A Hardware-Software Approach for On-line Soft Error Mitigation in Interrupt-Driven Applications

Antonio Martínez-Álvarez, Felipe Restrepo-Calle, Sergio Cuenca-Asensi, Leonardo M. Reyneri, Almudena Lindoso, Luis Entrena

Abstract— Integrity assurance of configuration data has a significant impact on microcontroller-based systems reliability. This is especially true when running applications driven by events which behavior is tightly coupled to this kind of data. This work proposes a new hybrid technique that combines hardware and software resources for detecting and recovering soft-errors in system configuration data. Our approach is based on the utilization of a common built-in microcontroller resource (timer) that works jointly with a software-based technique, which is responsible to periodically refresh the configuration data. The experiments demonstrate that non-destructive single event effects can be effectively mitigated with reduced overheads. Results show an important increase in fault coverage for SEUs and SETs, about one order of magnitude.

Index Terms— Single Event Upset (SEU), Single Event Transient (SET), fault tolerance, soft error, radiation effects, design hardening.

1 Introduction

MICROCONTROLLERS are key components in safety-critical and high availability missions, because their programmability, performance and cost-effectiveness. In addition, Commercial Off-The-Shelf electronic components (COTS) offer important capabilities and benefits in the implementation of low-cost systems, and are opening new opportunities in space and avionic industry, such as small satellites [1] or safety systems [2]. However, COTS microcontrollers' main drawback remains the low tolerance to radiation-induced effects. This fact can limit their applicability in the near future [3], [4] and consequently there is an increasing effort focused on developing new hardening techniques for COTS-based systems.

Different approaches have been proposed to tackle this problem. Among them, hardware redundancy is the most usual and effective solution when is applied in qualified RadHard microprocessors. However, this fine-grain redundancy does not fit in COTS components because the impossibility to modify their internal hardware. Coarse-grain alternatives, as duplication or triplication of components [5], have also been explored and even used in real systems, obtaining very good results. These approaches increase the complexity, cost and power consumption; hiding the benefits of COTS components and limiting their use in low-cost and small systems.

Software-based techniques also known as SIHFT (Software Implemented Hardware Fault Tolerance) do not require any modifications in the hardware of the microprocessor and they provide higher flexibility as well as lower development time and cost [6]. These techniques can detect and recover faults that are latent in the data inside the microprocessor (mainly in register file and micro-architecture registers) [14], as well as faults in the control flow [7]. Their main drawbacks are usually related to the overheads incurred in code size and performance degradation.

In this context, configuration data are one of the most fault-sensitive bits of microcontrollers, especially in mission-critical reactive systems which are usually driven by events. The control flow of these programs is determined by diverse event sources synchronized with interrupts. The main algorithm is modeled by means of diverse interrupt service routines (ISR) which react to the different external events. The use of interrupts avoids the need of a constantly polling loop and reduces the computational effort of the microprocessor. The configuration involves two main sets of data. Firstly, data devoted to define the operation of peripherals which generate or acquire the external events, such as timers, I/O ports, A/D converters, UARTs, etc… Secondly, data to define the behavior of interrupts (interrupt vectors, interrupt mask, etc…). It is worth noting the criticality of this configuration data set. For example, the corruption of the interrupt mask registers can have catastrophic consequences because some critical interrupt sources may be blocked or, alternatively, unwanted interrupts may be randomly generated.

- *Antonio Martínez-Álvarez, and Sergio Cuenca-Asensi are with the Computer Technology Department, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain. E-mail: {amartinez, sergio}@dtic.ua.es.*
- *Felipe Restrepo-Calle is with the Department of Systems and Industrial Engineering, Universidad Nacional de Colombia, Bogotá, Colombia (e-mail: ferestrepoca@unal.edu.co).*
- *Leonardo M. Reyneri is with the Department of Electronics and Telecommunications, Politecnico di Torino. Cso. Duca degli Abruzzi 24, 10129 Torino, Italy. E-mail: leonardo.reyneri@polito.it.*
- *Almudena Lindoso and Luis Entrena are with the Department of Electronic Technology, University Carlos III of Madrid, Leganes, Madrid 28911, Spain. E-mail: {alindoso, entrena}@ing.uc3m.es*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2382593, IEEE Transactions on Dependable and Secure Computing

Nevertheless, as far as we know, very little attention has been paid to the protection of configuration data of microcontroller peripherals and interrupts related registers. The fact that in most of the cases the configuration data are stored in some kind of protected memory may be the reason of this lack of attention. Designers have usually addressed this issue adopting as guidelines a set of good practices jointly with ad-hoc solutions [15].

Among them, one of the most usual techniques is blind scrubbing of the whole memory contents [27], which is too time consuming and also stops the operation of the system. This strategy is not sufficient for dependable designs where a short recovery time is required, because during the normal system operation those critical configuration data leave the secure storage of memory and pass to reside in vulnerable dedicated registers. On the other hand, protection by means of traditional software-only techniques may reduce the recovery time, but also may produce a high waste of resources due to the long (or unlimited) lifetime of configuration data.

In this work, we propose a hybrid technique that leverages the advantages of SIHFT and self-scrubbing techniques. It is aimed to the soft error mitigation in the system configuration data, and especially focused on interrupt-driven applications. The proposal is based on the utilization of a common built-in COTS microcontroller resource (timer) that works jointly with a software-based technique, which is responsible to periodically refresh the configuration data. Unlike usual scrubbing approaches, our on-line technique works on the fly without stopping the system. Preliminary results were presented in [24].

In order to assess the reliability provided by the studied approach, a fault injection campaign was carried out on a microcontroller using a non-intrusive tool [8]. The microcontroller was built around the PicoBlaze soft-core microprocessor [9].

2 Motivation

Let us assume p as the constant probability density of having a Single Event Upset (SEU) in a single configuration bit in a given time. Thus, it is measured in [SEU/bit/s] (number of SEU events per bit per second). Let *P(F(x), T)* be the fault probability of a set of configuration registers x having a lifetime of T seconds. It is supposed that n bits of the configuration registers are critical such that SEUs always become a failure. The same might not apply to the rest of bits within a real system.

