DOI: 10.1109/tns.2016.2636574

A hybrid fault-tolerant LEON3 soft core processor implemented in low-end SRAM FPGA

A. Lindoso, L. Entrena, M. García-Valderas, L. Parra

Abstract—**In this work we implemented a hybrid fault-tolerant LEON3 soft-core processor in a low-end FPGA (Artix-7) and evaluated its error detection capabilities through neutron irradiation and fault injection in an incremental manner. The error mitigation approach combines the use of SEC/DED codes for memories, a hardware monitor to detect control-flow errors, software-based techniques to detect data errors and configuration memory scrubbing with repair to avoid error accumulation. The proposed solution can significantly improve fault tolerance and can be fully embedded in a low-end FPGA, with reduced overhead and low intrusiveness.**

Index Terms—**Neutron cross-section, microprocessors, SEEs, soft errors, fault tolerance, hybrid fault-tolerance techniques**

I. Introduction

FPGAs are becoming increasingly attractive for a broad range of applications, including those that require a high reliability. In comparison with ASICs, they provide higher flexibility, lower cost, reduced time-to-market and even the capability of upgrading hardware functionality. Moreover, FPGAs are usually implemented in the latest available technologies and applications can take the benefits of them in a shorter time. Today, there is a wide offer of devices with several cost-performance tradeoffs [1], but even low-end devices have a huge amount of resources, enough to implement Systems-on-Chip (SoCs) with good performance at low cost. SoCs can use either hard-core processors [2] for higher performance or soft-core processors for higher flexibility [3], [4]. The former require specific FPGA devices that include hard-core processors, while the latter can be implemented in any FPGA.

Embedded systems based on soft-core processors are becoming very popular, even though they cannot match the performance, area, and power of hard-core processors. Due to shrinking technology sizes, soft-cores have increased commercial use in networking and data centres. Soft-core processors are flexible and can be customized for a specific application with relative ease. Moreover, they can be re-targeted to any new technology as soon as it becomes available and therefore they are less prone to become obsolete.

For critical applications, soft-core processors need to be made fault-tolerant. Unlike hard-core processors, soft-core processors can be modified to implement error mitigation, which is another advantage of soft-core processors. Conventional hardware error mitigation techniques, such as Triple Modular Redundancy (TMR) [5] can be used for this purpose. However, TMR usually involves large overheads and it is highly intrusive. In the case of FPGAs, although it is widely used, it requires a very careful design, because a single SEU affecting the configuration memory may originate multiple errors [37]. In addition, designers often have very limited access to the internal architecture of the soft-core processor. Alternatively, software or hybrid error mitigation techniques can be used. These techniques typically focus on error detection rather than correction to reduce the overheads [6-15].

Software-based techniques provide high flexibility, low development time and low cost, as they can be implemented without modifying the hardware. However, software-based techniques cannot achieve full system protection against soft errors [6] and may produce large overheads in processing time and storage needs, particularly when designed to protect the microprocessor against control-flow errors [7], [8].

Hybrid nonintrusive techniques typically use a hardware module to monitor the execution of the instructions in the processor from an available interface, such as the memory bus [7]-[10] or the trace interface [11]-[15]. This approach does not require any modification of the processor, except for possibly adapting the monitoring interface. A hardware monitor can be designed to detect several types of errors. Generally, control-flow errors can be easily detected by checking the sequence of executed instructions, whereas data errors are more complex to check as it requires large amounts of data to compare with [15].

Our goal in this work is to provide a solution that can be fully embedded in a low-end FPGA without the need of external hardware, with reduced overhead and with low intrusiveness. To achieve this goal, we combine a selection of techniques. Hardware changes are restricted to the hardening of memories and the use of the hardware monitor proposed in [14]. Memories are usually one of the most critical components [36] in an embedded system, but they can be

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under contract ESP2015-68245-C4-1-P.

A. Lindoso, L. Entrena, M. García-Valderas and L. Parra are with the University Carlos III of Madrid, Electronic Technology Department, Avda. Universidad, 30, Leganes (Madrid), Spain. (e-mails: alindoso@ing.uc3m.es, entrena@ing.uc3m.es, mgvalder@ing.uc3m.es and lparra@pa.uc3m.es).

easily identified and protected using Error Detection And Correction (EDAC) codes. The hardware monitor compares the instruction flow captured upstream at the bus between the memory and the microprocessor with the instruction flow captured downstream at the trace interface after the instruction has been executed to detect control-flow errors. The major advantage of this approach is that it can detect all control-flow errors with no performance degradation and a small hardware overhead. Then, software-based error detection techniques are applied to detect data errors. Finally, scrubbing is used to avoid the accumulation of errors. To this purpose, we take advantage of the Soft Error Mitigation (SEM) Core from Xilinx [16] that is included in the latest FPGA families.

A major advantage of the proposed approach is that the applied hardware techniques do not require a deep knowledge of the processor architecture or the FPGA architecture. As a matter of fact, the hardened memory blocks and the hardware monitor are implemented as soft IP (Intellectual Property) modules that can be reused for other systems.

To demonstrate the benefits of the proposed approach, we implemented a hybrid fault-tolerant LEON3 soft-core processor [35] in an Artix-7 FPGA and experimentally evaluated the SEU sensitivity by neutron irradiation and fault injection. Because errors may be detected by more than one of the used techniques, the analysis is performed in an incremental manner to show the incremental benefits that can be obtained as additional techniques are applied.

