

# Effectiveness of Software-Based Hardening for Radiation-Induced Soft Errors in Real-Time Operating Systems

Thiago Santini<sup>1</sup>, Christoph Borchert<sup>2</sup>, Christian Dietrich<sup>3</sup>, Horst Schirmeier<sup>2</sup>,  
Martin Hoffmann<sup>3</sup>, Olaf Spinczyk<sup>2</sup>, Daniel Lohmann<sup>3</sup>, Flávio Rech Wagner<sup>4</sup>,  
and Paolo Rech<sup>4</sup>

<sup>1</sup> University of Tübingen, Tübingen, Germany

<sup>2</sup> Technische Universität Dortmund, Dortmund, Germany

<sup>3</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen-Nürnberg, Germany

<sup>4</sup> Federal University of Rio Grande do Sul, Porto Alegre, Brazil

**Abstract.** For decades, radiation-induced failures have been a known issue for aero-space systems, in which redundancy mechanisms are employed as a protection method. Due to the shrinking of structures and operating voltages, these failures are increasingly becoming an issue even for terrestrial applications. Unfortunately, redundancy increases costs, area usage, and power consumption, which can hinder its utilization in cost- and power-sensitive safety-critical applications, such as automotive. To overcome this limitation, multiple software-based approaches have been proposed, which assume the existence of an underlying error-free operating system. In this paper, we investigate the radiation reliability of two dependability-oriented real-time operating systems, namely, the popular eCos operating system hardened through aspect-oriented programming methods, and *dOSEK*, an embedded kernel designed from the ground up having reliability as a major concern. Both operating systems were evaluated through extensive neutron-beam testings on a 28nm ARM-based state-of-the-art system-on-chip, and their fault tolerance mechanisms reached reductions in the overall cross-sections relative to their baselines up to 91 percent and 74 percent, respectively.

## 1 Introduction

Commercial-Off-The-Shelf (COTS) systems have become a valid alternative to specific radiation-hardened devices in safety-critical applications, like biomedical implantable devices, automotive control systems, and aircraft or satellite stabilizer and control circuitry. For instance, the spacecraft onboard computer in NASA's PhoneSat nano-satellite is built around COTS smartphones running the Android operating system [9]. The main reason for preferring a COTS device is that hardened devices are typically very expensive, as they require unique circuit design and lithography to meet the reliability requirements, and the produced volumes are very low. On the contrary, COTS components are low cost, flexible, and provide fast time-to-market as well as low power consumption.

Nonetheless, when reliability is a major concern, the use of general-purpose devices must be carefully evaluated. As technology scales down, CMOS devices are becoming more susceptible to soft errors induced by ionizing particles; in fact, nowadays radiation-induced failures are a concern not only in radiation-harsh environments, such as the space, but also in milder environments, such as at sea level. High-energy neutrons generated by the interaction of cosmic rays with the terrestrial atmosphere may in fact have enough energy to corrupt data stored in SRAM memories or to affect logic computations [2]. This is especially relevant in cost-sensitive domains, such as the automotive sector. Here, efficiency in terms of per-unit-prices is a key criterion, so full hardware redundancy can be prohibitively expensive. One of the proposed approaches to circumvent these limitations in a cost-effective and flexible way is through software-implemented fault tolerance, such as software-based redundant multi-threading [28] and process-level redundancy [25]. These approaches assume a fault-free underlying operating system. However, an operating system (OS) must keep several data structures containing critical data and pointers, such as device and file descriptors, memory information, and process list, which are very likely to lead to a device functional interruption if corrupted [8], thus making OSs particularly sensitive.

In this context, two approaches have been recently proposed in order to establish a reliable underlying operating system for real-time embedded computing: 1) a version of the popular eCos operating system hardened through aspect-oriented programming methods [4], and 2) *dOSEK*, an embedded kernel designed from the ground up with reliability as first-class design goal [12]. These approaches have been evaluated through ISA-level fault injection with the *FAIL\** [23] framework based on an IA-32 platform emulator and assuming a single-bit fault model over the entire fault space of the architectural view from the software’s perspective (i.e., in the main memory as well as instruction pointer, general-purpose, stack, and flags registers). In this work, we expand on these evaluations through extensive neutron-beam testing on a 28 nm ARM-based state-of-the-art system-on-chip. Our main contributions are: **1)** Cross-section data to help the device characterization. These data complement the information provided by sources that investigate the selected device’s radiation sensitivity, such as its bit [19,15], cache memories [20], and general purpose operating systems [22] cross-sections. **2)** A realistic evaluation of the radiation-reliability of the proposed OS mitigation approaches. Our experimental evaluation uses the ARM architecture, which is very common on the actual targets in the embedded domain. We provide expected Failure In Time (FIT) values in Section 4.3.