As reported by [22], the probability of not upsetting during a time t can be modeled by the *0*th order Poisson distribution. Thus, it can be expressed as:

$$P(correct, t) = e^{-tpn}$$

Therefore MTTF (mean time to failure) can be calculated as:

$$MTTF = \frac{1}{\mathrm{np}}$$

Real 16-bit MCU COTS-based systems running applications of some complexity usually need more than 300 configuration bits. To highlight the importance of hardening the configuration bits in an actual system, let us suppose a Texas Instrument MSP430-based system (used in [1][23]) having the following set of configuration bits: 12-bits ADC (36 bits), 2 UART(2 × 50 bits = 100 bits), clock (30 bits), four 8-bit I/O ports (32 × 4 = 128 bits), 2 timers (2 × 20 bits = 40 bits), Interrupt-Enable register (16 bits), Program Counter (PC) + Stack Pointer (SP) initial values (16 + 16 = 32 bits). That is, a total of 382 configuration bits.

Assuming an actual on-orbit probability density of p = 1.20e-4 SEU/bit-day [26], and n = 382 bits, we obtain a MTTF of 0.0597 years (≈ 22 days), which is an extremely high fault-rate for a mission-critical system.

Among the 382 configuration bits, only faults in the PC and the SP registers could be detected by the watchdog timer in a normal system (that is, when no configuration bits are hardened). Therefore only about 8% (32 out of 382) of those would be detected and corrected. Our proposal is focused on the protection of the remaining 92%.

It must be noted that this paper focuses on the protection of configuration bits, namely those which are written by the program itself. In fact, there are also status bits and, in particular, interrupt flags. These cannot be hardened with the proposed technique as they are volatile, that is, the hardware peripheral can autonomously modify them.

Yet the total number of such bits is about one order of magnitude less than configuration bits, therefore about ten times less frequent. Furthermore errors in most configuration bits (e.g., baud rate of a UART) normally cause malfunctions for a long period of time until a new configuration is written, while errors in status bits usually generate temporary faults whose effect ends after a limited amount of time (e.g., a SEU in a UART interrupt flag causes one byte to be duplicated inside a message; after the end of the message, all the following messages are again correctly transferred).

3 Soft-Error Mitigation In The System Configuration Data

The protection of the configuration data for microcontroller peripherals is a must in mission-critical systems. In this work, we address the hardening of configuration registers (for example: clock configuration registers, peripheral configuration registers, and interrupt configuration registers). In particular, hardening the configuration data of interrupts which is a mandatory issue in critical reactive systems.

An application whose functionality is directed by interrupt events, and which is mainly managed by means of a set of ISRs (Interrupt Service Routines) is known as a interrupt-driven application. One can identify two different tasks when hardening this kind of applications based on software: *the ISR-code hardening* itself, and *the system configuration hardening*.

For the first task, let us suppose the ISR-code to be a typical data driven routine such as a control attitude rou-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2382593, IEEE Transactions on Dependable and Secure Computing

tine running in a real-time control flight system. Techniques for hardening this kind of routines have already been presented in [10] and, therefore, they are not within the scope of this paper.

The second and most important task is the *system configuration hardening*. A soft error in the system configuration data is far more disruptive than an error occurring in a non-configuration register (e.g., a bit of a general purpose register used in the ISR-code): the former might prevent a given ISR to be ever called or a full sub-system to be accessed (even after reboot), whereas an error in a general purpose register only affects program functionality occasionally. Hence, this work is aimed to the hardening of the system configuration data.

Regarding the configuration data of a system, as first stated in [1], two different scenarios can be studied taking into account the lifetime or rating access of the involved storage resources. Firstly, the configuration data remain static during the program execution. Secondly, the scenario when the configuration data change dynamically throughout the program execution. It is worth mentioning that in both cases, storage resources do have infinite lifetime or a lifetime much higher than most other registers in the system.

The following subsections present the proposed hardening approach for each scenario.

3.1 Static configuration hardening

The term static refers to the contents of configuration registers which are never expected to change. The static configuration hardening addresses the protection of those peripheral configuration registers which remain unchanged during the whole program execution. For instance, this is the case of systems that statically configure their peripherals or I/O subsystems using specific configuration registers, and storing custom configuration words (e.g., baud rate and modulation for UARTs, acquisition modes for ADCs and DACs, counting mode and period for timers/counters, pin direction for I/O ports).

The proposed technique to harden the static configuration data in event-driven applications is summarized in Fig. 1. Although this approach is mainly based on software, it also needs a hardware timer. Briefly, the approach consists in refreshing the static configuration periodically by means of an additional interrupt service routine that is triggered by a hardware timer.

In order to maximize the fault coverage, the refreshing rate for the configuration data must be tuned according to two parameters: the expected soft error rate, and the application interrupt rate. A high refreshing rate may improve error mitigation while making the whole system less reactive. An optimal refreshing rate should be determined for each application taking into account these considerations.

Since only a single hardware timer is needed, regardless the number of configuration registers to be protected, the introduced overheads are negligible in terms hardware resources.

Main processing	Interrupt processing	
	Application interrupt i	*Hardening timer interrupt*
main:		
Setup conf.		
…	**ISR_i:**	**ISR refresh:**
forever:	Interrupt handler	Setup conf.
wait	…	
jump forever	return from int.	return from int.
end		

Fig. 1. Proposed approach for static configuration hardening in interrupt-driven applications.

3.2 Dynamic configuration hardening

The second scenario for the hardening of event-driven applications concerns those processors or peripheral configuration registers which are occasionally modified during program execution, i.e., the configuration data is dynamic. For instance, an interrupt occasionally enabled/disabled, a change of ADC configuration, etc.

As in the static case, the hardening interrupt service routine is triggered periodically. However, the main difference with respect to the previous case is that the ISR triggered by the hardware timer does not know beforehand what values to refresh into the configuration registers, as they change throughout the program execution.

