Dynamically Adaptive Scrubbing Mechanism for Improved Reliability in Reconfigurable Embedded Systems

Rui Santos
National University of Singapore
elergvds@nus.edu.sg

Shyamsundar Venkataraman
National University of Singapore
shyam@nus.edu.sg

Akash Kumar
National University of Singapore
akash@nus.edu.sg

ABSTRACT

Commercial off-the-shelf (COTS) reconfigurable devices have been recognized as one of the most suitable processing devices to be applied in satellites, since they can satisfy and combine their most important requirements, namely processing performance, reconfigurability and low cost. However, COTS reconfigurable devices, in particular Static-RAM Field Programmable Gate Arrays (FPGAs), can be affected by cosmic radiation, compromising the overall satellite reliability. Scrubbing has been proposed as a mechanism to repair faults in configuration memory. However, the current scrubbing mechanisms are predominantly static and unable to adapt to run-time variations in applications. In this paper, a dynamically adaptive scrubbing mechanism is proposed. Through a window-based scrubbing scheduling, this mechanism adapts the scrubbing process to the reconfigurations and modifications on the FPGA user-design at runtime. Conducted simulation experiments show the feasibility and the efficiency of the proposed solution in terms of system reliability and memory overhead.

1. INTRODUCTION

Satellites have to deal with increasingly tight requirements regarding their processing systems. They have to combine high performance, high reliability, operational flexibility and adaptability during the execution, and low cost. In particular, satellites are requiring more and more processing and operational performance, since their payload subsystems have to deal with increasingly complex sensing applications. These sensing applications usually require an efficient execution on the same system of image and video processing, software defined radio and generic signal processing. They also produce a large amount of information that has to be processed locally, avoiding the transmission of large amounts of data to a ground station, through the conventional transmission channels that are increasingly saturated. At the same time, reliability is always a special concern in space systems, since satellites are subjected to extreme radiation effects that often lead to malfunction in processes and loss of information. Therefore, the processing system has to implement suitable mechanisms to mitigate erroneous states. Moreover, the processing system should also provide mechanisms to allow runtime adaptations that will be very useful after the satellite launch. This way, satellites can easily be functionally adapted to any unexpected scenario that was not considered at design time. For instance, an efficient implementation of a synthetic aperture radar through the space-time-adaptive processing algorithms requires constant dynamic reconfigurability [12]. Finally, the cost and the resource efficiency are also a great concern. Therefore, the use of commercial off-the-shelf (COTS) devices has been considered as a good option to reduce the overall satellite costs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744827 .

Taking into account all these concerns, COTS Static-RAM (SRAM)-based Field Programmable Gate Arrays (FPGAs) have been considered as the ideal processing devices to be applied in satellites. They offer great operation capacity and performance, combined with reconfigurable properties. This latter allows changing or adapting the satellite functionalities after its launch. Moreover, regarding the costs, COTS SRAM-based FPGAs are widely spread and they have a wide range of tools that allow fast development and lower prices. However, in terms of reliability, COTS SRAM-based FPGAs were not originally developed to be placed in space. In space, FPGAs may be affected by charged particles that strike the silicon substrate. These events called Single Event Upsets (SEUs) can inadvertently change the configuration memory, corrupting the function results and device outputs. In order to overcome this drawback, several fault-tolerant mechanisms to be applied on the FPGA scope have been proposed. However, the majority of these fault-tolerant mechanisms are predominately static, i.e., they do not efficiently support runtime adaptation and reconfiguration on the satellite payload, jeopardising the overall system reliability.

Key contributions: This paper proposes a dynamically adaptive scrubbing schedule mechanism for mixed-criticality systems executed on reconfigurable embedded systems. This mechanism schedules scrubbing executions based on windows in runtime, following the fixed priority scheduling. As a result, this mechanism enables a higher reactive scrubbing system that adapts the scrubbing executions to the user task set reconfigurations and modifications in runtime. Moreover, this solution reduces significantly the amount of memory required to store the scrubbing schedule (at least 16 times better – for the highest level of system reliability).

The rest of the paper is organised as follows. Section 2 presents the background concerning the scrubbing mechanisms and the motivation for this work. Section 3 intro-

duces the system model. Section 4 presents the proposed adaptive scrubbing approach. In Section 5, the conducted experiments are introduced and their results are discussed. Finally, Section 6 presents the conclusions.