## 2 Background

### 2.1 eCos and Software-Implemented Fault Tolerance

For this study, we chose the off-the-shelf operating system *eCos* [16] as a typical representative for embedded real-time operating systems. eCos (embedded Configurable operating system), as the name suggests, offers configurability at

compile time of various system components, such as file systems and networking, resulting in roughly one million lines of C/C++ code. To apply software-implemented fault-tolerance to such an enormous code base, we chose two *generic* error-detection and error-correction mechanisms with transparent compiler support – a manual implementation in C/C++ would be infeasible. **1 Generic Object Protection (GOP):** The principle of GOP [5,3] is to introduce redundancy into the program data structures to implement an error-correcting code. In this study, we use a Hamming code [10], since it can be efficiently implemented in software by bit-slicing [24]. The implementation processes 32 bits in parallel, which allows for correction of multi-bit errors, in particular, all burst errors up to 32 bits. At program run time, the Hamming code gets verified *before* an instance of a data structure (C++ *object*) is used. Then, *after* object usage and potential data modification, the Hamming code gets updated. The object-oriented software structure of the eCos kernel restricts data access to member functions of a data structure. Thus, it suffices to carry out checks *before* member-function calls and to update the Hamming code *after* the member function has returned. GOP is implemented by means of Aspect-Oriented Programming [14], which allows for a completely modular implementation separated from the eCos source code. The *AspectC++* [27] compiler automatically inserts the protection rules at compile time. **2 Stack Checksum:** The second fault-tolerance mechanism applied to eCos is a 32-bit checksum for stack memory. When the eCos kernel preempts a thread of control, or when a thread blocks while waiting for a semaphore, a checksum covering the thread’s occupied stack memory is attached to the thread. When the thread is eventually resumed, the checksum gets verified. Thus, errors corrupting the stack memory while a thread is inactive are detected. Please note that extending this mechanism to error correction is straightforward by implementing a Hamming code similar to the GOP. Finally, the *Stack Checksum* mechanism is also implemented as a generic module in *AspectC++*, which instruments the eCos-kernel source code with minimal effort from the programmer.

## 2.2 dOSEK – A Soft-Error Resilient OS

As our second system, we chose *dOSEK* [12], a framework for generating dependable real-time kernels. The first-class design objective during the system development was resilience against soft-errors. In previous (exhaustive) fault-injection campaigns, the usage of *dOSEK* reduced the rate of undetected failures by multiple orders of magnitude.

*dOSEK* adheres to the OSEK-OS [18] specification, a standardized kernel Application Programming Interface (API) developed by the automotive industry. OSEK systems are specified declaratively: the number and configuration of threads, alarms, interrupts, resources, and events is known at compile time. *dOSEK*, following the tradition of OSEK system generators, exploits this static application knowledge to foster dependability. Furthermore, two basic design principles were also applied: removal of unnecessary indirections and integration of active dependability measures.

Like eCos, *dOSEK* provides static configurability at compile time. We used three configuration sets of dependability measures in our test setup. **1 Baseline:** All system objects are allocated statically; pointer indirection is avoided wherever possible; the kernel is activated through a supervisor call, but executed only with user privileges; inside the kernel, function calls are avoided by massive inlining. **2 Encoded:** On top of the baseline, specialized data protection is applied: checksums for thread contexts, parity bits for saved stack pointers, and dual-modular redundancy (DMR) for counters. For the scheduler, ANB encoding was applied, an active measure that protects data flow as well as the control flow. **3 Asserts:** On top of the baseline, application-specific protection mechanisms were added. By system-wide static analysis, knowledge about the dynamic behavior of the application-kernel interaction was extracted and run-time assertions to check for compliance were injected.<sup>5</sup>

### 3 Experimental Methodology

In this work, we have opted to perform an evaluation of the proposed approaches through accelerated radiation testing. Radiation testing does not restrain faults to a single part of the chip, whereas fault injection can be performed only on a selection of user-accessible resources for those devices, like COTS, for which an Register Transfer Level (RTL) description is not usually available. Moreover, although simulators and emulators allow a more controlled fault injection, they are always an oversimplification of the physical reality and, thus, cannot replace radiation tests for Radiation Hardness Assurance (RHA) testing [11].

