FMER: A Hybrid Configuration Memory Error Recovery Scheme for Highly Reliable FPGA SoCs

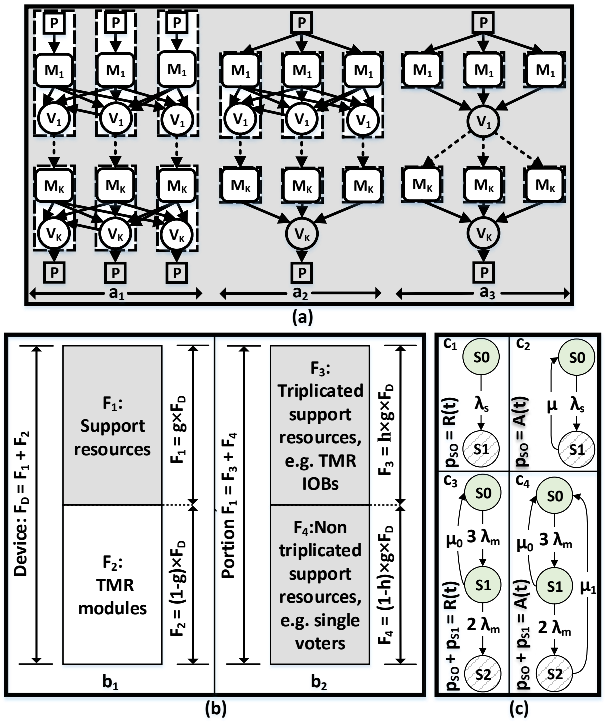
Dimitris Agiakatsikas[*], Ediz Cetin[†], and Oliver Diessel[*]
[*]School of Computer Science and Engineering, [†]School of Electrical Engineering and Telecommunications
UNSW Australia

Abstract—**High-reliability SRAM-based Field Programmable Gate Array (FPGA) designs that are deployed in space are commonly triplicated to mask Single Event Upsets (SEUs) and employ either scrubbing or modular reconfiguration to recover from radiation-induced configuration memory errors. Scrubbing benefits from vendor support and clears errors anywhere in the design but suffers from longer recovery times and higher energy use. Module-based error recovery is more energy efficient and responsive but repairs only corrupted TMR modules, leaving the supporting parts of the design such as pins or routing that are not included in the modules unrecovered. This paper proposes and assesses a hybrid technique we refer to as Frame- and Module-based Error Recovery (FMER) that uses modular reconfiguration to repair faulty TMR modules and otherwise scrubs the supporting parts of the design. We derive and compare the reliability, availability and power consumption of TMR-based System on Chip (SoC) designs that incorporate FMER, modular reconfiguration alone, blind scrubbing and no recovery. Our results reveal that FMER has the highest reliability and availability of the studied techniques in high radiation environments or when a mission's energy budget is limited.**

I. Introduction

Consider an FPGA-based System on Chip (SoC) design that is composed of a combination of some or all of the subsystems depicted in Fig. I(a), where each subsystem is composed of K Triple Modular Redundant (TMR) components. Of the subsystems depicted in Fig. I(a) the most reliable structure is that of subsystem (a_1), whose logic is completely triplicated including the voters (V) and the Input Output (I/O) pins (P). Although fully triplicated schemes provide better reliability than those with non-triplicated IO pins and voters, there are situations in which this triplication is not possible, such as when there are insufficient pins. Such a case is represented by subsystem (a_2) in which all the logic of the subsystem is triplicated except for the input pins and the output pins associated with their voters. Furthermore, as illustrated in subsystem (a_3), there are situations in which even the intermediate triplicated voters of a subsystem's components are not triplicated in order to decrease the overhead of the fully triplicated scheme. If the SoC incorporates configuration memory scrubbing [1], any erroneous resources are corrected after a scrub cycle (a complete reconfiguration of the FPGA). However, if Module-based Error Recovery (MER) [2] is applied to the SoC, only the errors inside the TMR modules are recovered, i.e. the resources inside the dashed bounding boxes of the reconfigurable regions depicted in Fig. I(a). Errors in the following resources are never recovered with MER: (i) the output pins since they are instantiated after the voters, (ii) the non-triplicated voters and their associated output pins, (iii) the non-triplicated input pins, and (iv) the routing resources that interconnect the TMR modules. We refer to all resources that are not included in the dashed shaded resources in Figs. I(a) & (b) as *"Support Resources"* (SRs). Scrubbing can therefore be considered as a slow but robust SEU error recovery technique in TMR systems since errors in both the TMR modules and the SRs are corrected by just reconfiguring the Configuration Frames (CFs) of the device. However with scrubbing, the Mean Time to Repair (MTTR) of the system is often considerably longer than with MER, which can respond rapidly when the voter detects repeated errors [3].