2. RELATED WORKS

2.1 Scrubbing Related Works

Scrubbing [6] is a mechanism used to repair faults on an FPGA that takes advantage of the FPGA reconfiguration capabilities. The FPGA reconfiguration is possible through the FPGA internal configuration access port (ICAP) that allows the reading and writing of the FPGA configuration frames, the lowest reconfigurable granular blocks found in an FPGA. Several fault-tolerance solutions have been developed around this mechanism with the simplest approach being blind scrubbing [4]. This solution does not detect the existence of faults on the FPGA, but it periodically rewrites the configuration frames (bitstream file) onto the FPGA instead, overwriting possible faulty bits caused by SEUs. The entire FPGA is scrubbed blindly without considering the tasks that have been implemented on it and the respective used configuration frames. An external memory with continuous access is required to store the original configuration frames, frequently called *golden copy*. Readback scrubbing is another solution, which enables fault detection, reading frame-by-frame the configuration data from the FPGA and then performing a bit-for-bit comparison to the original frames stored in the external memory (*golden copy*). Another alternative combines readback scrubbing with Error Correction Codes (ECCs) [1, 10, 8, 13]. This approach enables fault detection by reading the configuration data frame-by-frame, computing their error correction codes (ECCs) and comparing them to the original ones previously computed and stored externally for each frame. Nazar *et al.* [9] propose a mechanism that statistically finds the optimal frame to start the scrubbing, which reduces the mean time to repair a certain fault. All these scrubbing mechanisms are independent of the user tasks implemented on the FPGA. The FPGA is scrubbed subsequently with a constant and static rate defined, before the execution of the system. As a result, the reliability of the system is not maximized and scrubbing utilization is wasted. To solve part of this drawback, Santos *et al.* [11] propose a scrubbing mechanism that improves the reliability of the system. In order to achieve that, the scrubbing process is scheduled according to the criticality of the user tasks and also scheduled as close as possible of their executions. This way, the probability of the user tasks being affected by a fault is reduced. However, this solution is also static, i.e., the scrubbing scheduled is computed offline and during the user task execution it does not reflect on the scrubbing schedule modification/reconfigurations that may occur on the user task set. The memory required to store the scrubbing schedule may be large, since this solution is dependent on the least common multiple (LCM) among the scrubbing task periods on the system. Moreover, the task model considered in this solution is limited, since it assumes that the user tasks are strictly periodic, executing in well defined instants.

2.2 Motivation Example

In order to better understand the motivation for this work, please consider a simple example described in Figure 1. Initially, two user tasks (τ_1 and τ_2) are implemented on an FPGA system, where the ICAP module provides 90% of its

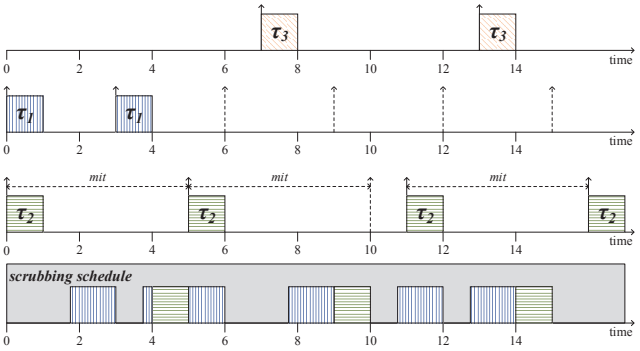

Figure 1: Motivation example.

capacity to the scrubbing mechanism. The task τ_1 executes periodically with a period equal to 3 *time units* and τ_2 executes sporadically with a minimum inter-arrival time (*mit*) equal to 5 *time units*. If the scrubbing schedule is obtained according to the technique proposed in [11], the scrubbing schedule has to be computed by the Earliest Deadline as Late as possible (EDL) algorithm for the LCM interval defined among the scrubbing periods (only 15 *time units* in this example, but it can be very large depending on the computed scrubbing periods). The scrubbing schedule is stored in an external memory and used at runtime during the user task execution. This dependency of the scrubbing schedule on the LCM interval causes three limitations. First, this mechanism may require a large memory to store the scrubbing schedule. Second, it does not allow the modifications/reconfiguration on the user tasks. For instance, at instant 5 task τ_1 is stopped and this change is not reflected on the scrubbing schedule. After the instant 5, task τ_1 continues to be scrubbed, wasting ICAP utilization and consequently power consumption. At the instant 7, task τ_3 is added to the system. This change is also not reflected on the scrubbing schedule, reducing the reliability of the system, since this task is not scrubbed. Third, variations on the execution of the sporadic task are not considered on the scrubbing execution, increasing the gap between the task execution and the corresponding scrubbing execution and consequently decreasing the system reliability.