#### 3.1 Device Under Test

The Device Under Test (DUT) is the Xilinx Zynq<sup>TM</sup>-7000 AP System-on-Chip (SoC) implemented in a 28 nm CMOS technology. The DUT disposes of two ARM<sup>®</sup>Cortex<sup>TM</sup>-A9 cores with a maximum frequency of 667 MHz. Each core has 32 KiB Level 1 4-way set-associative instruction and data caches, and they share a 512 KiB 8-way set-associative Level 2 cache [7]. During tests, only a single core (*CPU0*) was used. Parity checking was disabled for both cache levels to allow the assessment of the investigated approaches in the absence of hardware-based protection mechanisms. It is paramount to note that only the SoC chip was irradiated (i.e., the external DRAM chips were *not* irradiated). Both OSs were tested under heavy load, and the amount of threads and resources employed was selected as to fill up the cache memories in order to maximize attack surface.

---

<sup>5</sup> The Baseline and the Encoded variants are based on and discussed in more detail in [12], whereas the static application analysis and the system-state assertions are based on and detailed in [6].

### 3.2 eCos Configuration and Benchmarks

We used a port of eCos 3.0 for the aforementioned Zynq<sup>6</sup> hardware platform and selected a minimal configuration of eCos without unneeded device drivers. In addition, we ignored spurious device interrupts. To reduce corruption of program instructions, we disabled the instruction caching at the L2 cache (only allowing L2 data caching), and the L1 instruction cache was regularly invalidated before interrupt processing. For the evaluation of the OS under heavy load, we selected two benchmarks, both supporting a parameterizable number of threads, selected as to fill up the caches, from the kernel-test suite bundled with eCos itself: **BIN\_SEM2 (BS)** implements a classical synchronization problem known as *Dining Philosophers*. We configured 400 threads (*philosophers*) that use 400 *forks* (i.e., `Cyg_Binary_Semaphore` objects) for mutual exclusion (*eating* with two forks). Once a philosopher acquires both neighboring forks, it checks by an assertion that neighboring philosophers are not in the *eating* state. After a pseudo-random delay, the philosopher releases both forks and tries again for 25,000 iterations. **TIMESLICE2 (TS)** verifies that the per-thread time-slice distribution works under preemption. We configured 800 low-priority threads that continuously increment a per-thread counter, and a single high-priority thread is scheduled at regular intervals to preempt the other threads. The benchmark finishes after a predetermined period of time, such that each low-priority thread should have received two time slices. Finally, an assertion tests whether all threads have run.

These benchmarks were evaluated with two eCos variants: a baseline variant with no protection mechanisms, and a variant hardened through the methods described in section 2.1. **BIN\_SEM2** has a baseline run time of 1.98 seconds, whereas the hardened variant has a run time of 2.08 seconds (an overhead of 4.745%). **TIMESLICE2** has a baseline run time of 1.6 seconds, whereas the hardened variant has a run time of 1.65 seconds (an overhead of 2.932%).

### 3.3 dOSEK Configuration and Benchmark

We ported the *dOSEK* system generator to the ARM platform used on the Zynq hardware while preserving *dOSEK*'s basic design principles. To ease the comparison with the eCos benchmark, we did not use the MMU to provide spatial isolation between the OSEK threads. Privilege isolation was used to execute kernel and application in user mode; only kernel entry and thread dispatching were executed with supervisor privileges.