This research was supported in part by the Australian Research Council's Linkage (LP140100328) and Discovery (DP150103866) Projects funding schemes.


P
M₁
V₁
Mₖ
Vₖ
a₁
a₂
a₃
(a)
Device: $F_D = F_1 + F_2$
F_1: Support resources
$F_1 = g\times F_D$
F_2: TMR modules
$F_2 = (1-g)\times F_D$
b₁
Portion $F_1 = F_3 + F_4$
F_3: Triplicated support resources, e.g. TMR IOBs
$F_3 = h\times g\times F_D$
F_4: Non triplicated support resources, e.g. single voters
$F_4 = (1-h)\times g\times F_D$
b₂
(b)
c₁ c₂ c₃ c₄
S0 S1 S2
λ_s μ $3\lambda_m$ $2\lambda_m$ μ_0 μ_1
$p_{S0} = R(t)$
$p_{S0} = A(t)$
$p_{S0} + p_{S1} = R(t)$
$p_{S0} + p_{S1} = A(t)$
(c)
Fig. 1. (a) SoC subsystems, (b) SoC model and (c) R(t)&A(t) Markov models

In this paper we investigate a hybrid configuration memory error recovery mechanism that combines the advantages of both scrubbing and MER. This Frame- and Module-based Error Recovery (FMER) technique periodically scrubs the SRs until it is interrupted by a reconfiguration request from a faulty TMR module. Once the faulty TMR module is reconfigured, the periodic scrubbing resumes in order to recover any erroneous SRs. We model and compare the reliability, availability and energy consumption of four identical SoCs that incorporate (a) FMER, (b) blind scrubbing, (c) MER and (d) NR (no recovery technique). We explore the proposed SoC models at various radiation levels and design characteristics in order to determine which is the best error recovery technique to be used depending upon design/mission parameters.

II. Reliability, Availability And Energy Consumption Models

Fig. I(b) depicts an abstract model of the FPGA-based SoC design illustrated in Fig. I(a). As shown in Fig. I(b_1) the CFs of the device have been divided into two sets, $F_D = \{F_1, F_2\}$ where; (i) F_1 CFs are devoted to mapping (implementing) the SRs for the K TMR components, and (ii) F_2 CFs are devoted to mapping the logic of the $3K$ TMR modules (dashed bounding boxes in Fig. I(a)). Moreover, Fig. I(b$_2$) shows F_1 being further subdivided into two subsets, $F_1 = \{F_3, F_4\}$ where; (i) F_3 CFs are devoted to mapping the triplicated support logic, e.g. triplicated output pins and any triplicated interconnection routing resources between the TMR modules, and (ii) F_4 CFs are devoted to mapping the non-triplicated support logic. The proposed model includes two variables, $g, h \in [0, 1]$ that determine the relative sizes of the $F_{1,2,3,4}$ CF sets, i.e. $F_D = F_1 + F_2 = [g \times F_D] + [(1-g)F_D]$ and $F_1 = F_3 + F_4 = [h \times g \times F_D] + [(1-h) \times g \times F_D]$.