3. SYSTEM MODEL

3.1 Task and Scrubbing Model

An FPGA device can accommodate several user functionalities (tasks), each one implemented on its own reconfigurable partitions. The functionalities implemented on these partitions can be modelled by set Γ of either periodic or sporadic tasks. Each user task $\tau_i \in \Gamma$ is characterized by five parameters $\tau_i = (C_i, T_i/mit_i, \Phi_i, \eta_i, \zeta_i)$: C_i defines the worst case execution time in hardware; T_i represents the execution period for the periodic tasks and mit_i represents the minimum inter-arrival time between two consecutive executions of the sporadic tasks; Φ_i defines the initial offset, the release time of the first instance; η_i defines the number of FPGA configuration frames used to implement the task; and finally, ζ_i represents the criticality of the task in the system (zero criticality corresponds to the lowest criticality in the system).

Associated with the user task set Γ, there is a scrubbing task set $s\Gamma$. Each scrubbing task $s\tau_i \in s\Gamma$ represents the scrubbing process of task τ_i, as described in Figure 2. Moreover, each scrubbing task $s\tau_i$ can also be mod-

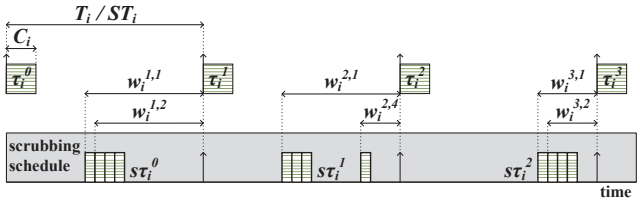

Figure 2: User task and scrubbing schedule exec.

elled as a periodic task characterized by four parameters $s\tau_i = (SC_i, ST_i, \Phi_i, \zeta_i)$: SC_i defines the time to scrub the η_i FPGA frames used to implement the task τ_i; ST_i represents the scrubbing period, which is a multiple of the corresponding task period T_i/mit_i.

In order to guarantee that no scrubbing preemptions occur during the scrubbing process of one frame, the following assumption must be taken into account during the task set definition.

Assumption 1. *The minimum time unit used to define the scrubbing schedule is equal to the time to scrub one FPGA frame. Therefore, the task periods (T_i) and consequently the corresponding scrubbing periods (ST_i) have to be multiples of that time to scrub one configuration frame.*

3.2 Error Model

The FPGA device composed by Θ configuration frames can be affected by SEUs, which follow a Poisson distribution with a rate of λ failures per unit of time [3] [5] [2]. For a given task τ_i, the ideal scrubbing instant, i.e., the instant that minimizes the probability of the k^{th} job to be executed without suffering any fault, is the interval immediately before its executions. Therefore, the probability of k^{th} job execution of the task τ_i being executed without any fault, considering the last corresponding $s\tau_i$ job execution, is given by:

$$P_{ne}[\tau_i^k] = \prod_{f=1}^{\eta_i} e^{-\frac{\lambda}{\Theta} w_i^{k,f}}, \quad (1)$$

$w_i^{k,f}$ is the time interval between last scrubbing process, in particular the frame f, and the beginning of the k^{th} job execution, as described in Figure 2. Note that the SEU rate that affects a task τ_i is proportional to the hardware resources (configuration frames) used by it. Moreover, faults during each task execution instance are not considered.

Definition 1. (Reliability of a task) *Reliability of a task is a metric that defines the probability of a task τ_i being executed in the interval $[0, t]$ without faults [7].*

Therefore, the reliability of the task τ_i can be expressed in the following equation as the probability of all instances of τ_i in the interval $[0, t]$ being executed without faults.

$$R_i(t) = P_{ne}[\tau_i^0 \wedge \tau_i^1 \wedge ... \wedge \tau_i^{k_i}], \quad (2)$$

where k_i defines the last τ_i job execution in the interval $[0, t]$.

Definition 2. (system reliability based on criticality) *System reliability based on criticality is a metric that defines the system reliability during the interval $[0, t]$ taking into account the reliability as well as the criticality of each task executing in the system.*

Therefore, the system reliability based on criticality can be expressed as follows,

$$R(t) = \sum_{i=0}^{|\Gamma|-1} R_i(t) \times \zeta_i, \quad (3)$$

where $|\Gamma|$ is the number of tasks that are executing in the system.