As benchmark, we generated an application compliant with the OSEK BCC1 conformance class, consisting of 250 threads organized in 125 pairs. The test case was designed to particularly fill up the cache, which is hit by the neutron beam, with OS state. Each thread pair has a lower-priority non-preemptable thread (*L-thread*) and a high-priority thread (*H-thread*). We configured 250 alarms connected to 250 OSEK counter objects; 125 counters are driven by a hardware timer and activate the L-thread. The other 125 counters are incremented by

<sup>6</sup> <https://github.com/antmicro/ecos-mars-zx3/>

the L-threads and activate the associated H-thread on alarm expiration. The periods and phases for the alarms were shuffled once by a pseudo-random number generator. Besides the pair coupling, we also added (pseudo-randomly) cross-dependencies between pairs: a L-thread activates the H-thread of another pair; a H-thread chains its execution to another pair’s L-thread; a L-thread waits actively for another H-thread to set a global variable. In total, 42 such cross dependencies were introduced.

During execution, each thread queries its associated alarm value, applies some calculation to it, and hashes the result and its thread *ID* onto a global CRC32 checksum. The hash update operation is protected by an OSEK non-preemptable critical section. After 1500 hash updates, the application asserts that the resulting hash equals to a golden value calculated at compilation time. Both, checksum storage and hash update counter are protected with triple-modular redundancy.

The exactly same application was evaluated with the three variants of *d*OSEK described in section 2.2, namely, **Baseline**, **Encoded**, and **Asserts**. All variants exhibited a similar run time ( $\approx 3.42$  s). Since the kernel run time is orders of magnitude smaller than the application run and idle time, the incurred run-time penalties of the additional protection measures can be considered negligible.

### 3.4 Experimental Setup

Radiation experiments were performed at Los Alamos National Laboratory (LANL) in the Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House II, called ICE House II. The ICE House II beam line provides a white neutron source that emulates the energy spectrum of the atmospheric neutron flux. The available neutron flux was approximately  $1 \times 10^6$  n/(cm<sup>2</sup>s) for energies above 10 MeV. The beam was focused on a spot with a diameter of two inches, which provided uniform irradiation of the SoC, without directly affecting nearby board power control circuitry and DRAM chips. It is worth noting that, even if the flux of neutrons at ICE House II is several orders of magnitude higher than the natural one at sea level (which is estimated to be about 13 n/(cm<sup>2</sup>h) [13]), the test was tuned to make negligible the probability of having more than one neutron generating a failure in one single code execution (estimated through the method described in [21] to be no higher than  $1.38 \times 10^{-5}$  errors/execution). This allows the scaling of experimental data in the natural radioactive environment without introducing artificial behaviors.

To reduce the uncertainty of the experimental results, four DUTs were irradiated in parallel. The four boards with the same hardware revision were aligned with the beam, placed at 62, 64, 66.5, and 68.5 inches from the source, respectively. A flux de-rating factor was calculated for each board to take beam degradation due to the distance from the source into account. To minimize the statistical error and to avoid experimental results biased on the selected board and distance de-rating factor, the benchmarks were executed alternatively in all four devices. In total, the boards received a fluence of  $9.87 \times 10^{11}$  n/cm<sup>2</sup>, thus receiving the radiation equivalent to  $8.67 \times 10^6$  years of exposure in the natural environment at sea level. It is worth noticing that hardened variants received more beam time

than baseline ones. Since these systems are intrinsically less prone to errors, they require longer testing times to achieve a statically significant amount of observed errors.

A test manager application was responsible for collecting and time-stamping incoming logs from the boards through UART connections. The test manager application also served as a watchdog, responsible for detecting otherwise irrecoverable failure situations and rebooting the boards through an Ethernet controlled switch. Whenever such situations happened, they were time-stamped and logged. Irrecoverable situations are considered when the board exceeds a time-out much larger than the application execution time without sending successful execution logs.

## 4 Experimental Results

We report our results as cross-sections. The cross-section  $\sigma$  is the most widely used metric to evaluate a device radiation sensitivity and is evaluated by dividing the amount of observed errors by the particle fluence ( $n/cm^2$ ) received by the device. By definition, the cross-section, expressed in  $cm^2$ , is the device sensitive area – that is, the area that generates a failure if hit by an impinging particle [2]. Values are shown with relative intervals to account for the failure rate estimation error (95% CI) and neutron count uncertainty.