This paper is organized as follows. Section II summarizes the related work. Section III describes the proposed hybrid fault-tolerant approach. Section IV describes the experiments that have been performed to validate this approach and presents the experimental results. Finally, section VI summarizes the conclusions of this work.

II. Related Work

Soft error mitigation for microprocessors has received much attention in the past years, as microprocessors play a very important role in many electronic systems that require a high reliability. In the literature [14], [17], [18], error mitigation in microprocessor-based systems can be divided into three main types of techniques: hardware techniques, software techniques and hybrid techniques. Most of these techniques can be applied to both hard-core and soft-core processors. However, the reliability of soft-core processors implemented in SRAM-based FPGAs in the presence of SEUs is not well understood yet [19].

Software-based techniques are very attractive because only software is modified, which is generally simpler than modifying the hardware. In the case of soft-cores, designers often have very limited knowledge about the internal details of the core, so software-based techniques are preferred. However, this type of techniques usually involve significant code size enlargement and performance decrease.

Software-based techniques can be divided into control-flow techniques and data-flow techniques. Control-flow techniques focus on errors regarding the program flow. Commonly, these techniques divide the program into basic blocks and use signatures [20] or assertions [21] to check for incorrect changes in the control flow.

Basic blocks are blocks of code without branches (i.e., the instructions of a basic block are executed sequentially without taking decisions which affect the program flow). Signature techniques provide a signature for the basic blocks of a program at compilation time. When the program is executed, a run-time signature is computed and compared with the one generated at compilation time. Whenever both signatures differ, an error is detected. Assertion techniques modify the software to insert special instructions named assertions which assert the beginning and end of basic blocks.

The most popular data-flow software techniques are based on duplication [15], [22], [31] or assertions [23]. Duplication techniques duplicate data and check the consistency of variables and their copies. To reduce code size and performance overheads, duplication and consistency checks can be made at different levels or with different grades of error coverage [18]. Regarding the granularity, duplication can be accomplished at instruction, procedure or program level. A coarser granularity may reduce the overheads but latency will increase. Regarding the coverage, variables can be selected to decide which of them should be protected or checked. For instance, in the Final Variables technique [24] only certain variables are checked to reduce code size and performance overheads.

Data-flow assertion techniques use assertions to check data validity and correctness. The main drawbacks of these techniques are that they are usually application dependent and that error coverage depends on the ability to place the assertion in the most effective code location [23].

Hardware-based techniques modify the architecture or the hardware implementation. These techniques may be quite effective, but they are often expensive in terms of FPGA resource utilization and power consumption [5]. In addition, they are highly intrusive because they require a deep modification of the hardware and must be carefully applied.

At first glance, hardware-based techniques may appear to be an attractive mitigation approach for soft-core processors. However, designers often lack enough knowledge about the internal architecture of the processor and the FPGA to make an effective hardening. Alternatively, general hardware solutions based on Triple Modular Redundancy (TMR) can be used, which can be implemented with the help of some tools, such as TMRTool from Xilinx [25]. These tools are generally intended for full mitigation and produce high overhead. For low error rates, partial mitigation can be used to reduce the overhead. However, partial mitigation techniques for FPGAs require a complex analysis [26], [27].

Hybrid techniques benefit from the best characteristics of both hardware and software techniques. They usually combine some level of redundancy in both software and hardware. In some cases, special hardware is introduced to accelerate the checking of redundant computations [28]. Alternatively, some error checking tasks are implemented through an external hardware module, sometimes known as a watchdog processor [7]-[15], [29]. The watchdog processor may be designed with different capabilities and complexities. It can be used to just verify signatures or assertions stored internally or produced by the processor, or it can even consist of a simplified processor that executes a program concurrently with the main processor. Hybrid approaches using an external hardware module reduce intrusiveness and can be applied to any processor as long as a suitable monitoring interface is available. This is not always possible for hard-core processors, because the most appropriate interfaces may not be accessible from external pins, but it is usually feasible for soft-core processors.

III. Hybrid Fault-Folerant Approach

The proposed hybrid mitigation approach builds on several existing techniques to provide an effective solution for soft-cores implemented in low-end FPGAs. Namely, it combines the use of hardened memories, a hardware monitor to detect control-flow errors [14], software-based techniques to detect data errors [9], [31] and configuration memory scrubbing. Each of these techniques is described in more detail in the following sections.

A. *Hardening of memories*

A microprocessor-based system typically includes several memories. In addition to RAM and ROM, which can be implemented on chip or off chip, caches are commonly used to speed up memory access. Caches should be implemented on chip for fast access. Finally, the register file can be implemented with flip-flops or with RAM blocks. In the case of the LEON3 processor, the register file is quite large because it uses register windows, so it is more efficiently implemented with RAM blocks.

Memories are usually protected using EDAC codes. In addition, memory scrubbing is often employed to prevent a memory from accumulating errors in a single word, which could eventually defeat EDAC [30]. In our design, RAM, ROM and caches were hardened using an in-house implementation of Single Error Correction / Double Error Detection (SEC/DED) code with scrubbing. For memory scrubbing, we took advantage of the dual-port featured by FPGA BRAMs (Block RAMs). Thus, one port was used for regular operation while the other was used for continuous scrubbing. Scrubbing is continuously performed every clock cycle, so the scrubbing cycle time is 1 clock cycle times the memory size. Single errors were corrected without notice, while double errors were reported.