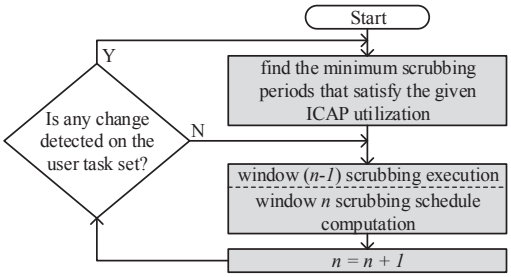

Figure 3: Proposed approach flowchart.

4. PROPOSED APPROACH

The scrubbing mechanism of an FPGA device can be classified as a soft real-time system. If some user task is not scrubbed at the right instant according to the computed scrubbing schedule, there is no negative impact on the user task execution, i.e., the user task is executed normally. However, the reliability of the system may decrease, i.e, the probability of that task to be executed with a fault may increase. Taking this factor into account, the proposed mechanism computes the scrubbing schedule in pre-defined time windows, enabling reactive adaptations of the scrubbing process to the changes on the implemented user task set. Scheduling the scrubbing tasks based on windows is not the optimal solution, since the scrubbing tasks may miss their deadlines. However, the reliability of the system is improved through a reactive update of the scrubbing mechanism to the changes and the execution of the user task set.

4.1 Window-based Scrubbing Schedule

The scrubbing schedule windows have duration Δ (defined as the multiple of the time to scrub one configuration frame) specified by the user and according to the reactivity requirements of the scrubbing process. Their execution can be interpreted as follows: the scrubbing schedule windows are executed sequentially; each window starts at the instants $n \times \Delta$ with $n = \{0, 1, 2, 3, ...\}$ and it has a duration of Δ. Therefore, the window n defines the scrubbing schedule for the interval $(n\Delta, (n+1)\Delta]$. Figure 3 describes the sequence of steps regarding the proposed approach. The first step computes the minimum scrubbing periods according to the criticality of each task (**min** $\sum_{i=0}^{s\Gamma-1} \frac{ST_i}{T_i} \times \zeta_i$) that maximize the allowed ICAP utilization defined for the scrubbing mechanism. Therefore, the computed scrubbing periods have to satisfy the following equation, $\sum_{i=0}^{s\Gamma-1} \frac{SC_i}{ST_i} \leq uBound$, where $uBound$ defines the maximum ICAP utilization provided to the scrubbing mechanism. As a result, the most critical tasks are scrubbed more frequently than the less critical ones. Then the scrubbing schedule is executed and computed in windows. During the execution of the scrubbing schedule defined in the window $n-1$, the scrubbing schedule for the window n is computed. Only the scrubbing task activations that occur in the interval $(n\Delta, (n+1)\Delta]$ will affect the schedule produced by the window n. If during the scrubbing schedule execution of the window n any change on the user task set is detected, the minimum scrubbing periods that satisfy the maximum defined ICAP utilization have to be recomputed. The new periods are used then to compute the scrubbing schedule for the incoming windows. The scrubbing schedule is computed according the fixed priority scheduling (FPS), contrary to the solution proposed by Santos *et al.* [11]. FPS is more predictable and easier to implement than dynamic priority scheduling (DPS), as well