The outcome of each application run was classified as *benign* or *malign*. Benign executions are those in which the expected output was produced, or an error was detected before it could lead to a Silent Data Corruption (SDC) or Functional Interruption (FI). Malign executions are those in which a SDC was produced (e.g., one of the assertions described in sections 3.2 and 3.3 failed, garbage was found in the output) or a FI occurred (e.g., the board rebooted by itself, no correct output was produced before the test manager watchdog ran out). Each malign execution was accounted as a single error when calculating cross-sections and only if the preceding execution was benign. For the remainder of this paper, we will use the term *very rare* to refer to events that had less than three occurrences observed per benchmark; we consider their probability to be negligible and, since we cannot draw any additional statistically significant conclusion about these events, refrain from further discussing them. Events with zero occurrences are explicitly shown through a cross-section of  $0$ .

### 4.1 eCos

As shown in Fig. 1a, the hardening resulted in a reduction in the overall cross-section by a factor of at least 55% (upper  $TS_{Hardened}$  relative to the lower  $TS_{Baseline}$ ) up to 91% (lower  $BS_{Hardened}$  relative to the upper  $BS_{Baseline}$ ).

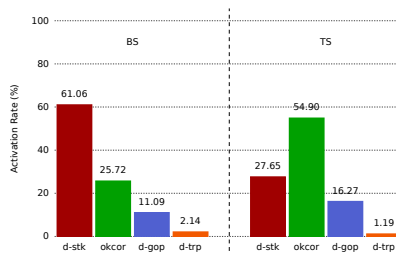
Table 1 details the possible outcomes for each benchmark run, and the overall cross-section is broken down into its contributors in Fig. 1b. The occurrences of *rst* and *scorr* were *very rare*. From the remaining (and major) cross-section contributors, it is clear that in all cases *tout* occurrences are fairly more probable

than *fail* ones. In other words, a system hang (the system stops producing any output) was more common than an SDC. These *hangs* likely originate from illegal memory accesses and jumps; invalid data accesses can leave the system in a corrupt state, and deviant instruction accesses (e.g., stemming from corrupted return addresses in the stack) can lead to the execution of arbitrary code, both likely to stop the system from producing an output in a timely manner. Moreover, both the eCos kernel and the application run in supervisor mode [16], which can exacerbate this effect since invalid accesses from the application do not cause the OS to terminate the application.

The hardening had similar effects in both applications: *fail* became a *very rare* event, whereas *tout* occurrences were significantly reduced. Unfortunately, it is not possible to establish one-to-one relationships between the employed fault-tolerance mechanisms and the malign events reduction due to the Architectural Vulnerability Factor (AVF) [17]; in other words, there are errors that are corrected by the employed mechanisms that would not influence the system in an observable way. In fact, the cross-section for correction/detection events ( $\approx 1.2 \times 10^{-8}$  for both hardened benchmarks) is much larger than those of malign events for the baseline versions. Nonetheless, we break down the relative activations for these mechanisms in Fig. 4.1. This figure suggests that stack data are the largest attack surface for the *BS* benchmark, in contrast to the *TS* benchmark, in which eCos class members data seem to present the largest attack surface. Furthermore, it is worth noting that the *d-trp* cross-section for both benchmarks ( $BS_{Baseline} = 9.45 \times 10^{-10}$  and  $TS_{Baseline} = 1.01 \times 10^{-9}$ ) were diminished with the employment of the hardening mechanisms ( $BS_{Hardened} = 2.66 \times 10^{-10}$  and  $TS_{Hardened} = 1.42 \times 10^{-10}$ ), showing a replacement of generic hardware traps by more specific detection mechanisms, which could be more easily corrected if possible and desired.