The average number of CFs per TMR module then is $\bar{F_M} = \frac{F_2}{3\times K} = \frac{(1-g)\times F_D}{3\times K}$. Additionally, we assume that every TMR module in the system requires; (i) $\bar{F_{TS}} = \frac{F_3}{3\times K} = \frac{h\times g\times F_D}{3\times K}$ and (ii) $\bar{F_{SS}} = \frac{F_4}{K} = \frac{(1-h)\times g\times F_D}{K}$ CFs on average for their triplicated and non-triplicated SRs respectively. We assume that the device suffers configuration memory errors at rate $\lambda_D = F_D \times B_F \times \lambda_b$, where B_F denotes the configuration bits of a CF and λ_b denotes the failure rate of one configuration bit. Consequently, the failure rate of $\bar{F_M}, \bar{F_{TS}}$ and $\bar{F_{SS}}$ is: (i) $\lambda_m = \frac{(1-g)\times\lambda_D}{3\times K} \times \mathit{UTM} \times \mathit{CBF}$, (ii) $\lambda_{TS} = \frac{h\times g\times\lambda_D}{3\times K} \times \mathit{UTS} \times \mathit{CBF}$ and *(iii)* $\lambda_{SS} = \frac{(1-h)\times g\times\lambda_D}{K} \times \mathit{UTS} \times \mathit{CBF}$. Here, *UTM* and *UTS* denote the resource utilization of the TMR modules and the SRs respectively, while CBF refers to the Criticality Bit Factor (CBF) of the TMR modules or SRs and represents the proportion of their utilized configuration bits that will lead to observable errors if they are corrupted.

The reliability R(t) and availability A(t) of an SoC, which is composed of the subsystems depicted in Fig. I(a), in which each subsystem is composed of K components can be simplified to a series logic structure of K independent components with R(t) = $\prod_{i=1}^{K} \mathrm{R}_i^{type}(t)$ and A(t) = $\prod_{i=1}^{K} \mathrm{A}_i^{type}(t)$, respectively [4]. $\mathrm{R}_i^{type}(t)$ and $\mathrm{A}_i^{type}(t)$ denote the reliability and availability functions of the $i^{th} \in [1, K]$ individual component in the SoC respectively, where *types* $\in$ {(a) Simplex & NR (no recovery), (b) Simplex & Scrub, (c) TMR & NR, (d) TMR & Scrub, (e) TMR & MER}. Therefore in SoC/FMER and SoC/MER the F_2 CFs implement *type* (e) components, while SoC/Scrub and SoC/NR implement *type* (d) and *type* (c) components respectively. Similarly, the F_1 CFs of SoC/FMER and SoC/Scrub implement *type* (b) & (d) components, while SoC/MER and SoC/NR implement *type* (a) & (c) components.

We apply the method presented in [4], which assumes that failures follow a Poisson distribution and that components fail and recover independently, to find R(t) and A(t) for all the above *types* of components with the Markov models of Fig. I(c). We assume that the initial probability distribution p_{S0} of the Markov models is $p_{S0} = 1$ and 0 elsewhere. R(t) and A(t) of type (a) & (b) components is given by the probability distribution p_{S0} of the Markov models depicted in Fig. I(c$_1$) and Fig. I(c$_2$) respectively. In this model λ_s represents the rate at which the component fails (S0:operational state→S1:failed state), while μ represents the recovery rate of the components. When scrubbed, $\mu = (0.5 \times F \times t_F + w)^{-1}$, which is the reciprocal of the MTTR in the system or the SRs, i.e. $F = F_D$ when the system is completely scrubbed (FER) and $F = F_1$ when only the SRs are scrubbed (FMER). We denote with t_F the configuration time of a CF and with w any waiting time that is inserted between the scrub cycles. When the component is not recovered, then μ = 0.

Similarly, the R(t) and A(t) models for components of *type* (c), (d) and (e) are depicted in Fig. I(c$_3$) and Fig. I(c$_4$) respectively. Their R(t) and A(t) are calculated by the probability distribution of the operational states $p_{S0}+p_{S1}$ (S0: zero faulty modules, S1: one faulty module, S2: two faulty modules). Independent of the repair mode in the component, λ_m always represents the failure rate of one TMR module. In contrast, the rates μ_0 and μ_1 in which one ($S1 \rightarrow S0$) or three ($S2 \rightarrow S0$) TMR modules recover respectively, depends on the incorporated recovery technique where for; *type* (c): TMR & NR, $\mu_0 = \mu_1 = 0$, *type* (d): TMR & Scrub $\mu_0 = \mu_1 = (0.5 \times F \times t_F + w)^{-1}$ (Scrub: $F = F_D$, FMER: $F = F_1$) and *type* (e): TMR & MER $\mu_0 = (\bar{F_M} \times t_F)^{-1}$, $\mu_1 = \frac{\mu_0}{3}$.