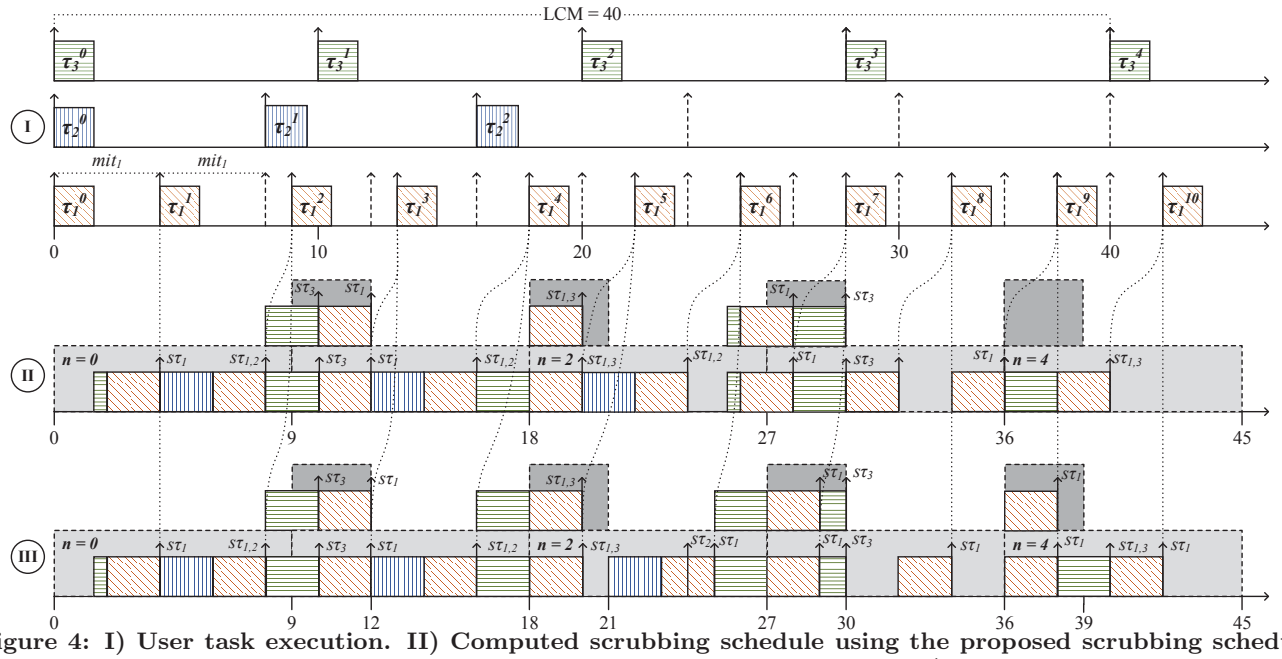

Figure 4: I) User task execution. II) Computed scrubbing schedule using the proposed scrubbing schedule based on windows and improved by using an auxiliary schedule window. III) Proposed scrubbing schedule solution improved by the feedback execution of the user sporadic tasks. Note 1: the auxiliary windows are represented by the darker boxes. The tasks inside them are the tasks considered for the scrubbing schedule of the previous window. Note 2: the user tasks execution time are not in scale with the scrubbing executions.

as the definition of criticality fits perfectly on the notion of priority. The most critical tasks are always scrubbed closer their user task executions than the less critical ones. The better schedulability of the DPS is almost irrelevant in this context, since the scrubbing mechanism should be used for low ICAP utilizations, leaving free space for FPGA reconfigurations. Note that the memory required by this approach is only the memory required to store the scrubbing schedule of two Δ windows. In contrast, the current similar approaches have to store the scrubbing schedule for the LCM interval. The gains in terms of memory would be really high when the proposed approach uses small windows sizes (Δ) and the LCM interval given by the task periods is really large.

4.1.1 Auxiliary Window – Improvement

The scrubbing schedule based on windows raises a problem. The scrubbing tasks that are activated in the beginning of the window n may not be executed since the remaining execution space in window n may not be enough. In order to solve this limitation, an auxiliary scrubbing schedule window is proposed. Every scrubbing schedule window n is succeeded by an auxiliary window with Ω size (multiple of the time to scrub one configuration frame), which helps to compute the scrubbing schedule. This way, for the produced scrubbing schedule in the window n, all the scrubbing requests in the interval $(n\Delta, (n+1)\Delta + \Omega]$ are considered. However, the produced schedule in the auxiliary window $((n+1)\Delta, (n+1)\Delta + \Omega]$ is not taken into account, since it will be recomputed in the scrubbing schedule window $(n+1)$.

4.1.2 User Task Feedback Execution – Improvement

Another improvement has to do with the sporadic tasks. The variations on the sporadic tasks execution considering their minimum inter-arrival time can be updated on the scheduled scrubbings, increasing the system reliability. During the execution of the scrubbing schedule window n, the feedback of the user task's execution is used to compute for each user task job execution $\tau_i^k \in (n\Delta, (n+1)\Delta]$ the deviation to the mit_i considering the predecessor job τ_i^{k-1}. The execution deviation of each task τ_i in the scrubbing schedule window n is given by:

$$dev_i^n = \sum_{\tau_k \in (n\Delta,(n+1)\Delta]} \tau_i^k - (\tau_i^{k-1} + mit_i). \quad (4)$$

Note that for the periodic tasks the execution deviation dev_i^n is always zero. The computed deviation of each task τ_i in the window n (dev_i^n) will be used to update the scrubbing task activations during the computation of the scrubbing schedule of the window $n+2$. Therefore, all the scrubbing task activations $s\tau_i^p$ in the window $((n+2)\Delta+\Omega, (n+3)\Delta+\Omega]$ will be delayed by the computed task execution deviation in all n windows executed before, i.e,

$$\forall s\tau_i^p \in ((n+2)\Delta + \Omega, (n+3)\Delta + \Omega],\ s\tau_i^p = s\tau_i^p + \sum_{j=0}^{n} dev_i^j. \quad (5)$$

