David I. August, Automatic instruction-level software-only recovery, IEEE Micro 27 (1) (2007) 36–47.
[7] José Rodrigo Azambuja, Samuel Pagliarini, Mauricio Altieri, Fernanda Lima Kastensmidt, Michael Hubner, Jürgen Becker, Gilles Foucard, Raoul Velazco, A fault tolerant approach to detect transient faults in microprocessors based on a non-intrusive reconfigurable hardware, IEEE Trans. Nucl. Sci. 59 (4) (2012) 1117–1124.
[8] Corrado De Sio, Sarah Azimi, Andrea Portaluri, Luca Sterpone, Seu evaluation of hardened-by-replication software in risc- v soft processor, in: 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2021, pp. 1–6.
[9] M. Grosso, M.Sonza Reorda, M. Portela-Garcia, M. Garcia-Valderas, C. Lopez-Ongil, L. Entrena, An on-line fault detection technique based on embedded debug features, in: 2010 IEEE 16th International On-Line Testing Symposium, 2010, pp. 167–172.
[10] M. Portela-Garcia, M. Grosso, M. Gallardo-Campos, M.Sonza Reorda, L. Entrena, M. Garcia-Valderas, C. Lopez-Ongil, On the use of embedded debug features for permanent and transient fault resilience in microprocessors, Microprocess. Microsyst. 36 (5) (2012) 334–343. Special Issue on Design of Circuits and Integrated Systems.
[11] José Rodrigo Azambuja, Mauricio Altieri, Jürgen Becker, Fernanda Lima Kastensmidt, Heta: Hybrid error-detection technique using assertions, IEEE Trans. Nucl. Sci. 60 (4) (2013) 2805–2812. Accessed: 2021-11.
[12] Luis Parra, Almudena Lindoso, Marta Portela, Luis Entrena, Felipe Restrepo-Calle, Sergio Cuenca-Asensi, Antonio Martínez-Alvarez, Efficient mitigation of data and control flow errors in microprocessors, IEEE Trans. Nucl. Sci. 61 (4) (2014) 1590–1596.
[13] L. Parra, A. Lindoso, M. Portela-Garcia, L. Entrena, B. Du, M. Sonza Reorda, L. Sterpone, A new hybrid nonintrusive error-detection technique using dual control-flow monitoring, IEEE Trans. Nucl. Sci. 61 (6) (2014) 3236–3243.
[14] A. Lindoso, L. Entrena, M. García-Valderas, L. Parra, A hybrid fault-tolerant leon3 soft core processor implemented in low-end sram fpga, IEEE Trans. Nucl. Sci. 64 (1) (2017) 374–381.
[15] Radiation effects facility (radef) - university of jyväskylä - finland. www.jyu.fi/science/en/physics/research/infrastructures/accelerator-laboratory/radiation-effects-facility.
[16] Cyclotron of louvain-la-neuve (cyclone) - uclouvain - belgium. uclouvain.be/en/research-institutes/irmp/crc/applications-technologiques.html.
[17] Jong Chul Lee, Andrew S. Gardner, Roman Lysecky, Hardware observability framework for minimally intrusive online monitoring of embedded systems, in: 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, 2011, pp. 52–60.
[18] K. Peterson, Y. Savaria, Assertion-based on-line verification and debug environment for complex hardware systems, in: 04 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512) volume 2, 2004, pp. II–685.
[19] M. Abramovici, A reconfigurable design-for-debug infrastructure for socs, in: 2006 43rd ACM/IEEE Design Automation Conference, July 2006, pp. 7–12.
[20] V. Easwaran, et al., A unique non-intrusive approach to non-ate based cul-de-sac soc debug, in: 2014 27th IEEE International System-on-Chip Conference (SOCC), 2014, pp. 336–339.
[21] H.F. Ko, Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging, in: 2010 15th IEEE European Test Symposium, May 2010, pp. 62–67.