The energy consumption of each SoC for a mission of length *T* is estimated as follows. Denoting with E_F the energy required to reconfigure one CF in the device, then the energy consumption of SoC/Scrub is $E_{\mathit{Scrub}} = \frac{T}{F_D t_F + w} \times F_D E_F$, i.e. the number of scrub cycles during the mission multiplied by the energy consumption of a scrub cycle. Similarly, the energy consumption of SoC/MER is $E_{\mathit{MER}} = 3\lambda_m T \times \bar{F_M} E_F$, i.e. the product of the likelihood of a module error over the mission's duration and the average energy needed to recover a TMR module. Last, the energy consumption of SoC/FMER can be calculated as follows. The Reconfiguration Controller (RC) of the SoC recovers the F_2 CFs of the TMR components with MER for $T_{\mathit{MER}} = 3\lambda_m T \times \bar{F_M} t_F$ time of the mission. Thus, for the rest of the mission, $T_{\mathit{Scrub}} = T - T_{\mathit{MER}} = T(1 - 3\lambda_m \bar{F_M} t_F)$ the RC is either scrubbing the F_1 CFs of the SRs or is waiting between scrub cycles. Thus, the energy consumption of SoC/FMER for a mission time T is $E_{\mathit{FMER}} = E_{\mathit{MER}} + E_{\mathit{Scrub}} = \left(3\lambda_m T \times \bar{F_M} E_F\right) + \left(\frac{T(1-3\lambda_m \bar{F_M} t_F)}{g F_D t_F + w} \times g F_D E_F\right)$.

III. Analytical Results

This section explores and compares the reliability, availability and energy consumption of the SoCs depending upon which recovery technique is incorporated into the system. We assume that each SoC is implemented on a Xilinx Artix-7 (XC7A200TFBG484-1) FPGA which has the following specifications: F_D = 18,300 CFs, B_F = 3,232 bits and t_F = 1.01E-6 *s* considering the maximum configuration speed of the FPGA. We feel that the following initial variable setup of our model (Fig. I(b)) captures a realistic FPGA-based SoC that operates in a relatively high radiation environment; λ_b = 2.66E-10 *SEU/bit/s*, w = 0 *s*, g = 0.4, h = 1.0, *CBF* = 0.1 [5], *UTS* = 0.1, *UTM* = 0.7, K = 5 and T = 5 yrs. However, we explore all possible values of the model's variables, i.e. we vary λ_b, w, K etc. We explore our SoC model for $\lambda_b \in$ [3.76E-14, 2.66E-10], which was estimated for Xilinx 7-series series [6] from Low Earth Orbit (LEO) up to Geosynchronous Equatorial Orbit (GEO) with the CREME96 [7] Worst Week, Worst Day and Peak 5-minute models assuming 2.54 mm of Al shielding. The reliability, availability and energy consumption results are captured on the Y axis of 2D or 3D plots in Fig. 2, while the other dimensions are devoted to varying the parameters of our model. All plots use the mentioned default values unless otherwise stated. The physical units for λ_b is *SEUs/bit/s*, for w it is seconds (*s*), while the mission duration *T* is given in hours (*hrs*) or years