Hardening the dynamic configuration data requires the actions illustrated in Fig. 2. Firstly, during the main processing is necessary to maintain redundant copies (two) of the configuration data (using protected registers/memory). Every time there are configuration changes, the replicas have to be updated. Notice that data should be copied before proceeding to configure the system. Secondly, the ISR driven by the hardware timer includes *majority voters* to check the configuration registers correctness (using the original registers and their two copies); and finally, it refreshes the configuration data accordingly.

Next section presents a case study for both cases: hardening static and dynamic system configuration of an event-driven application.

Main processing	Interrupt processing	
	Application interrupt i	*Hardening timer interrupt*
main:		
Setup initial conf.		
…	**ISR_i:**	**ISR refresh:**
forever:	Int. handler	Majority voter
if (new conf.)	…	Refresh conf.
Replicate conf.	return from int.	
Conf. update		return from int.
end if		
jump forever		
end		

Fig. 2. Proposed approach for dynamic configuration hardening in interrupt-driven applications.

4 Case Study

4.1 Experimental setup

The microcontroller used for the experimental setup was built around the *PicoBlaze* microprocessor [9]. The 8-bit soft-core microprocessor used in this experiment is a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2382593, IEEE Transactions on Dependable and Secure Computing

technology-independent version, especially developed for this work (*RTL PicoBlaze)*. This version is cycle accurate and RTL equivalent to the original *PicoBlaze-3* version.

The main features of the microprocessor are: 16 byte-wide general-purpose data registers, byte-wide Arithmetic Logic Unit with *CARRY* and *ZERO* indicator flags, 64-byte internal scratchpad *RAM*, 256 input and 256 output ports, 31-location *CALL/RETURN* stack, 1 interrupt input, *INTERRUPT ENABLE* (*IE*) indicator flag.

Besides the 1K instructions (10 bits) of programmable on-chip program store, additional features have been implemented externally to provide *PicoBlaze* with some of the microcontroller common resources: 1 interrupt controller (*int_ctlr*), 2 timers (*timer0* and *timer1*), and several I/O ports. In addition, an LFSR (Linear Feedback Shift Register) module has been included in the circuit. The block diagram of the circuit can be seen in Fig. 3.

The interrupt controller (*int_ctrl*) is a peripheral that extends the capability of *PicoBlaze* to manage up to eight interrupt sources. It should be enabled and properly configured from the software. The configuration of interrupts has two hierarchical levels: the *Interrupt Enable (IE)* flag which is a global enable/disable control for the microprocessor interrupts, and the *interrupt_mask* which is an 8-bit register to enable/disable each interrupt line in the *int_ctrl*.

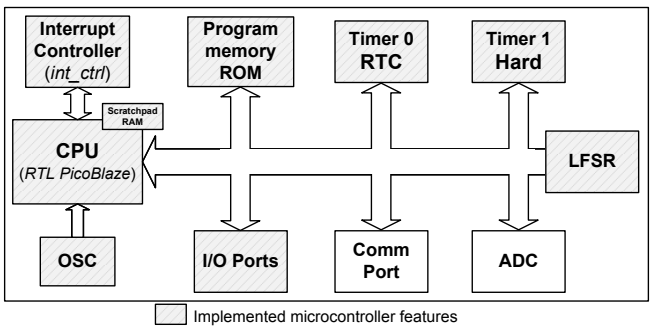

Fig. 3. Extended Picoblaze microcontroller block diagram.

Each timer can be configured to generate a system "*tick*" (interrupt) in the range of 1µs-10ms, by means of an 8-bit register called *timerX_conf* (where *X* is equal to 0 or 1). The interrupt signal of every peripheral is automatically cleared once the *interrupt_ack* has been received from the *PicoBlaze*.

The LFSR module will be used in the second part of this case study. It generates a bit sequence that will be responsible to emulate dynamic changes to the interrupt controller configuration.

This case study comprises two parts. The first one is focused on a scenario for the static configuration case, whereas the second part presents a dynamic configuration scenario.

Static configuration scenario

Using the resources described above, a critical reactive application has been developed, namely a *Real Time Clock* (*RTC*). In this application, configuration registers remain unchanged during program execution, so it has a static configuration with infinite lifetime.

Fig. 4 presents the flowchart of this application. Hereafter, this version will be called *non-hardened static RTC*. This is a typical interrupt-driven application, where the main processing is only responsible for the configuration of the microcontroller and its peripherals. The *RTC* application logic is implemented by the "*Interrupt handler RTC*" procedure within the *ISR*. Configuration data for this application is comprised of the following registers: *IE flag*, *interrupt_mask* (*int_ctrl*), and *timer0_conf*.

According to our proposal, Fig. 5 shows the flowchart of the hardened static RTC application. The hardening strategy is hybrid. On the hardware side, an additional timer (timer1) is used; whereas on the software side, three transformations have been applied to the original code.

Transformation 1 is responsible for the new peripheral configuration. Firstly, the interrupt controller is configured with a new value for the interrupt_mask to enable interrupts coming from timer1 (besides those coming from timer0). Secondly, the "hard timer" (timer1) is configured according to the required refresh configuration rate.

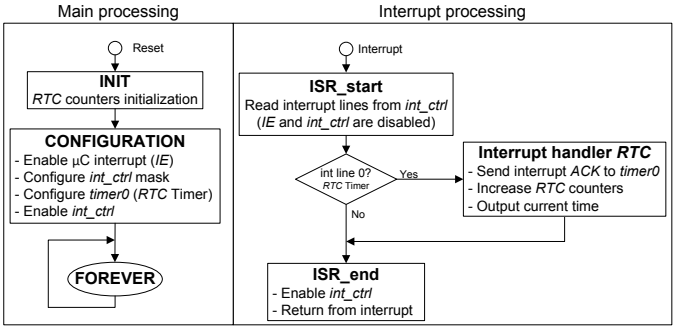

Fig. 4. Flowchart describing the non-hardened static RTC application (main and interrupt processing).