The register file was implemented using BRAMs, but it makes use of dual ports. In this case, we just implemented a SED (Single Error Detection) approach using duplicated register files. This way we can have a report of the errors in the register file. Errors in the register file could be masked by using TMR, although the impact is moderate because the register file uses just a few BRAMs.

B. *Hardware monitor*

In [14] a general nonintrusive mitigation technique was proposed to harden microprocessors against control-flow errors for ASIC technologies. This approach adds a small hardware module, called Hardware Monitor (HM), to the microprocessor architecture that observes the instruction flow at the beginning and at the end of the microprocessor data path, namely, at the fetch stage, through the memory bus, and after execution, through the trace interface. From each of these observations points, the address of the instruction, given by the Program Counter (PC), and the instruction code (opcode and operands), are collected. Then, the HM compares the information collected from these observation points and raises an error signal if they do not match.

In addition, the HM can predict the next PC value from the current instruction and the current PC value, which can be observed from the trace interface. PC prediction is based on the analysis of the instruction opcode. The PC must be incremented by the instruction size except for branch instructions. If the opcode corresponds with a branch instruction, the PC must be incremented either by the branch offset if the branch is taken or by the instruction size if the branch is not taken. Any change in the PC value that does not meet these requirements can be considered an error. To this purpose, the predicted PC value is then compared with that of the next executed instruction to detect abnormal changes in the control flow.

Finally, exceptions and traps are also checked thanks to the information provided by the trace interface. Unexpected traps, such as those produced by invalid instructions or memory addresses, are considered errors. Otherwise, if a corrupted instruction executed without a trap, it will probably produce an error that will be caught later on through software checking. In addition, memory hardening protects data and instructions while in memory. A basic block diagram of the architecture of the hardware monitor is provided in Figure 1.

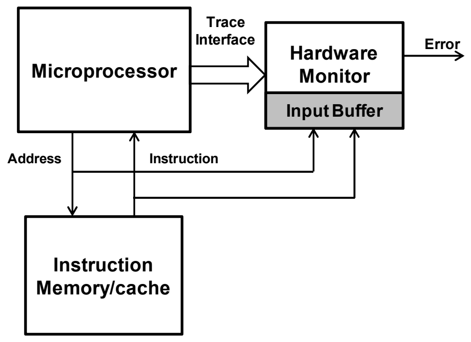

Figure 1. Block diagram of the hardware monitor

This technique has several advantages. First, it is a general nonintrusive technique that can be applied to any hard-core or soft-core processor, and particularly to pipelined processors with several pipeline stages. It is able to detect all control-flow errors and it uses a low amount of resources because it does not need to store information about the expected control flow. Finally, it does not reduce performance, because the processor is observed in a nonintrusive manner and it does not require any software support.

C. *Software-based fault tolerance*

Data errors are not covered by the hardware monitor and software-based techniques must be used for this purpose. Many different software-based approaches can be found in the literature [18]. Among them, we have selected data-flow duplication techniques [31] and Inverted Branches technique [9] because they have been shown to be effective for data errors and are relatively easy to implement.

Our approach focuses on maximizing the error coverage with the smallest error detection latency. Modifications are performed directly in high-level language (C code) instead of modifying assembly code as proposed in [9]. This approach provides a portable solution for any architecture. Compilation is made with the –O2 optimization option, which is able to optimize the code without breaking software redundancy. The applications used in this work are bare codes which do not use libraries and modifications were made manually, but they can easily be automated and embedded in a compiler. In the general case, software libraries should be hardened as well.

To harden the software, we apply the following rules:

1. All variables are duplicated. This rule applies to loop indices and procedure parameters as well.
2. Variable copies are updated whenever a variable is modified.
3. Consistency checks are performed whenever variables and copies are modified. They are performed just after variable modification to decrease error detection latency. This rule applies also to loop indices.
4. Consistency checks of all procedure parameters are performed just after a procedure call.
5. All conditional branches and loops are hardened with the Inverted Branches technique [9]. From the control-flow point of view, branches imply two possible correct paths. The choice depends on the evaluated condition. Each condition is evaluated twice to make sure that the choice made was right. When the branch is taken, the branch is repeated with an inverted condition. When the branch is not taken, the branch is simply repeated. In both cases, if the branch is taken the second time, an error has occurred. This technique was proposed for code modification in assembly. However, we have applied it directly in high-level code.

Software hardening techniques do not cover certain specific architectural parts that cannot be accessed from software. In most processor architectures there are some internal or shadow registers which are not part of the programming model and therefore cannot be directly accessed by the software. An example of them is the internal pipeline registers. However, errors in the pipeline registers can be detected by the HM. This drawback is present in all software-based techniques and its effects depend mainly on the complexity of the utilized architecture. However, thanks to our hybrid approach we can achieve high error detection coverage by the combined use of software and hardware error mitigation techniques.

D. *Configuration memory scrubbing*

Modern SRAM-based FPGAs utilize a large amount of configuration memory (CRAM) which accounts for the vast majority of memory cells in a device. CRAM is highly critical as it can alter the function of the logic and the interconnections. Contrary to application memory cells, whose contents may be refreshed with new data during normal operation, errors in the CRAM remain until the device is reconfigured, so that errors may accumulate and eventually lead to multiple faults that can break the redundancy.