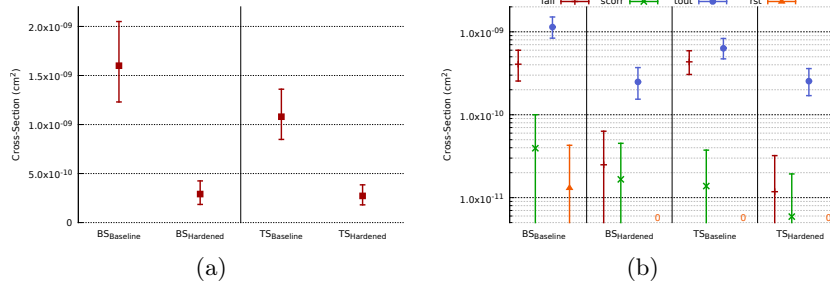
**Table 1.** Possible outcomes for the eCos benchmarks

	Baseline	Hardened	Description
ok	✓	✓	Successful run
okcor	-	✓	GOP corrected
d-gop	-	Detect	GOP (uncorrectable)
d-stk	-	Detect	Wrong stack checksum
d-trp	Detect	Detect	Hardware trap
fail	SDC	SDC	Application assertion failed
scorr	SDC	SDC	Serial corrupted
rst	FI	FI	Board rebooted
tout	FI	FI	Timeout without output



**Fig. 2.** Relative activations for the detection/correction mechanisms for the hardened benchmarks versions.





**Fig. 1.** Overall cross-sections for the BIN\_SEM2 (BS) and TIMESLICE2 (TS) benchmarks (a) as well as their comprehensive cross-section list (b); note the *y-semilog* scale on (b).

## 4.2 dOSEK

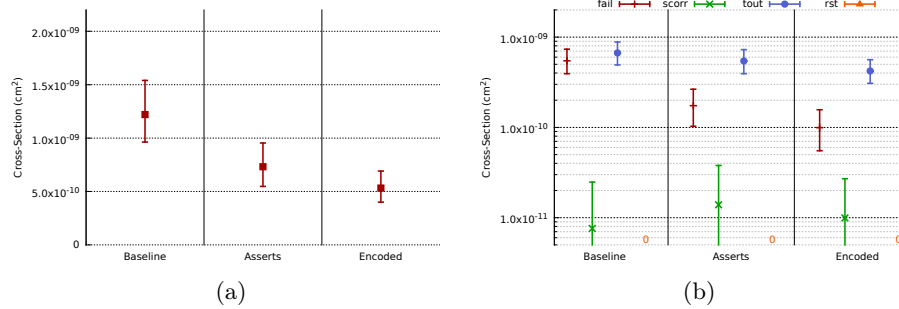
In Fig. 3a, the overall cross-section of the observed errors is shown for all three variants. The application-specific assertions reduce the cross-section by at least 0.93 % (upper Asserts relative to lower Baseline) up to 64 % (lower Asserts relative to upper Baseline); the kernel encoding by at least 28 % up to 74 %.

Each application run was classified into one of the categories from Table 2. *Fail*, *scorr*, *rst*, and *tout* are counted as errors and contribute to the overall cross-section, which is broken down in Fig. 3b. The results for *dOSEK* are similar to *eCos*: Occurrences of *scorr* were *very rare*, and *rst* events did not occur. A hanging system was more likely than a failing one, whereas Asserts and Encoded significantly reduced these *tout* events. The actual *fail* cross-section was reduced at least by 33 % (Asserts) up to 92 % (Encoded).

It is noteworthy that in both variants the detection was mainly driven by a single measure: the detection for the Assert variant ( $\sigma = 8.57 \times 10^{-10}$ ) is dominated by the introduced assertions (76 %). For the Encoded variant, detection ( $\sigma = 1.32 \times 10^{-9}$ ) stems mainly from the ANB-encoded scheduler (68 %). Both observations are in accordance with the simulated fault-injection experiments.

## 4.3 FIT Figures

As mentioned in Subsection 3.4, due to the characteristics of our neutron source and failure rate, it is possible to scale our experimental results to Earth's natural environment. Table 3 reports the worst-case FIT figures at sea level given the measured cross-sections, expressed as errors per billion hours of device operation. These values represent a reference for evaluating if the tested device meets the reliability requirement of a project based on the environment of operation and



**Fig. 3.** Overall cross-sections for the three *dOSEK* variants (a) as well as their comprehensive cross-section list (b); note the *y-semilog* scale on (b).

the relevant functional safety standard (e.g., ISO 26262 [1]).