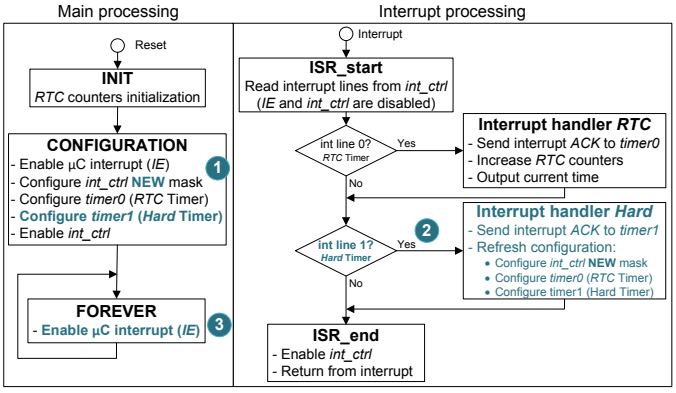

Fig. 5. Flowchart describing the hardened static RTC application.

Transformation 2 does not affect the existing "*Interrupt handler RTC*"; it only appends a new handler for the "*hard timer*" interrupt. This handler sends the ACK to the interrupt source and refreshes the static configuration registers (*interrupt_mask* in the *int_ctrl*, *timer0_conf*, and *tim-*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2382593, IEEE Transactions on Dependable and Secure Computing

er1_conf). It is worth noting that interrupt priority is defined by the designer within the code. In this case, since the *RTC* is a real time application, the "*Interrupt handler RTC*" has the highest priority.

Finally, *Transformation 3* includes an instruction to enable the microcontroller interrupt flag (*IE*) within the infinite loop in the main process. This will not affect the main process, which instead of doing nothing will be refreshing the interrupt flag. In case a soft error affects the *IE* flag, it will recover its correct value the next time the "*forever*" loop is executed. Otherwise, if a fault disables this flag, future interrupts will be ignored.

It must be noted that the additional timer used in our hardening approach (*hard timer*), can protect the static configuration of several peripherals simultaneously.

Dynamic configuration scenario

The experimental setup for this scenario is the same as in the previous case but including the LFSR module, which is responsible to emulate the dynamic nature of the system configuration. This module is connected to a *PicoBlaze* input port, and depending on every value of the generated sequence, it determines to enable/disable the interrupt controller dynamically. Fig. 6 depicts the flowchart of this application, which is called *non-hardened dynamic RTC*. The differences with respect to the static case can be seen in the main processing, specifically in the three emphasized blocks.

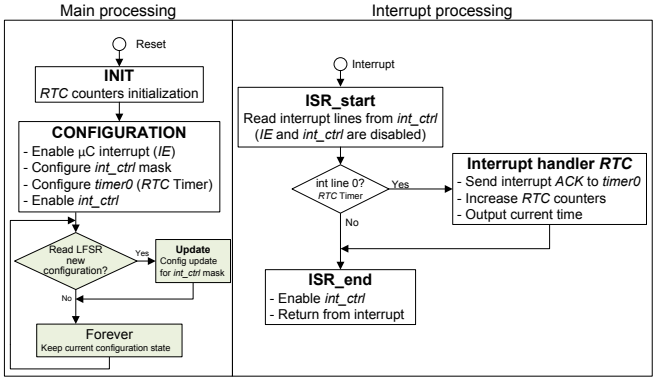

Fig. 6. Flowchart describing the non-hardened dynamic RTC application (main and interrupt processing).

As explained in Section III.B., in this case, additionally to having a refreshing interrupt service routine which is triggered periodically (as in the static case), the protection for the dynamic configuration consists in maintaining replicated copies of the current configuration. In this way, configuration correctness is checked every time before proceeding to refresh the system configuration by means of a software-based majority voter. Fig. 7 illustrates the flowchart describing the *hardened dynamic RTC* application.

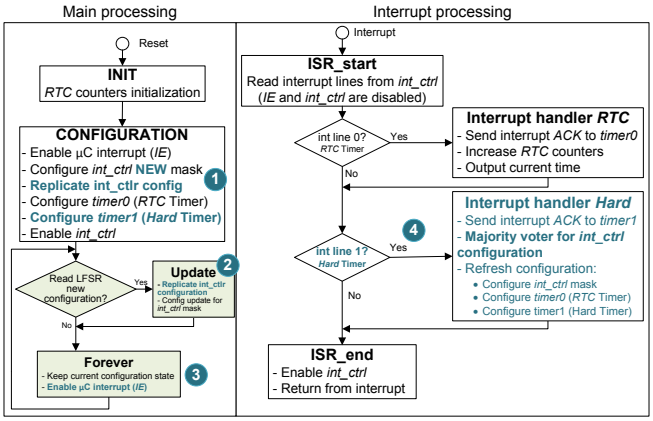

Fig. 7. Flowchart describing the hardened dynamic RTC application.

As in the static case, *Transformation 1* is responsible for the new peripheral configuration (*timer1 - Hard Timer*). In addition, notice that the configuration should be replicated as well. In this particular case study, we replicate the configuration of the interrupt controller (*int_ctlr*). Copies should be stored in available memory, which can be located externally with its own protection mechanisms. In the experiments, the worst case scenario was implemented since copies are stored within the microprocessor register file.

Transformation 2 inserts replication instructions to create copies every time the configuration changes. Note that replicas should be created before updating the system configuration because it avoids overwriting the correct new configuration value from the periodic refreshing ISR.

Similarly to the static case, *Transformation 3* enables the microcontroller interrupt flag (*IE*) constantly.

Transformation 4 appends a new handler for the "*hard timer*" interrupt. This handler includes a majority voter procedure to detect and recover any possible data corruption in the configuration. Then, it refreshes the proper configuration registers.

Although the code size overhead caused by this technique is higher than the one produced in the static case, in both cases these overheads are negligible compared to usual software-based techniques [6]. The additional code required to implement the proposed approach is in the order of 15 instructions for the static case and 34 instructions for the dynamic case, including the ISR for the hard-timer. Unlike conventional software-based techniques, the proposed transformations are focused only in the mentioned pieces of the source code, and they are not applied to the entire program.