A SEU correction mechanism becomes essential to avoid accumulation of latent faults and ensure correct operation of FPGA devices. CRAM scrubbing is a widely used solution to avoid error accumulation in the FPGA configuration memory. It consists in the periodic refresh of the configuration while the FPGA is operating, and it can be implemented either internally or externally to the device [32]. Because of the increasing demand of solutions to this problem, FPGA manufacturers provide support for CRAM scrubbing in the most recent FPGA families. In particular, Xilinx provides the SEM Controller IP starting from the 7 series. This module provides soft error detection and correction capabilities.

The SEM Controller supports three different error correction modes [16]:

- Repair: ECC (Error Correction Code) algorithm-based correction
- Enhanced Repair: ECC and CRC (Cyclic Redundancy Check) algorithm-based correction
- Replace: Data reload based correction

In our case, we used the Enhanced Repair mode, because it provides higher error correction capabilities, namely correction of CRAM frames with single-bit errors or double-bit adjacent errors. The Replace mode supports correction of CRAM frames with arbitrary errors, but requires partial reconfiguration of the FPGA.

The SEM Controller features also a fault injection capability, which is very convenient to implement fault injection experiments. To this purpose, we included a fault injection controller that can be enabled through an external pin. This way we ensure the same design is used for radiation testing and fault injection. Fault injection is disabled when testing under the beam, but it can be enabled when testing in the lab.

IV. Experimental Results

To test the benefits of the proposed hybrid approach with soft cores and low-end FPGAs, a neutron radiation experiment was performed at Los Alamos National Laboratory's (LANL) Los Alamos Neutron Science Center (LANSCE) in December 2015. LANSCE provides a white neutron source that emulates the energy spectrum of the atmospheric neutron flux. As the neutron radiation experiments require a long exposure time, fault injection was used to complement the analysis for several software benchmarks. In the following section the hardware and software setup is described. Then, the results of the tests are reported.

A. *Experimental setup*

The experiments were run on an Artix-7 XC7A100T FPGA from Xilinx, in which we implemented two instances of the fault-tolerant LEON3 design. These two instances are totally independent of each other and were implemented just to speed up the test. The selected configuration was a basic one, using data and instruction caches of 2kB each, 8 register windows for the register file, 128 kB of RAM and 4kB of ROM, all embedded in the FPGA. Table I shows the synthesis results for each LEON3 core, for each hardware monitor (HM) and for the SEM Controller, which is only instanced once.

TABLE I. SYNTHESIS RESULTS

	#LUTs	#FFs	BRAM36
LEON3	4654	2013	47.5
HM	522	431	0
SEM	781	586	4.5

The utilization of FPGA logic resources is relatively low (24% of slices). However, the implementation of RAM, ROM, cache memories and the register file used up to 74% of the available BRAMs. Therefore, we could not add more instances of the LEON3 soft core due to the memory limitations.

The HM is not protected, because an error in the HM does not affect the computation. From this point of view, we can consider that the hardware monitor does not increase the cross section, which is due to the undetected errors. At most, the HM can detect some false positives (the HM signals an error but the computation is correct). False positives may trigger some unnecessary error recovery action. For low error rates, the impact of some sporadic error recovery action is negligible. Otherwise, the hardware monitor can be hardened to reduce the chance of false positives.

In addition, we included the SEM core in Enhanced Repair mode. This core is able to correct errors that appear in the configuration memory of the FPGA. The SEM provides scrubbing capabilities without needing a scrubbing module outside the FPGA, so that the whole system, including error mitigation hardware, is embedded in the FPGA.

Three different software benchmarks were tested: quicksort, matrix multiplication (MMULT) and AES (Advanced Encryption Standard) encryption. Quicksort consists in a recursive implementation of the quicksort algorithm with an array of 5000 integers. It specifically tests recursive calls and uses a significant amount of data memory. This benchmark was selected for compliance with current efforts towards a common set of benchmarks that can be used for comparison among different experiments [33]. The MMULT software benchmark implements a 5x5 matrix multiplication, mainly using loops and arithmetic operations. The AES benchmark implements the AES encryption algorithm with a key length of 256 bits and 10 iterations. This last benchmark makes a more intensive use of logical operations and data memory to get encryption look-up table data.

The selected software benchmarks also have very different execution times. Table II shows the maximum number of clock cycles required to execute each benchmark for the unhardened and hardened software versions, respectively. At the nominal clock frequency (50 MHz), the execution times vary between 0.14 ms for the MMULT benchmark and 47 ms for the hardened quicksort benchmark. The hardened software versions produce a large performance overhead because they check for errors after every operation in order to reduce the error detection latency. Generally, this overhead can be reduced by checking less frequently at the expense of larger error detection latency.

TABLE II. CLOCK CYCLES FOR THE EXECUTION OF THE SOFTWARE BENCHMARKS

SW Benchmark	Unhardened	Hardened	Performance Overhead
Quicksort	937,376	2,343,943	2.50x
MMULT	7,022	19,948	2.84x
AES	31,137	97,514	3.13x

All tests were run with the same hardware and only differ in the software application stored in RAM. Thus, all results are referred to a common reference design. To properly relate fault injection results to neutron radiation results, this reference design includes RAM protection because errors in RAM cannot be tested with fault injection. Upon reset, the processor executes the boot program and then it repeatedly executes the application software in an infinite loop. The results are checked with respect to a golden, precomputed result after each execution.