**Table 2.** Possible outcomes for the *dOSEK* variants.

	Baseline	Encoded	Asserts	Description
ok	✓	✓	✓	Successful run
d-xor	-	Detect	-	Thread context checksum
d-dmr	-	Detect	-	Counters DMR
d-anb	-	Detect	-	Scheduler ANB encoding
d-par	-	Detect	-	Saved stack pointer parity
d-sta	-	-	Detect	<i>dOSEK</i> assertion failed
d-log	Detect	Detect	Detect	Impossible control flow
d-trp	Detect	Detect	Detect	Hardware trap
d-unk	Detect	Detect	Detect	Spurious fault detection hook
fail	SDC	SDC	SDC	Application assertion failed
scorr	SDC	SDC	SDC	Serial corrupted
rst	FI	FI	FI	Board rebooted
tout	FI	FI	FI	Timeout without output

**Table 3.** FIT at sea level for energies higher than 10 MeV (Flux  $\approx 13$  n/(cm<sup>2</sup>h) [13]).

OS	Variant	FIT
eCos	Baseline / BIN_SEM2	26.65
	Hardened / BIN_SEM2	5.53
	Baseline / TIMESLICE2	17.68
	Hardened / TIMESLICE2	5.01
<i>dOSEK</i>	Baseline	20.02
	Asserts	12.40
	Encoded	8.98

## 5 Final Remarks

In this paper, we evaluated the radiation reliability of two dependability-oriented real-time operating systems and the efficacy of their fault-tolerance mechanisms. Both investigated approaches (eCos and *dOSEK*) exhibited a significant reduction in the overall cross-section (up to 91 percent and 74 percent relative to the baseline variants, respectively), attesting for the capabilities of the investigated fault-tolerance mechanisms for usage at an environment with similar neutron flux to

the terrestrial one. In fact, the baseline versions would limit the Safety Integrity Level (SIL) of the Equipment Under Control (EUC) in *continuous operation* mode at sea level to IEC61508 SIL 3 – i.e., within  $(10^{-7}, 10^{-8}]$  failures per hour [26]. In contrast, the *hardened* eCos variant and *dOSEK Encoded* variant would mitigate enough faults as to allow the EUC to attain SIL 4 (i.e., within  $(10^{-8}, 10^{-9}]$  failures per hour), the highest SIL<sup>7</sup>. It is worth noticing that we cannot directly compare the results for eCos to those of *dOSEK* because the evaluation is highly dependent on the application. In retrospect, it would have been more advantageous to have used exactly the same application to evaluate both operating systems. Nonetheless, the evaluated applications are conceptually similar (in the sense that they stress-test the kernel scheduling, preemption, and timer functionalities), and the investigated approaches exhibited failure rates in the same order of magnitude. Furthermore, due to massive function inlining to avoid run time indirections, the code size of *dOSEK* is two orders of magnitude higher than that of eCos, and it is worth noticing that the protection mechanisms applied to harden eCos are generic and can be applied to other object-oriented C++ programs easily.

As future work, we plan to extend the FAIL\* framework to evaluate the systems here evaluated through fault injection campaigns. The intention of this future work is threefold: 1) to corroborate FAIL\* and the accelerated radiation tests, 2) to better comprehend the way in which these OSs fail and help developing further fault tolerance mechanisms, and 3) to provide an open framework to evaluate the reliability of ARM-based processors.

## References

1. ISO/DIS 26262. Tech. rep. (2011)
2. Baumann, R.: Soft errors in advanced computer systems. *IEEE Design & Test of Computers* 22(3) (2005)
3. Borchert, C., Spinczyk, O.: Hardening an L4 microkernel against soft errors by aspect-oriented programming and whole-program analysis. In: *Proc. of the 8th Workshop on Programming Languages and Operating Systems*. ACM (2015)
4. Borchert, C., et al.: Generative software-based memory error detection and correction for operating system data structures. In: *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP Int. Conf. on*. pp. 1–12. IEEE (2013)
5. Borchert, C., et al.: Generic soft-error detection and correction for concurrent data structures. *IEEE Trans. on Dependable and Secure Computing* PP(99) (2015)
6. Dietrich, C., et al.: Cross-kernel control-flow-graph analysis for event-driven real-time systems. In: *Proc. of the Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES '15)*. ACM (Jun 2015)
7. Diligent: Zedboard data sheet overview (2014), [http://www.xilinx.com/support/documentation/data\\_sheets/ds190-Zynq-7000-Overview.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf)

---

<sup>7</sup> It is important to notice that this is *based solely on the estimated failure rate figures and assuming all failures could lead to dangerous consequences*; no hazard and risk assessment was carried out, nor was the software tested for coverage; we do not claim the EUC to achieve these SILs.