Table 1: task set (Γ) and scrubbing task set ($s\Gamma$) parameters

τ_i	C_i[1]	T_i	SC_i	Φ_i	ζ_i	ST_i
τ_1	–	4	2	0	3	4
τ_2	–	8	2.5	0	2	8
τ_3	–	10	2	0	1	10

4.2 Scrubbing System Reactivity

Definition 3. (SCRUBBING REACTIVITY TIME) *Scrubbing reactivity time is the time that defines the interval between one change on the user design at the instant t and the instant of its effective impact on the scrubbing execution.*

[1]The faults are not considered during each task execution instance. Therefore, the user tasks execution time are omitted.

The worst case scenario occurs when the change happens in the beginning of the scrubbing schedule window. Therefore, in this case the scrubbing reactivity time is $2 \times \Delta$. On the other hand, the best case occurs when the change happens just before the beginning of one scrubbing schedule window. In this case, the scrubbing reactivity time is Δ.

4.3 Illustrative example

In order to better understand this mechanism, please consider a simple example. Three user tasks were implemented on the FPGA, where the ICAP module can be 100% used by the scrubbing mechanism. Table 1 presents the user task set and the scrubbing task set parameters. Note that task τ_1 is a sporadic task. Note also that all tasks will be scrubbed every period, since the computed periods of the scrubbing tasks are equal to the corresponding user task periods. Figure 4–I describes the user task execution. Note also that at the instant 17, task τ_2 is stopped (task jobs τ_2^3, τ_2^4 and τ_2^5 are not executed). Figure 4–II presents the proposed scrubbing solution using the auxiliary window. The scrubbing schedule window size considered in this example (Δ) is 9 *time units*. As we can see, the most critical tasks are regularly scrubbed and the change occurred on task τ_2 is reflected on the scrubbing mechanism as the scrubbing schedule window 3. The auxiliary windows ($\Omega = 3$ *time units*) are represented in the figure by the darker boxes. This mechanism improves the scrubbing schedule, since all the scrubbing task activation at the beginning of each window n that do not have enough space to execute partially or totally in the window n are executed in the window $n-1$. Figure 4–III presents the scrubbing schedule using the feedback execution of the user sporadic tasks. As we can see, the deviation from the *mit* on the execution of the task τ_1 in window 0 will be reflected on the scrubbing schedule of window 2. Similarly, the deviation occurred in window 1 is reflected on window 3. In the window 5 and 6 the scrubbing requests $s\tau_1$ are already synchronized with the task τ_1 executions.

5. EXPERIMENTAL RESULTS

Experiments were conducted in order to better evaluate the proposed scrubbing mechanism. The experiments were based on a Virtex-6 LX240T SRAM-based FPGA with 28,464 configuration frames. It was assumed that each frame is scrubbed at the maximum ICAP frequency (100MHz). Therefore, each FPGA configuration frame requires $0.81\mu s$ to be scrubbed. Moreover, for these experiments the FPGA device was simulated to be placed in the space environment, subjected to SEUs with a rate $\lambda = \frac{1}{1Hour}$[3]. The experiments use task sets synthetically generated. The period (T_i/mit_i) of each task τ_i assumes only values that are multiples of $5ms$, synthetically generated and uniformly distributed between $10ms$ and $50ms$. The offset (Φ_i) assumes values uniformly distributed between 0 and $20ms$. For sporadic user tasks, each job execution can suffer a deviation to the respective *mit* uniformly distributed between 0 and $1ms$. The number of configuration frames (η_i) used to implement each task τ_i assumes only values that are multiples of 100, synthetically generated and uniformly distributed from $1,000$ and $2,000$, corresponding to a scrubbing execution time (SC_i) between $810\mu s$ and $1,620\mu s$, respectively. Moreover, the criticality of the tasks are also synthetically generated and uniformly distributed between 1 and 100.

The proposed window-based scrubbing schedule (wss) improved by the auxiliary window (wss_{aw}) and by the user task feedback execution (wss_{aw+fe}) are evaluated and compared to the blind scrubbing mechanism [4], to the scrubbing mechanism proposed by Nazar *et al.* [9] and to the scrubbing schedule mechanism proposed by Santos *et al.* [11].