Our experimental setup stores the errors reported by the various checkers in a register. The error register is updated whenever an error appears and can only be cleared by reconfiguring the FPGA. A suitable timeout condition is set so that an error is also reported in the case the software execution does not finish within a short margin of the expected time. Note that the classification of errors is based on the means used to detect them. We do not know the location of the errors nor intend to draw conclusions about the sensitivity of the components of the system, but about the system as a whole. The contents of the error register are sent through a Serial Peripheral Interface (SPI) to a host in order to store all the errors and classify them. Every time an error is reported, the host reconfigures the FPGA. The error collection module and the SPI interface were tripled in order to reduce the impact of possible errors in these modules on the obtained measures.

B. *Neutron radiation test results*

For the neutron radiation test, the LEON3 processor was executing the quicksort algorithm, which is the benchmark with a longer execution time. Only the unhardened software version was tested due to the limited beam time.

The experiment was run over several days with a total fluence of $4.7{\cdot}10^{11}$ n/cm^2. Table III shows a summary of the number of errors and the cross-section as we incrementally apply different techniques. To this purpose, we use the report of errors detected by each technique, and consider that these errors would be detected if the technique was applied, but would go undetected otherwise. Thus, the number of undetected errors and the cross-section decrease as more hardening is applied.

The first row corresponds to the tested implementation and considers all errors observed. Note that in this case the register file is not protected, but the other memories are protected by single error correction and scrubbing. The size of the protected memory is 2.1Mbit, which according to the sensitivity data provided by the manufacturer [38], results in a cross-section of 1.42×10^{-8} cm^2. This is much larger than the measured cross-section and demonstrates that the hardening of memories is crucial to reduce it. On the other hand, it is necessary to exclude errors in the memories in order to properly relate neutron irradiation results to fault injection results, because memories cannot be fault injected with our approach. The second row shows the result when the register file (RF) is protected. The cross section of the register file according to manufacturer's data is 1.1×10^{-10} cm^2, which is in line with the obtained reduction taking into account that not all registers are used all the time. Finally, the last row shows the cross-section when the hardware monitor (HM) is used.

The HM can additionally detect 44 (46%) of the observed errors. Excluding the errors in the register file and the memories, the HM can detect 66% of the remaining errors, which is in agreement with the results in [14]. Additional improvements can be obtained by using software-based error detection techniques, as we will show in the next section.

TABLE III. RADIATION TEST RESULTS

Hardening approach			Undet. errors	Cross-section (cm^2)	Relative Imp.
Memories	RF	HM			
SEC+ scrubbing	No	No	95	2.02×10^{-10} (1.6×10^{-10}, 2.4×10^{-10})	1.00x
SEC/DED + scrubbing	SED	No	67	1.43×10^{-10} (1.1×10^{-10}, 1.8×10^{-10})	1.42x
SEC/DED + scrubbing	SED	Yes	23	4.89×10^{-11} (2.9×10^{-11}, 6.9×10^{-11})	4.13x

C. Fault injection results

Due to the long exposure times required by neutron irradiation experiments, it is not generally possible to test many different software benchmarks and software mitigation solutions. To this purpose, the neutron irradiation experiments were complemented by fault injection.

Fault injection is commonly used to evaluate and validate the robustness of FPGA designs, although it is not as accurate as radiation testing due to several reasons [19], [27], [34]. First of all, access to internal FPGA resources is limited and it strongly depends on the interfaces provided by the manufacturer. Some internal resources cannot be fault injected as they are not accessible to the user. However, faults in the CRAM can be injected by changing the configuration bitstream. For simplicity, fault injection typically assumes that all configuration bits have the same susceptibility and that all faults produce a single bit-flip. However, results can be adjusted with measured statistics of SEU susceptibility, if available. Despite these drawbacks, FPGA fault injection is more flexible, makes large fault injection campaigns possible in order to obtain additional data and can provide a reasonable assessment of error mitigation approaches.

For the implementation of fault injection, we used the SEM Controller IP. This module was included in the design in order to carry out continuous configuration memory scrubbing and repair. However, we also took advantage of its fault injection feature to implement our fault injection system. A scheme of the experimental setup is shown in Fig. 2. This setup was used for both neutron irradiation and fault injection. Fault injection was disabled for neutron irradiation.

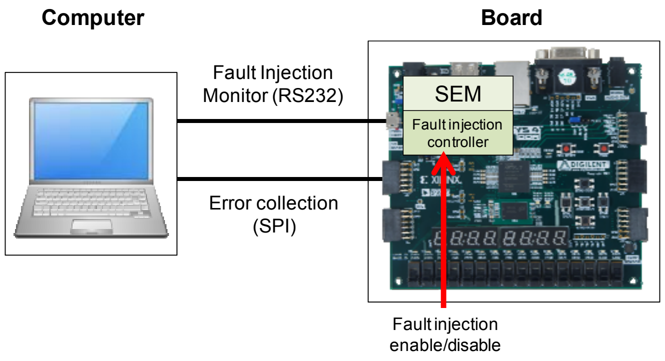
Computer
Board
Fault Injection Monitor (RS232)
SEM
Fault injection controller
Error collection (SPI)
Fault injection enable/disable

Figure 2. Common experimental setup for neutron irradiation and fault injection. Fault injection is disabled for neutron irradiation.