[22] J.M. Mogollón, J. Nápoles, H. Guzmán-Miranda, M.A. Aguirre, Real time seu detection and diagnosis for safety or mission-critical ics using hash library-based fault dictionaries, in: 2011 12th European Conference on Radiation and Its Effects on Components and Systems, 2011, pp. 705–710.
[23] A. Lindoso, L. Entrena, M. Garcia-Valderas, S. Philippe, Y. Morilla, P. Martin-Holgado, M. Peñ-Fernandez, Ptm-based hybrid error-detection architecture for arm microprocessors, Microelectron. Reliab. 88-90 (2018) 925–930, 29th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF 2018).
[24] M. Peña-Fernandez, A. Lindoso, L. Entrena, M. Garcia-Valderas, Y. Morilla, P. Martín-Holgado, Online error detection through trace infrastructure in arm microprocessors, IEEE Trans. Nucl. Sci. 66 (7) (2019) 1457–1464.
[25] M. Peña-Fernandez, A. Lindoso, L. Entrena, M. Garcia-Valderas, The use of microprocessor trace infrastructures for radiation-induced fault diagnosis, IEEE Trans. Nucl. Sci. 67 (1) (2020) 126–134.
[26] M. Peña-Fernández, A. Lindoso, L. Entrena, I. Lopes, V. Pouget, Microprocessor error diagnosis by trace monitoring under laser testing, IEEE Trans. Nucl. Sci. 68 (8) (2021) 1651–1659.
[27] Coresight debug and trace - arm. https://developer.arm.com/ip-products/system-ip/coresight-debug-and-trace. Accessed: 2021-11.
[28] Risc-v specifications - risc-v international. https://riscv.org/technical/specifications/. Accessed: 2021-11.
[29] Global embedded processor debug interface - nexus 5001 forum standard. https://nexus5001.org/nexus-5001-forum-standard/. Accessed: 2021-11.
[30] Debug and trace applications - mipi alliance. https://www.mipi.org/specifications/debug. Accessed: 2021-11.
[31] Microprocessor development tools - lauterbach. https://www.lauterbach.com/frames.html?home.html. Accessed: 2021-11.
[32] B. Du, et al., Online test of control flow errors: a new debug interface-based approach, IEEE Trans. Comput. 65 (6) (2016) 1846–1855.
[33] Hai Tao Sun, First failure data capture in embedded system, in: 2007 IEEE International Conference on Electro/Information Technology, 2007, pp. 183–187.
[34] Taiki Uemura, Byungjin Chung, Jeongmin Jo, Hai Jiang, Yongsung Ji, Tae-Young Jeong, Rakesh Ranjan, Youngin Park, Kiil Hong, Seungbae Lee, Hwasung Rhee, Sangwoo Pae, Euncheol Lee, Jaehee Choi, Shota Ohnishi,

Ken Machida, Investigating of ser in 28 nm fdsoi-planar and comparing with ser in bulk-finfet, in: 2020 IEEE International Reliability Physics Symposium (IRPS), 2020, pp. 1–5.
[35] Scr1 microcontroller core - syntacore, in: https://syntacore.com/page/products/processor-ip/scr1. Accessed: 2020-11.
[36] G. Miremadi, Two software techniques for on-line error detection, in: 1992 FTCS - The Twenty-Second International Symposium on Fault-Tolerant Computing, IEEE Computer Society, Los Alamitos, CA, USA, jul 1992, pp. 328–335.
[37] Panesar Gajinder, Efficient trace for risc-v, version 1.1.2-draft. Technical Report be029f6479b0326c3fdff41927f83ac45f9bdc85, Mentor Graphics, a Siemens Business, Dec 2020.
[38] scikit-learn - machine learning in python. https://scikit-learn.org/stable/. Accessed: 2020-11.
[39] Europeam synchrotron radiation facility (esrf) id09 beamline - grenoble - france. https://www.esrf.fr/home/UsersAndScience/Experiments/CBS/ID09.html.