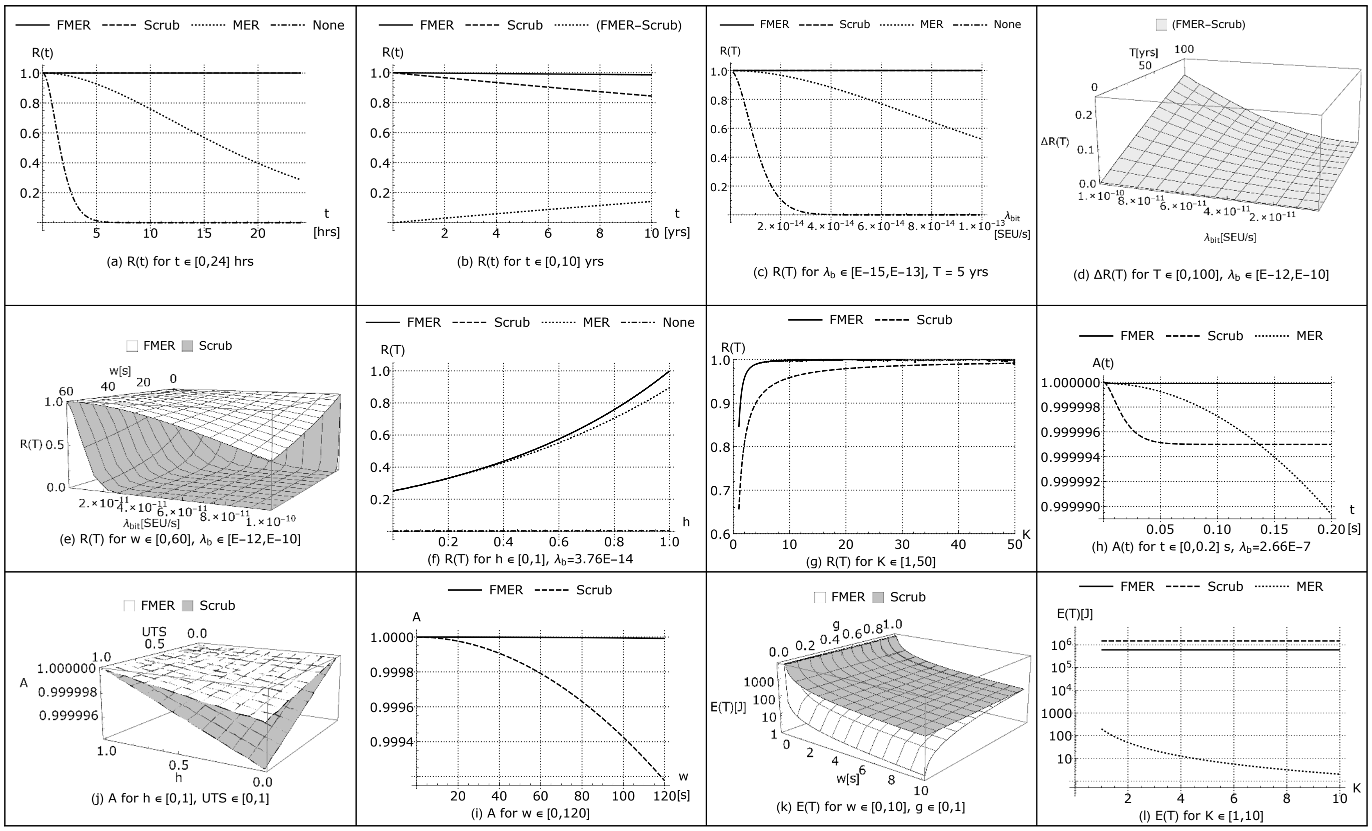


Fig. 2. Reliability, Availability and Energy Consumption Results

(*yrs*). We report energy consumption in Joules (*J*), and assume that the RC requires E_F = 535E-9 *J* to configure a CF [8].