The SEM Controller provides a fault injection interface and a monitor interface. To inject faults, we implemented a small fault injection controller connected to the fault injection interface, which was also embedded in the FPGA. The fault injection controller drives the interface to inject faults in random configuration memory positions at regular time intervals. The selected time interval was 147 ms, which is long enough to allow the SEM to correct an injected error, avoiding error accumulation, and to completely execute the longest benchmark at least three times before injecting a new fault. The maximum estimated error detection and correction latency for the tested FPGA is approximately 26 ms [16]. It is possible that a fault is corrected before causing an error, although it is unlikely. We count it as an error if it is observed before it is corrected. The fault injection controller was hardened by TMR in order to reduce the impact of errors in this module on the obtained measures.

The monitor interface implements an RS-232 protocol compatible, full duplex serial port for exchange of commands and status. Although it can also be used to inject faults, the main use of this interface is to monitor the fault injection process from a host. This interface provides a report of every injected fault, including its CRAM address, the status of the controller and the correction actions that are taken. This report was continuously monitored and logged in the host.

It must be noted that as the SEM Controller is embedded in the tested FPGA, it can also be affected by the injected faults. However, this situation can be detected through the monitor interface. When we observe that the SEM Controller stops injecting faults or shows an abnormal behaviour, the FPGA is fully reconfigured and the fault injection process is resumed.

These errors do not affect the results of the test because they have no effect on the error register collected through the SPI interface.

The three software benchmarks in both the unhardened and hardened software versions were tested. The results of the fault injection campaigns are summarized in Table IV following the same incremental approach used for neutron experiments. For each benchmark, the table shows the number of injected faults, the total number of observed errors and the number of errors that were detected by any of the techniques along with the corresponding percentage and confidence interval. The last column shows the relative improvement with respect to the reference version (first row of Table III), which is computed as the ratio of undetected errors between the reference design and the fault-tolerant design.

Fault injection was run as long as needed in order to achieve at least 1,000 observed errors. The number of injected faults is relatively high due to the low utilization of the FPGA. For completeness, we have included the results of fault injection on a plain design without any hardening and another design only with the HM. In these cases, more faults must be injected into the configuration memory to observe a similar amount of errors because the design is smaller. However, note that these results do not include memory errors, which cannot be injected by the SEM, and therefore they cannot be related to the neutron results shown in the previous section.

TABLE IV. FAULT INJECTION RESULTS

Benchmark		Injected faults	Errors	Detected errors	Rel. Impr.
Plain	Quicksort	126,743	1,021	0 (0%)	-
HM only	Quicksort	95,766	1,002	638 (63.7±3.0%)	-
Unhardened SW	Quicksort	81,099	1,028	778 (75.7±2.6%)	4.1x
	MMULT	122,154	1,279	950 (74.3±2.4%)	3.9x
	AES	83,479	1,363	831 (61.0±2.6%)	2.6x
Hardened SW	Quicksort	77,775	1,004	947 (94.3±1.4%)	17.6x
	MMULT	90,515	1,049	981 (93.5±1.5%)	15.4x
	AES	77,827	1,140	1,098 (96.3±1.1%)	27.1x

For the unhardened software benchmarks, up to 75% of errors were detected by the memory checkers or by the HM. This percentage rises up to 96% when hardened software is used. The amount of detected errors is similar for the quicksort and MMULT benchmarks, but it goes lower in the case of the unhardened AES benchmark. This result can be explained by the fact that the AES benchmark is more data intensive and therefore more prone to data errors which are not detected by the HM. However, most of these data errors can be detected by hardening the software. As a matter of fact, the software-hardened AES is the benchmark that achieves the highest error detection rate.

The contribution of each hardening method is summarized in Table V for the hardened software versions. It must be noted that some errors may be detected by more than one technique. The columns of Table V show, from left to right, the percentage of errors that are detected by the memory checkers, the HM, the hardened software (SW), the memories plus some other technique, and finally by both the HM and the hardened software (HM+SW). The HM has the highest single contribution (57.1% on average) and it is about 1.6 times more effective than the software hardening (35.3% on average). The overlapping of the memory checking with the other techniques is small, but the overlapping between the HM and the hardened software is significant (14% on average).

TABLE V. RELATIVE CONTRIBUTION OF THE HARDENING TECHNIQUES

Benchmark	Mem.	HM	SW	Mem. + HM/SW	HM+SW
Quicksort	27.8%	52.8%	33.4%	7.2%	12.5%
MMULT	25.9%	64.0%	30.8%	5.6%	17.4%
AES	19.1%	54.4%	41.8%	6.9%	12.0%
Average	24.3%	57.1%	35.3%	6.6%	14.0%

Fig. 3 shows a graphical comparison of the cumulative results obtained by neutron irradiation and fault injection for the tested software versions. In this figure, different colors are used to illustrate the different types of detected errors. The lower part (in red) corresponds to errors detected in the memories, the intermediate part (in blue) corresponds to additional errors detected by the hardware monitor and the upper part (in green) corresponds to additional errors detected by software. The latter applies only to the software-hardened versions. The error bars at 95% confidence level for the total percentage of detected errors are also shown.

It can be observed that the fault injection results match the neutron irradiation results quite well. Errors detected in the memories are slightly more relevant in the irradiation experiment, but the total percentage of detected errors obtained by fault injection is within the confidence interval of the neutron irradiation experiment.