4.2 SEU/SET Emulation System (AMUSE)

Fault injection is commonly used in COTS microprocessors to evaluate the error rate. A fault is injected by changing the contents of a register. This action can be triggered by software [12], interrupts [11] or breakpoints [13]. However, these approaches cannot be used in our case, because they are highly intrusive and can interfere

with the generation of true system events. To overcome these limitations, we used the AMUSE (Autonomous MUltilevel emulation System for Soft Error evaluation) emulation system [8][18][19][20][21].

AMUSE is an emulation-based fault injection system that supports SEU and SET fault injection for any ASIC technology. For SETs, pulses of a selected duration can be injected at any node in the circuit and their propagation analyzed across many clock cycles. Since the propagation of a pulse across combinational logic is delay-dependent, ASIC delays must be properly modeled. To this purpose, AMUSE uses a voltage-time quantization approach that accurately models dynamic delay effects, including electrical masking effects [19]. Quantized rising/falling transition curves are implemented by a non-linear counter driven by a time quantization clock. This approach enables embedding ASIC delays into a model that can be synthesized and mapped into a FPGA to speed up the fault injection process. Furthermore, AMUSE combines a delay-accurate gate-level model with a fast cycle-accurate register transfer level model to improve performance without loss of accuracy [8]. Thanks to this multilevel scheme, very high fault injection rates are obtained and very large fault injection campaigns can be executed in a short time.

Since our experimental setup is based on a soft-core microcontroller, it can be used within the FPGA-based fault emulation system AMUSE in order to exhaustively evaluate the fault coverage in real conditions. Hence, fault injection tests with a large number of faults can be carried out to obtain statistically representative reliability results.

5 Experimental Results and Discussion

The proposed approach has been validated by fault injection using the AMUSE emulation system. The extended Picoblaze microcontroller was synthesized for the 90nm ASIC library technology provided by Synopsys (SAED90nm) [25]. Fault injection campaigns were performed for several software versions with different interrupt and refresh rates, also including the original, unprotected software version. The results presented in this paper focus only on the evaluation of the proposed approach to harden the configuration data of microcontroller peripherals and interrupts. The SEU and SET sensitivities of the microprocessor core using partial hardware and software hardening have been analyzed previously in [16] and [17], respectively.

For the sake of comparison, each hardened software version is characterized by the ratio of the refresh rate to the interrupt rate. We refer to this ratio as the Relative Refresh Rate (RRR). For instance, RRR = 1 means the configuration registers are refreshed on average as often as interrupts occur. If the configuration registers are refreshed several times between two interrupts, then RRR is greater than 1. Conversely, a RRR value smaller than 1 means the refresh rate is smaller than the interrupt rate and that several interrupts may occur without refreshing the configuration registers. The non-hardened version is characterized by RRR = 0, i.e., no refresh at all.

The refresh rate is determined by the "hard timer" configuration. The interrupt service routine of the "hard timer" requires 64 clock cycles to execute and refresh the peripheral registers. A critical situation may happen when there is a collision between the "RTC timer" and the "hard timer" interrupts. For simplicity, we considered that a delay of 100 clock cycles in servicing the "RTC timer" interrupt is acceptable for the application. Thus, in case the "RTC timer" interrupt signal arrives while the "hard timer" interrupt is being serviced, the "RTC timer" interrupt service will be slightly delayed. Interrupt priorities can be used if the "RTC timer" interrupt cannot be delayed.

5.1 Static configuration results

For the first experiments, we set the RTC interrupt rate to 500 clock cycles and the refresh rate to several values between 5000 clock cycles (RRR = 0.1) and 50 clock cycles (RRR = 10). The original version with no refresh (RRR = 0, hard timer disabled) was also included in the experiments. The results for SEU and SET fault injection are summarized in Figs. 8 and 9, respectively.

For each refresh rate, we run the Real Time Clock application for 65,000 clock cycles, servicing 130 RTC interrupt events. Along this time, we injected 12,534 random SETs and SEUs into every gate and flip-flop, respectively, of the peripheral circuit, which includes the timers and the interrupt controller. SEUs were also injected into the interrupt enable flag register of the microcontroller, since this register is critical for the peripheral operation. For gates, SET pulses of 500 ps were injected at random instants. For flip-flops, SEUs were injected at random clock cycles. The total number of faults injected in each version was 5,414,688 faults.

The results in Fig. 8 show that the SEU sensitivity can be reduced from 13.4% in the original circuit to 1.3% in the case of RRR = 10. As expected, the error rate reduces when the refresh rate increases, and the reduction is significantly higher for RRR greater than 1. Remarkably, the interrupt enable flag produced no error in all hardened versions while it has a sensitivity of 85% in the original non-hardened version. The results for SETs in Fig. 9 are very similar, but with smaller error rates.

In practical cases, very high error mitigation can be obtained since the time between interrupts is typically much larger than the one we used in the experiments (500 clock cycles). The opportunity window for an error to appear is then reduced to the time between the last refresh and the next application interrupt, which can be made very small by increasing the refresh rate.

We have also checked that errors in the hard-timer and errors due the execution of the hardening ISR are mostly

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2382593, IEEE Transactions on Dependable and Secure Computing

benign, because the effect in most cases is just a temporary change in the refresh rate, which is eventually corrected when the ISR is executed again. For instance, the hardening ISR uses a single register (s0) to configure the refresh rate, and the SEU error sensitivity of this register varies from 0.37% to 0.69% when the RRR varies between 0 and 10 respectively. This result has been obtained when this register is not hardened at all. These errors can be easily removed by software hardening of the ISR.