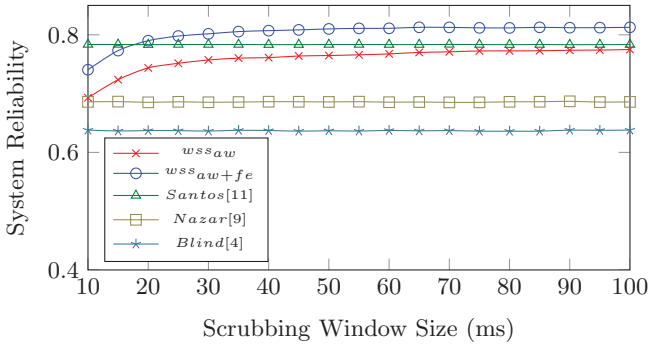

Figure 5: System reliability over the scrubbing schedule window size.

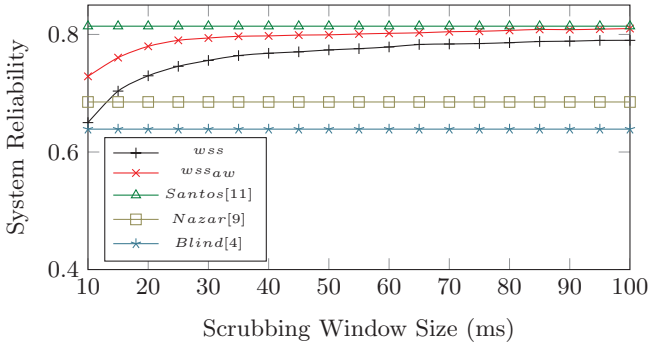

Figure 6: System reliability over the scrubbing schedule window size (user task set composed only by periodic tasks).

5.1 Reliability over the Schedule Window Size

The graph in Figure 5 shows the system reliability (given by the Equation 3) for a user task set with 20 tasks over the scrubbing schedule window size (Δ). The user task set is composed by 10 periodic tasks and 10 sporadic tasks. The window size (Δ) ranges from $10ms$ to $100ms$. The 20 tasks use 100% of the resources (configuration frames) in the FPGA and the ICAP utilization provided to the scrubbing mechanism is 100%. As expected, the reliability of the system using the proposed approach increases with the increase of the scrubbing schedule window size. The proposed solution is evaluated using the auxiliary window mechanism and the user task feedback execution mechanism. With an auxiliary window (Ω) equal to 25% of the main scrubbing schedule window, the proposed approach performs almost equal to the approach proposed by Santos *et al.* [11] for bigger Δs. Adding to the proposed solution the user task feedback execution mechanism, the system reliability increases 5% in the best case. Note that this improvement on the system reliability can be even bigger depending on the deviation (jitter) from the *mit* of the sporadic tasks activations. When the deviation increases, the performance of the solutions Santos *et al.* [11] and wss_{aw} are increasingly worse.

For a fair comparison between the proposed scrubbing mechanism and the technique proposed by Santos *et al.* [11], please considerer a user task set only composed by strict periodic tasks. The technique proposed by Santos *et al.* [11] is optimal for these scenarios, since it is based on the EDL scheduling algorithm. As the previous experiment, Figure 6 shows the system reliability for a user task set with 20 peri-

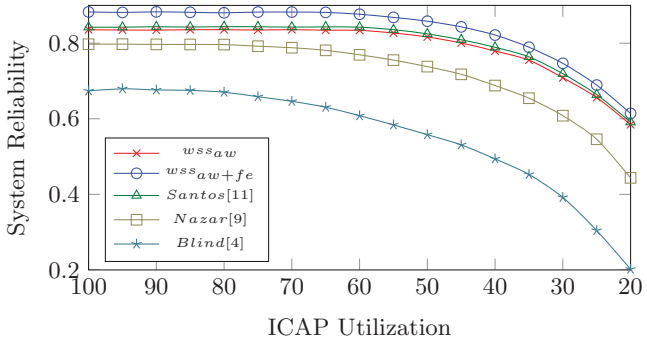

Figure 7: System reliability over the ICAP util.

odic tasks over the scrubbing schedule window size (Δ). As we can observe, the proposed scrubbing solution improved by the auxiliary window method ($\Omega = 0.25\Delta$) has almost the same performance as the solution presented in [11] for bigger Δs. Without the auxiliary window the proposed technique performs 3% worse. When compared to the other scrubbing mechanisms the proposed approach is always better. These results prove the efficiency of the proposed mechanism, even for purely periodic task sets.