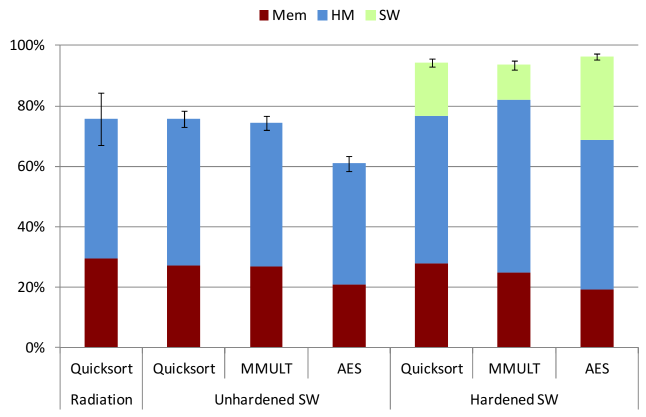

Figure 3. Comparison of error detection rates

The three benchmarks show quite similar error detection rates, with the exception of the unhardened AES benchmark, which was discussed before. The HM shows the highest contribution to the error detection rate, particularly for the hardened software versions. The hardened software has a different and more complex flow, because it includes the software checks, and there are more chances that a fault results in a control-flow error, which can be detected by the HM. Notwithstanding, software hardening is necessary to achieve an error detection rate in the order of 95%. In turn, this high error detection rate is needed to significantly improve the cross-section. An estimation of the relative cross-section improvement based on the fault injection results yields an average factor of 3.5x for the unhardened software versions, which is similar to the result of the neutron irradiation experiment, and 20x for the hardened software versions. Cross-sections can be estimated by dividing the reference cross-section (2.02×10^{-10} cm^2) by the corresponding improvement factor, considering the confidence intervals and assuming that the techniques used in the reference design are applied.

V. Conclusions

In this work we have implemented a hybrid fault-tolerant LEON3 soft-core processor in a low-end FPGA (Artix-7) and evaluated its error detection capabilities. The proposed solution combines the use of SEC/DED codes for memories, a hardware monitor to detect control-flow errors, software-based techniques to detect data errors and configuration memory scrubbing with repair to avoid error accumulation.

The applied hardware techniques do not require a deep knowledge of the processor architecture or the FPGA architecture. As a matter of fact, the hardened memory blocks, the hardware monitor and the configuration memory scrubber can be implemented by soft IP modules that can be easily configured for other processors and FPGAs. Software-based error mitigation techniques are applied in high-level language (C code) and the resulting hardened code is portable. This solution can be fully embedded in a low-end FPGA without the need of external hardware, with reduced overhead and with low intrusiveness. The experimental results demonstrate that the proposed approach can substantially improve fault-tolerance for practical applications.

Acknowledgment

The authors would like to acknowledge the support of Los Alamos Neutron Science Center (LANSCE) to perform the neutron irradiation experiment.

References

[1] S.M. Trimberger, “Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology”, Proceedings of the IEEE, vol. 103, no. 3, pp. 318-331, Mar. 2015.

[2] “Zynq-7000 All Programmable SoC Overview”, Product Specification, Xilinx Inc., DS190, Jan. 2016.

[3] V. Kale, “Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development”, Xilinx Inc., WP469, June 2016.

[4] “Nios II Gen2 Processor Reference Guide”, Altera Corp., NII51001, April 2015.

[5] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, “SEU-induced persistent error propagation in FPGAs,” IEEE Trans. Nucl. Sci., vol. 52, no. 6, pp. 2438–2445, Dec. 2005.

[6] J. R. Azambuja, S. Pagliarini, L. Rosa, and F. L. Kastensmidt, “Exploring the limitations of software-only techniques in SEE detection coverage,” J. Electron. Test., vol. 27, no. 4, pp. 541–550, Aug. 2011.

[7] J. R. Azambuja, A. Lapolli, L. Rosa, and F. L. Kastensmidt, “Detecting SEEs in microprocessors through a non-intrusive hybrid technique,” IEEE Trans. Nucl. Sci., vol. 58, no. 3, pp. 993–1000, June 2011.

[8] J. R. Azambuja, M. Altieri, J. Becker, and F. L. Kastensmidt, “HETA: Hybrid error-detection technique using assertions,” IEEE Trans. Nucl. Sci., vol. 60, no. 4, pp. 2805–2812, Aug. 2013.

[9] J. R. Azambuja, S. Pagliarini, M. Altieri, F.L. Kastensmidt, M. Hubner, J. Becker, G. Foucard, R. Velazco, “A Fault Tolerant Approach to Detect Transient Faults in Microprocessors Based on a Non-Intrusive Reconfigurable Hardware”, IEEE Trans. Nucl. Sci., vol. 59, no. 4, pp. 1117-1124, Aug. 2012.

[10] P. Bernardi, L. Sterpone, M. Violante, M. Portela-Garcia, "Hybrid Fault Detection Technique: A Case Study on Virtex-II Pro's PowerPC 405", IEEE Trans. Nucl. Sci., vol. 53, no. 6, pp. 3550-3557, Dec. 2006.

[11] 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”, Proc. 16th IEEE On-Line Testing Symp., pp. 167-172, July 2010.

[12] 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”. Microprocessors and Microsystems, vol. 36, no. 5, pp. 334-343, July 2012.