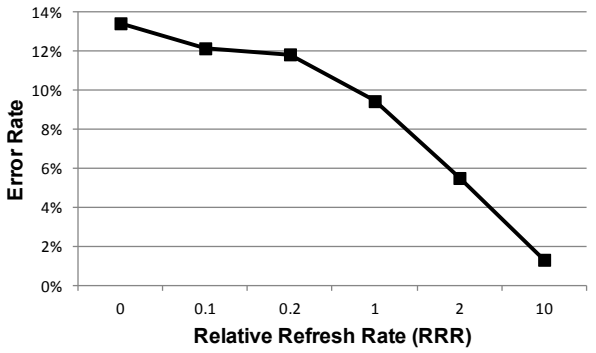

Fig. 8. SEU Error Rates for static configuration and variable refresh rate

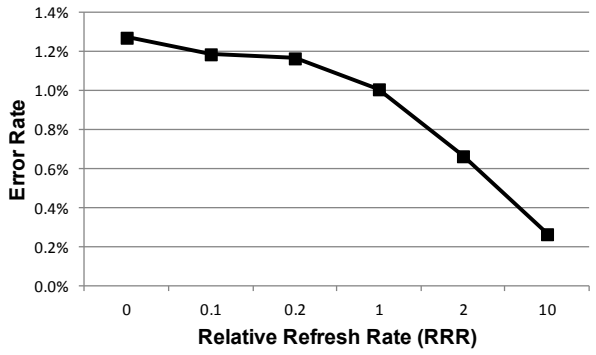

Fig. 9. SET Error Rates for static configuration and variable refresh rate

In a second set of experiments, the refresh rate was set to the minimum feasible value of 100 clock cycles, while the RTC interrupt rate was varied to evaluate how the error rate decreases as interrupts occur less frequently. Fig. 10 shows the SEU error rate in this case. For RRR = 1, the error rate is smaller than in Fig. 8 because the refresh rate is higher now. For RRR higher than 25 (2500 clock cycles between interrupts), the error coverage is higher than 99%.

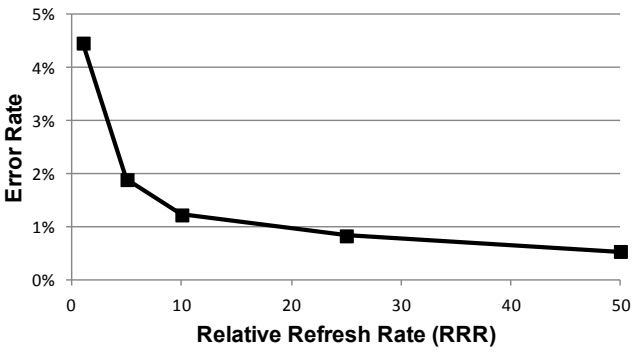

Fig. 10. SEU Error Rates for static configuration and fixed refresh rate

5.2 Dynamic configuration results

Figure 11 shows the results for the dynamic configuration case. As in the first experiment, the RTC interrupt rate was set to 500 clock cycles, but interrupts were randomly enabled/disabled, depending on the bit sequence generated by the peripheral LFSR. The RRR is referred to the average interrupt rate. Although there is no fixed interrupt rate, the proposed approach is similarly effective: the higher the refresh rate, the higher the mitigation.

As it can be seen, the error rate is higher than in the static case, but presents a similar trend reaching to an important decrease of errors for moderate refresh rates.

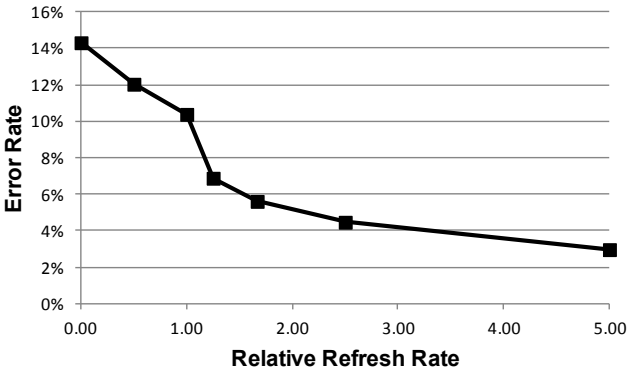

Fig.11. SEU Error Rates for dynamic configuration and variable refresh rate

An additional experiment was carried out injecting faults into the configuration register replicas (located in the microprocessor register file). Error rate did not vary significantly due to the software redundancy.

A more detailed analysis of the results in both cases, static and dynamic, points out that only one single internal register of the interrupt controller is the source of more than 80% of the residual errors (those ones that are not mitigated using high RRR). This specific register is not accessible from the software and, therefore, determines the maximum fault coverage that can be achieved with this technique, above 99%. If this register is hardened by hardware, the error rate reduces by more than one additional order of magnitude.

Regarding the overheads introduced by the technique, on the software side, the increase in the code size comprises the insertion of a few lines of code responsible to: rewrite the configuration data and its replicas, implement the voter procedure, and handle the hardening interrupt event. On the hardware side, the overhead only supposes one additional timer, regardless the number of configuration data to protect. In addition, there is no performance overhead because the execution of the refreshing procedure takes place when the main processing is not being executed. However, there is a possible low impact in the system responsiveness. This occurs only in case the hardening interrupt routine is being executed when the application interrupt arrives. This supposes to delay the system response for the number of clock cycles until the hardening ISR finishes its work (in the worst case in terms of latency, 64 and 94 clock cycles respectively for the static and dynamic scenarios). Moreover, power consumption overhead is a must issue to be addressed in the design of embedded systems with tight power budget. Although the repetitive execution of the refreshing routine increases the power consumption, this overhead is negligible for low RRR because it comprises only a few instructions. In case of high RRR causing excessive power demand, designers should always find the best trade-offs among RRR, fault coverage, and power to cope with the system requirements.

6 Conclusions and Future Work

The use of Commercial Off-The-Shelf components in the implementation of low-cost safety-critical systems offers important benefits. However, it is necessary to reduce their vulnerability to radiation-induced effects, such as soft-errors. This work proposes a systematic approach for soft-error mitigation that attempts to overcome the limitations of common ad-hoc solutions. It is focused on the protection of the configuration registers, which are the most critical resources in interrupt-driven applications. Both possible cases, static and dynamic configuration, were studied.