Fig. 2(a) shows the reliability of all SoCs in the first 24 hours of a mission (T = 24 *hrs*). As can be observed, SoC/FMER and SoC/Scrub based systems are the only survivors after 24 *hrs* of operation and have negligible difference in reliability ΔR(T) $\approx$ 43E-6. However, in longer missions the reliability of SoC/FMER is interestingly higher than that of SoC/Scrub. For example, Fig. 2(b) shows the reliability of SoC/FMER and SoC/Scrub, and their difference in reliability (FMER-Scrub) for a 10 yrs mission, at the end of which SoC/FMER has R(T) $\approx$ 0.98 and SoC/Scrub has R(T) $\approx$ 0.84. On the other hand, as the failure rate λ_b of the SoC's configuration memory decreases, the difference in reliability between all SoCs becomes negligible. For example, Fig. 2(c) shows the reliability at T = 5 *yrs* and $\lambda_b \in$ [E-15, E-13], i.e. the failure rate at LEO (International Space Station (ISS) orbit at the lower end of this range). The FMER and Scrub recovery techniques result in almost the same reliability, since TMR error masking at low configuration memory failure rates allows plenty of time for recovery, i.e. μ of scrubbing is adequate to reduce the probability of having more than two faulty modules in one TMR component to a negligible level. However, due to unrecovered errors in the SRs, SoC/MER has a considerably lower reliability than SoC/FMER and SoC/Scrub over the complete λ_b spectrum, while the reliability of SoC/NR drops to 0.5 when λ_b = 9.8E-15. Additionally, the 3D plot of Fig. 2(d) shows the difference in reliability ΔR(t) between SoC/FMER and SoC/Scrub for $T \in$ [0, 100] *yrs* and $\lambda_b \in$ [E-12, E-10]. The figure shows that ΔR(t) $\to$ 0 as $\lambda_b, T \to 0$. These results indicate that FMER achieves better reliability than scrubbing does in higher radiation environment or as the mission time increases. Moreover, FMER should be considered in missions with a tight energy budget. For example, Fig. 2(e) shows the reliability of a mission with $\lambda_b \in$ [E-12, E-10] and $w \in$ [0,60] *s*. The figure reveals that the reliability of SoC/Scrub is affected more than the reliability of SoC/FMER as w increases. The reason for this is that with blind scrubbing the system is completely scrubbed, while with FMER only a portion of the device, i.e. only the F_1 CFs are scrubbed. Therefore, w can be longer in the case of FMER and still recover errors more effectively than scrubbing is able to. SoC/FMER achieves the reliability of SoC/Scrub with less frequent scrub cycles and is thus able to reduce its energy consumption.

The above results only hold when the SRs in the SoCs are fully triplicated (h = 1). As shown in Fig. 2(f), even in relatively low radiation orbits (λ_b = 3.76E-14) a small amount of non-triplicated SRs can dramatically impact the reliability of all SoCs independent of the recovery technique used. Note that the reliability of SoC/NR remains close to 0 because the TMR modules and the SRs of SoC/NR are not repaired when errors occur in them. However, the SoC designer may increase the system's reliability by partitioning the SoC at a finer granularity ($K \to \infty$), in order to increase the reliability of the system. The improvement in reliability as K increases (due to a reduced likelihood of multiple errors affecting the one component and reduced MTTR for MER) is captured in Fig. 2(g), which depicts the reliability of SoC/FMER and SoC/Scrub for $K \in$ [1,50].

Achieving high availability in SoCs is easier than achieving high reliability. The ratio between the configuration memory failure rate and error recovery rate in modern SRAM-based FPGAs makes them attractive SoC candidates for high-availability space missions. For example, Fig. 2(h) depicts the transition to steady-state availability of SoC/FMER, SoC/Scrub and SoC/MER for an extremely high failure rate, λ_b = 1,000 $\times$ 2.66E-10 (1,000x the Peak-5-Min GEO λ_b). As can be observed the steady-state availability of SoC/Scrub is 5 nines, while the steady-state availability for SoC/FMER is even higher. However, the availability of SoC/MER does not reach a steady-state since the SRs never recover from errors. Nevertheless, SoC/FMER and SoC/Scrub achieve high availability even when $h \to 0$ or $UTS \to 1$, i.e. when the SRs are not fully triplicated or are highly utilized. This is shown in Fig. 2(j) where, in the worst case, (h = 0, UTS = 1) the steady-state availability for both SoC/FMER and SoC/Scrub is approximately the same. However, FMER can be used to achieve high availability with less energy. Fig. 2(i) shows the steady-state availability of SoC/FMER and SoC/Scrub as w is varied. SoC/FMER achieves 5 nines availability when w = 120 s, while SoC/Scrub 3 nines for the same w, however $E_{FMER} \approx 5083$ J and $E_{Scrub} \approx$ 12686 J, i.e. SoC/FMER also consumes 2.5 times less energy than SoC/Scrub.