8. Gu, W., et al.: Characterization of Linux kernel behavior under errors. In: Int. Conf. on Dependable Systems and Networks (DSN). IEEE (2003)
9. Guillen Salas, A., et al.: PhoneSat in-flight experience results. In: Proc. of the Small Satellites and Services Symp. (May 2014)
10. Hamming, R.W.: Error detecting and error correcting codes. Bell System Technical Journal 29(2), 147–160 (1950)
11. Herrera-Alzu, I.e.a.: System design framework and methodology for Xilinx Virtex FPGA configuration scrubbers. IEEE Trans. on Nucl. Sci. 61(1), 619–629 (2014)
12. Hoffmann, M., et al.: dOSEK: The design and implementation of a dependability-oriented static embedded kernel. In: Proc. of the 21st Real-Time and Embedded Technology and Applications (RTAS '15). pp. 259–270. IEEE (Apr 2015)
13. JEDEC Solid State Technology Association: JESD89-3A: Test Method for Beam Accelerated Soft Error Rate (Nov 2007), <http://www.jedec.org/standards-documents/docs/jesd-89-3a>
14. Kiczales, G., et al.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) 11th European Conf. on Object-Oriented Programming (ECOOP '97). LNCS, vol. 1241, pp. 220–242. Springer (Jun 1997)
15. Lesea, A., et al.: Soft error study of ARM SoC at 28 nanometers. Proc. of the IEEE Workshop on Silicon Errors in Logic - System Effects, 2014 (2014)
16. Massa, A.: Embedded Software Development with eCos. Prentice Hall Professional Technical Reference (2002)
17. Mukherjee, S.S., et al.: A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: Proc. of the 36th annual IEEE/ACM Int. Symp. on Microarchitecture. IEEE (2003)
18. OSEK/VDX Group: Operating system specification 2.2.3. Tech. rep. (Feb 2005), <http://portal.osek-vdx.org/files/pdf/specs/os223.pdf>, visited 2014-09-29
19. Quinn, H., et al.: Single-event effects in low-cost, low-power microprocessors. In: Radiation Effects Data Workshop (REDW), 2014 IEEE. pp. 1–9 (July 2014)
20. Santini, T., et al.: Reducing embedded software radiation-induced failures through cache memories. In: Test Symp. (ETS), 2014 19th European. pp. 1–6. IEEE (2014)
21. Santini, T., et al.: Beyond cross-section: Spatio-temporal reliability analysis. ACM Trans. Embed. Comput. Syst. 15(1), 3:1–3:16 (Dec 2015)
22. Santini, T., et al.: Exploiting cache conflicts to reduce radiation sensitivity of operating systems on embedded systems. In: Proc. of the Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems. pp. 49–58. CASES, IEEE (2015)
23. Schirmeier, H., et al.: FAIL\*: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance. In: Proc. of the 11th European Dependable Computing Conf. pp. 245–255. IEEE (Sep 2015)
24. Shirvani, P.P., et al.: Software-implemented EDAC protection against SEUs. IEEE Trans. on Reliability 49(3), 273–284 (Sep 2000)
25. Shye, A., et al.: PLR: A software approach to transient fault tolerance for multicore architectures. IEEE Trans. on Dependable and Secure Computing (2009)
26. Smith, D.J., Simpson, K.G.: SAFETY CRITICAL SYSTEMS HANDBOOK: a straightforward guide to functional safety, IEC 61508 and related standards, including process IEC 61511 and machinery IEC 62061 and ISO 13849. Elsevier (2010)
27. Spinczyk, O., Lohmann, D.: The design and implementation of AspectC++. Knowledge-Based Systems, Special Issue on Techniques to Produce Intelligent Secure Software 20(7), 636–651 (2007)
28. Wang, C., et al.: Compiler-managed software-based redundant multi-threading for transient fault detection. In: Proc. of the Int. Symp. on Code Generation and Optimization. pp. 244–258. CGO '07, IEEE (2007)