The technique was assessed by extensive fault injection campaigns for SEUs and SETs. The impact in code size and cost was negligible independently of the number of configuration registers to be protected. It is worth noting that a single timer can be used to protect several configuration registers. The results show an important increase in fault coverage for SEUs and SETs, about one order of magnitude. From the experiments, it is deduced that the refresh rate and the interrupt rate are directly related in static and dynamic cases. Regarding the final application, the RRR can be used to find the best trade-off between reliability and system responsiveness. The proposed approach can be combined with other hardening approaches [10] to protect the complete microprocessor-based system.

Acknowledgment

This work was funded in part by the Spanish Ministry of Education, Culture and Sports with the project "Developing hybrid fault tolerance techniques for embedded microprocessors" (PHB2012-0158-PC).

References

[1] Reyneri, L.M.; Roascio, D.; Passerone, C.; Iannone, S.; De los Rios, J.C.; Capovilla, G.; Martínez-Álvarez, A.; and Hurtado, J.A. "Modularity and Reliability in Low Cost AOCSs", *Advances in Spacecraft Systems and Orbit Determination*, Dr. Rushi Ghadawala (Ed.), InTech, March 2012.

[2] Pignol, M. "COTS-based applications in space avionics," In Proc. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, vol., no., pp.1213-1219, 8-12 March 2010.

[3] Edwards, R.; Dyer, C.; and Normand, E. "Technical standard for atmospheric radiation single event effects (SEE) on avionics electronics". In Proc. *IEEE Radiation Effects Data Workshop (REDW)*, pages 1–5. IEEE, 2004.

[4] Michalak, SE; Harris, KW; Hengartner, NW; Takala, BE; and Wender, SA. "Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer". *IEEE Transactions on Device and Materials Reliability*, 5(3):329–335, SEP 2005.

[5] Pignol, M. "How to cope with SEU/SET at system level?," *In Proc. 11th IEEE International On-Line Testing Symposium*, 2005. IOLTS 2005, pp. 315- 318, 6-8 July 2005.

[6] Azambuja, J. R.; Sousa, F.; Rosa, L.; Kastensmidt, F. L. "Evaluating the efficiency of software-only techniques to detect SEU and SET in microprocessors", *In Proc. IEEE Latin American Symposium on Circuits and Systems*, 2010.

[7] Oh, N.; Shirvani, P.P.; and McCluskey, E.J. "Control-Flow Checking by Software Signatures," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp. 111-122, Mar. 2002.

[8] Entrena, L.; Garcia-Valderas, M.; Fernandez-Cardenal, R.; Lindoso, A.; Portela Garcia, M.; Lopez-Ongil, C. "Soft Error Sensitivity Evaluation of Microprocessors by Multilevel Emulation-Based Fault Injection," *IEEE Transactions on Computers*, pp. 313-322, March, 2012.

[9] XILINX. "PicoBlaze 8-bit Embedded Microcontroller User Guide". UG129 (v1.1.2). Xilinx Ltd., June 2008.

[10] Martínez-Álvarez, A.; Cuenca-Asensi, S.; Restrepo-Calle, F.; Palomo Pinto, F.R.; Guzmán-Miranda, H.; Aguirre, M.A. "Compiler-Directed Soft Error Mitigation for Embedded Systems," IEEE Transactions on Dependable and Secure Computing, vol. 9, no. 2, pp. 159-172, March-April 2012,

[11] Velazco, R.; Rezgui, S.; and Ecoffet, R. "Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection", IEEE Transactions on Nuclear Science, vol. 47, pp. 2405–2411, Dec. 2000.

[12] Benso, A.; Rebaudendo, M.; Sonza-Reorda, M. "Fault Injection for Embedded Microprocessor-based Systems". *Journal of Universal Computer Science*, vol. 5, no. 10, pp. 693-711, 1999.

[13] Portela-García, M.; Lopez-Ongil, C.; García-Valderas, M.; Entrena, L. "Fault Injection in Modern Microprocessors Using On-Chip Debugging Infrastructures". *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, 308-314, March 2011.

[14] Reis, G. A.; Chang, J.; and August, D. I. "Automatic instruction-level software-only recovery," *IEEE Micro*, vol. 27, no. 1, pp. 36–47, 2007.

[15] "NASA Software Safety Guidebook". NASA-GB-8719.13, March 2004.

[16] Cuenca-Asensi, S.; Martínez-Álvarez, A.; Restrepo-Calle, F.; Palomo, F. R.; Guzmán-Miranda, H.; Aguirre, M. A.; "A Novel Co-Design Approach for Soft Errors Mitigation in Embedded Systems," *IEEE Transactions on Nuclear Science*, Vol. 58, no. 3, pp. 1059-1065, June 2011.

[17] Lindoso, A.; Entrena, L.; Millan, E. S.; Cuenca-Asensi, S.; Martínez-Álvarez, A.; Restrepo-Calle, F.; "A Co-Design Approach for SET Mitigation in Embedded Systems," *IEEE Transactions on Nuclear Science*, vol.59, no.4, pp.1034-1039, Aug. 2012

[18] Lopez-Ongil, C.; Garcia-Valderas, M.; Portela-Garcia, M.; Entrena, L.; , "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation," *IEEE Transactions on Nuclear Science,* vol.54, no.1, pp.252-261, Feb. 2007

[19] Entrena, L.; García-Valderas, M.; Cardenal, R.F.; Portela-Garcia, M.; López-Ongil, C.; , "SET Emulation Considering Electrical Masking Effects," *IEEE Transactions on Nuclear Science,* vol.56, no.4, pp.2021-2025, Aug. 2009

[20] Entrena, L.; Lindoso, A.; García-Valderas, M.; Portela, M.; López-Ongil, C.; , "Analysis of SET Effects in a PIC Microprocessor for Selective Hardening," *IEEE Transactions on Nuclear Science,* vol.58, no.3, pp.1078-1085, June 2011

[21] Pagliarini, S.; Kastensmidt, F.; Entrena, L.; Lindoso, A.; Millan, E.S.; , "Analyzing the Impact of Single-Event-Induced Charge Sharing in Complex Circuits," *IEEE Transactions on Nuclear Science*, vol.58, no.6, pp.2768-2775, Dec. 2011.