[13] L. Parra, A. Lindoso, M. Portela, L. Entrena, F. Restrepo-Calle, S. Cuenca-Asensi, A. Martinez-Alvarez, “Efficient Mitigation of Data and Control Flow Errors in Microprocessors”. IEEE Trans. Nucl. Sci., vol. 61, no. 4, pp. 1590-1596, Aug. 2014.

[14] 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., vol. 61, no. 6, pp. 3236-3243, Dec. 2014.

[15] B. Du, M. Sonza Reorda, L. Sterpone, L. Parra, M. Portela-García, A. Lindoso, L. Entrena, “On-line Test of Control Flow Errors: A new Debug Interface-based approach”, IEEE Trans. on Computers, vol. 65, no. 6, pp. 1846-1855, June 2016.

[16] “Soft Error Mitigation Controller v4.1”, Product Guide, Xilinx Inc., PG036, Nov. 2014.

[17] F. Kastensmidt, P. Rech, “FPGAs and Parallel Architectures for Aerospace Applications. Soft Errors and Fault-Tolerant Design”, Springer, 2016.

[18] M. Nicolaidis, “Soft errors in modern electronic systems”, Springer, 2011.

[19] N. A. Harward, M. R. Gardiner, L. W. Hsiao, M. J. Wirthlin, “Estimating Soft Processor Soft Error Sensitivity Through Fault Injection”. Proc. IEEE 23rd Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), pp. 143-150, May 2015.

[20] R. Vemu, S. Gurumurthy, and J. A. Abraham, “ACCE: Automatic correction of control-flow errors”, Proc. Int. Test Conf., paper no. 27.2, Oct. 2007.

[21] Z. Alkhalifa, V. S.S. Nair, N. Krishnamurthy, J.A. Abraham, “Design and evaluation of System-Level Checks for On-line Control Flow Error Detection”, IEEE Trans. on Parallel and Distributed Systems, vol. 10, no. 6, pp. 627–641, June 1999.

[22] O. Goloubeva, M. Rebaudengo, M.S.Reorda,and M.Violante, “Software-Implemented Hardware Fault Tolerance”, Springer, 2006.

[23] M. Hiller. “Executable assertions for detecting data errors in embedded control systems”, Proc. IEEE Int. Conf. on Dependable Systems and Networks, pp. 24-33, June 2000.

[24] B. Nicolescu, R. Velazco, “Detecting soft errors by a purely software approach: method, tools and experimental results”. Design, Automation and Test in Europe Conf., pp. 57-62, 2003.

[25] “Xilinx TMRTool”, Product Brief, Xilinx Inc., 2009.

[26] A. Sanchez-Clemente, L. Entrena and M. Garcia-Valderas, "Partial TMR in FPGAs Using Approximate Logic Circuits", IEEE Trans. Nucl. Sci., vol.63, no. 4, pp. 2233-2240, Aug. 2016.

[27] B. Pratt, M. Caffrey, P. Graham, E. Johnson, K. Morgan, and M. Wirthlin. “Improving FPGA design robustness with partial TMR”. Proc.

IEEE Int. Rel. Phy. Symp. (IRPS), pp. 27-30, Mar. 2006.

[28] G. A. Reis, J. Chang, N. Vachharajani, S. S. Mukherjee, R. Rangan and D. I. August. “Design and evaluation of hybrid fault-detection systems”. Proc. 32nd Int. Symp. on Comp. Arch., pp. 148-159, June 2005.

[29] T. Michel, R. Leveugle, G. Saucier. “A New Approach to Control Flow Checking Without Program Modification”, 21th Int. Symp. on Fault-Tolerant Computing (FTCS-21), pp. 334-341, June 1991.

[30] Y. Li, B. Nelson, and M. Wirthlin. “Reliability Models for SEC/DED Memory With Scrubbing in FPGA-Based Designs”. IEEE Trans. Nucl. Sci., vol. 60, no. 4, pp. 2720-2727, Dec. 2013.

[31] M. Rebaudengo, M. S. Reorda, M. Torchiano, M. Violante. “Soft-error detection through software fault-tolerance techniques”, Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 210- 218, Nov. 1999.

[32] M. Berg et al. “Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis”, IEEE Trans. Nucl. Sci., vol. 55, no. 4, pp. 2259-2266, June 2008.

[33] H. Quinn et al. “Using Benchmarks for Radiation Testing of Microprocessors and FPGAs”, IEEE Trans. Nucl. Sci., vol. 62, no. 6, pp. 2547-2554, Dec. 2015.

[34] H. Quinn, M. Wirthlin. “Validation Techniques for Fault Emulation of SRAM-based FPGAs”, IEEE Trans. Nucl. Sci., vol. 62, no. 4, pp. 1487-1500, Aug. 2015.

[35] “GRLIB IP Core User’s Manual”, Version 1.1.0 - B4113, Aeroflex Gaisler, Jan. 2012

[36] T. Heijmen, “Soft errors from space to ground: Historical overview, empirical evidence, and future trends”, In *Soft Errors in Modern Electronic Systems*, pp. 1-25, Springer, 2011.

[37] L. Sterpone, “A Novel Design Flow for Fault Tolerance SRAM-Based FPGA Systems”, in *Electronics System Design Techniques for Safety Critical Applications*, pp.85-99, Springer, 2009.

[38] “Device Reliability Report”, Xilinx Inc., UG116 (v10.3.1), Sept. 2015.