5.2 Reliability over the ICAP Utilization

The graph in Figure 7 shows the system reliability for a task set with 10 tasks over the maximum ICAP utilization ($uBound$) provided to the scrubbing mechanism. The user task set is composed by 5 periodic tasks and 5 sporadic tasks. The provided ICAP utilization ranges from 20% and 100%. The scrubbing schedule window size (Δ) was defined equal to $20ms$ (very reactive). As we can see in the graph, the proposed solution using an auxiliary window (Ω), which is equal to 25% of the main window, has a very similar performance when compared to Santos *et al.* [11]. Using the user task feedback execution technique, the proposed solution achieves an improvement on the system reliability of 5% in the best case. When compared to the other approaches, the proposed solution always performs better.

5.3 Memory Overhead

This experiment compares the proposed approach to the approach proposed by Santos *et al.*[11] in terms of memory overhead required to store the scrubbing schedule produced. For a fair comparison, the memory overhead was measured considering a user task set with 20 periodic tasks and a scrubbing schedule window size (Δ) equal to $100ms$. According to the first experiment, this is the biggest window considered that almost reaches the system reliability obtained by Santos *et al.* [11]. In average the memory required by Santos *et al.* [11] to store the scrubbing schedule is 21.3 KBytes. On the other hand, the memory required by the proposed approach in this paper is 1.3 KBytes. Therefore, there is an improvement of 16 times. The improvement can be much significant, when smaller Δs are considered.

6. CONCLUSIONS

In this paper, a new scrubbing mechanism is proposed in order to dynamically adapt the scrubbing to the user task reconfigurations/modifications at runtime. This new mechanism schedules the scrubbing executed based on the windows with a pre-defined size. Conducted experiments show the feasibility of the proposed approach. They show improvements on the system reliability comparing to the other scrubbing mechanism when sporadic tasks are included in the user task set. They also show a very small decrease on the system reliability comparing to the optimal scrubbing mechanism, when the user task set is only composed by periodic tasks. In terms of memory overhead, the proposed solution performs at least 16 times better than the static approach that also schedules the scrubbing executions, considering the highest level of system reliability.

7. REFERENCES

[1] C. Argyrides, D. Pradhan, and T. Kocak. Matrix Codes for Reliable and Cost Efficient Memory Chips. *IEEE Transactions on Very Large Scale Integration Systems (VLSI'11)*, 2011.

[2] P. Axer, M. Sebastian, and R. Ernst. Reliability Analysis for MPSoCs with Mixed-critical, Hard Real-time Constraints. In *IEEE/ACM/IFIP International Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'11)*, 2011.

[3] C. C. Brendan Bridgford and C. W. Tseng. *Single-Event Upset Mitigation Selection Guide*. Xilinx Corporation. 2008.

[4] C. Carmichael, M. Caffrey, and A. Salazar. Correcting Single-Event Upsets Through Virtex Partial Configuration. Technical report, Xilinx, 2000.

[5] A. Das, A. Kumar, and B. Veeravalli. Aging-aware Hardware-software Task Partitioning for Reliable Reconfigurable Multiprocessor Systems. In *IEEE International Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES'13)*, 2013.

[6] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb. FPGA partial reconfiguration via configuration scrubbing. In *IEEE International Conference on Field Programmable Logic and Applications (FPL'09)*, 2009.

[7] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.

[8] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri, and P. Corsonello. A self-hosting configuration management system to mitigate the impact of Radiation-Induced Multi-Bit Upsets in SRAM-based FPGAs. In *IEEE International Symposium on Industrial Electronics (ISIE'10)*, 2010.

[9] G. Nazar, L. Santos, and L. Carro. Accelerated FPGA Repair Through Shifted Scrubbing. In *IEEE International Conference on Field Programmable Logic and Applications (FPL'13)*, 2013.

[10] S. P. Park, D. Lee, and K. Roy. Soft-Error-Resilient FPGAs Using Built-In 2-D Hamming Product Code. *IEEE Transactions on Very Large Scale Integration Systems (VLSI'12)*, 2012.

[11] R. Santos, S. Venkataraman, A. Das, and A. Kumar. Criticality-aware Scrubbing Mechanism for SRAM-based FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL'14)*, 2014.

[12] L. Sterpone, M. Porrmann, and J. Hagemeyer. A Novel Fault Tolerant and Runtime Reconfigurable Platform for Satellite Payload Processing. *IEEE Transactions on Computers*, 2013.

[13] S. Venkataraman, R. Santos, S. Maheshwari, and A. Kumar. Multi-Directional Error Correction Schemes for SRAM-Based FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL'14)*, 2014.