[22] Bajura, M.A.; Boulghassoul, Y..; Naseer, R..; DasGupta, S..; Witulski, A.F.; Sondeen, J..; Stansberry, S.D.; Draper, J..; Massengill, L.W.; Damoulakis, J.N.; , "Models and Algorithmic Limits for an ECC-Based Approach to Hardening Sub-100-nm SRAMs," *IEEE Transactions on Nuclear Science*, vol.54, no.4, pp.935-945, Aug. 2007.

[23] Del Corso, D.; Passerone, C.; Reyneri, L.; Sansoe, C.; Speretta, S.; Tranchero, M.; , "Design of a University Nano-Satellite: the PiCPoT Case," *IEEE Transactions on Aerospace and Electronic Systems*, vol.47, no.3, pp.1985-2007, July 2011.

[24] A. Martínez-Álvarez, F. Restrepo-Calle, S. Cuenca-Asensi, L. Reinery, A. Lindoso, L. Entrena. A hybrid technique for soft error mitigation in interrupt-driven applications. In Proc. of the *13th European Conference on Radiation and its Effects on Components and Systems RADECS 2012*. Biarritz, France. Sept 24 - 28, 2012.

[25] SAED 90nm Generic Library, Synopsys Armenia Educational Department, [Online]. Available: http://www.synopsys.com/Community/UniversityProgram

[26] Michael A. Cosgrove, "*Using a System-Level Bit-Error-Rate Model to Predict On-Orbit Performance*", IEEE Transactions on Nuclear Science, vol. 50, NO. 6, pp. 2352–2357, Dec. 2003.

[27] A. Saleh, 1. Serrano, and J. Patel, *"Reliability of scrubbing recovery techniques for memory systems,"* IEEE Transactions on Reliability, vol. 39, no. 1, pp. 114-122, Apr 1990.

Antonio Martínez-Álvarez received the MS and PhD degree in electronics engineering from the University of Granada, Spain, in 2002 and 2006, respectively. From 2002 to 2006 he joined the Department of Computer Architecture and Technology at the University of Granada, Spain. He is currently an associate professor with the Department of Computer Technology, University of Alicante, Spain. His main research interests deal with methods and tools for dependable design of digital integrated circuits and FPGAs, embedded systems based on reconfigurable devices, bioinspired high-performance image-processing architectures, and automatic hardware generation of bioinspired visual systems. He is also interested in neuroengineering and neuroprosthesis devices. Currently, he is working in the design and development of SHE (Software Hardening Environment).

Felipe Restrepo-Calle was born in Pereira, Colombia in 1981. He received his Systems and Computer Engineering degree and M.Sc. Physics Instrumentation degree cum laude by the Universidad Tecnológica de Pereira (Colombia) in 2004 and 2011, respectively. He received the PhD degree cum laude from the University of Alicante (Spain) in 2011. He worked as a postdoctoral research fellow in the University of Seville (Spain) during 2012 and 2013. In addition, he did a postdoctoral stay during the first semester of 2014 in the University Federal Rio Grande do Sul (Porto Alegre, Brazil). Since August of 2014 he joined the Department of Systems and Industrial Engineering at the National University of Colombia (Bogotá, Colombia). His field of interest is related with methods and tools for dependable design in embedded systems. He is also interested in programming languages theory and assistive technologies. Currently, he is working in the design and development of SHE (Software Hardening Environment).

Sergio Cuenca-Asensi is an Associate Professor in the Computer Technology and Computation Department at University of Alicante, Spain. He received the B.S. degree in Electronic Physics in 1990 from University of Granada, Spain. He received a Ph.D. in Computer Engineering from the University Miguel Hernández of Elche, Spain, in 2002. His current research interests are reconfigurable computing, Multi-Objective Optimization, high-performance image-processing architectures, hardware/software co-design and soft error mitigation in embedded systems.

Leonardo M. Reyneri received the M.Sc. degree cum laude at Politecnico di Torino in 1984 and the Ph.D. in 1992. He is Associated Professor of Electronics at the Faculty of Engineering of Politecnico di Torino and is currently researching in the fields of VLSI design, HW/SW and mixed-signal co-design and co-simulation, parallel computing and real-time signal processing, neuro-fuzzy networks and their applications to industry and agriculture. He has worked on the development of tools and methods for co-design and co-simulation of HW/SW and mixed-signal systems, and on the design of systems centered on digital and analog FPGAs, with particular interest in real time signal processing and telecommunication systems. He is also working on industrial and agricultural applications of neuro-fuzzy classification, characterization, and forecasting techniques. He has also coordinated and participated to the development of an HW/SW co-design environment based on Simulink, aimed at high-performance systems. Dr. Reyneri has published more than 160 papers and he holds 6 patents. He has also been guest editor and referee of international magazines and conferences and program and steering chairman of international conferences. He has also spent periods of time at the European Space Agency and at the Universities of Pisa, Edinburgh, and Granada.

Almudena Lindoso received the MS degree in 2001 from the University Politécnica of Madrid and the PhD degree in 2009 from the University Carlos III of Madrid. Since 2003, she has been an assistant professor and a researcher in the Electronic Technology Department at University Carlos III of Madrid. Her current research interests include image processing, fault tolerance, hardware acceleration, and reconfigurable devices.

Luis Entrena received the MS degree from Universidad de Valladolid (Spain) in 1989, and the PhD degree from Universidad Politecnica de Madrid (Spain) in 1995. From 1990 to 1993 he was with AT&T Microelectronics at Bell Labs (USA). From 1993 to 1996 he was with TGI, Spain. Since 1996 he is Associate Professor at Universidad Carlos III de Madrid (Spain), where he has been head of the Electronic Technology Department. He has worked on the development of fast fault injection methods and efficient soft error mitigation techniques. Dr. Entrena has published more than 100 papers and holds 1 patent for the invention of the redundancy addition and removal method of logic optimization. His current research interests include on-line testing, fault tolerance, soft error mitigation and hardware acceleration.