We found that SoC/FMER and SoC/Scrub energy consumption decreases geometrically as w decreases. Fig. 2(k) shows the energy consumption for both systems for $w \in$ [0,10], $g \in$ [0,1]. We observe that E_{FMER} is always less than E_{Scrub} when w is equal in both SoCs and when $g > 0$. When $g = 1$ in SoC/FMER then the system is completely scrubbed, thus $E_{FMER} = E_{Scrub}$. Additionally, as K increases, the energy consumption of SoC/MER decreases. This is true because the systems are partitioned at finer granularity which means faulty TMR components can be localized and corrected more precisely as $K \to \infty$, thus repairing less CFs per fault. This is shown in Fig. 2(l), in which we plot the energy consumption of SoC/FMER, SoC/Scrub and SoC/MER for $K \in$ [1,10] and w = 1 s. We observe that E_{MER} decreases when K in SoC/MER increases. In case of SoC/FMER the energy consumption that is spent in repairing TMR modules is negligible compared to the energy consumed scrubbing the SRs. Therefore K does not significantly affect the overall energy consumption of SoC/FMER. Furthermore, the energy consumption of SoC/Scrub is not affected at all as K varies, since μ depends on F_D and not on $\bar{F_M}$. Last, we observe that E_{MER} is less than E_{FMER} and E_{Scrub} since SoC/MER does not involve periodic scrubbing of the SRs and it reconfigures the CFs of the TMR modules only when required.

IV. Implementation of FMER

The realization of various internal or external RCs for SRAM-based FPGAs has been extensively researched [1]. Rather than detailing the implementation of an RC for FMER, we simply outline its operating principles and the bitstream analysis flow required to provide the necessary partial bitstreams for our recovery mechanism. It is assumed that the custom FMER RC has the following specifications in order to execute both scrubbing and MER: (a) it is able to reconfigure a contiguous sequence of CFs, (b) it is able to reconfigure individual CFs, and (c) it is fault-tolerant.

The most important step for the realization of the proposed FMER RC is the extraction of the "golden" F_1 and F_2 CFs from the SoC model of Fig. I(a). We propose a Dynamic Partial Reconfiguration (DPR) implementation flow [9], in which the design is partitioned into DPR and static partitions. Each TMR module of the SoC is mapped into a DPR partition, while the SRs (shaded region of Figs. I(a) & (b)) are mapped to the static area of the FPGA. The FMER RC downloads the corresponding partial bitstream of any faulty TMR module when required [3] and otherwise scrubs the F_1 CFs of the design. The vendor's DPR implementation flow guarantees that a partial bistream for each TMR module is generated and that the modules and SRs are not mapped on the same CFs. However, the CFs of the SRs can only be extracted from the design's full bitstream with the aid of bitstream manipulation tools, such as RapidSmith [10]. In more detail, the tool reads a full bitstream and creates a list with all the CF addresses of the device, while it reads and creates another $3K$ similar lists for each of the $3K$ partially reconfigurable TMR modules of the design. Next the CF addresses of the partial bitstreams are removed from the full bitstream address list in order to create a list with only the F_1 CFs (static region). Thereafter, the F_1 CF's address list is further divided into contiguous sequences of CF addresses, whereby for each such sequence a partial bitstream is generated for scrubbing the static region of the design. Each of these partial bitstreams has an initial value for the FPGA's Frame Address Register (FAR) and the corresponding CF data for the sequence.

V. Conclusions

This work has shown that FMER in TMR-based SoC FPGA designs can achieve higher reliability and availability using less energy than SoCs that are completely scrubbed. Moreover, we have shown that the energy consumption of MER is less than that of scrubbing or FMER, but it becomes ineffective during long missions or in high radiation environments since the SRs of the SoC are never recovered. FMER is under development for the "RUSH" satellite payload [11] in order to experimentally assess the correctness of our reliability, availability and power consumption models in a real world space application with fault injection experiments.

References

[1] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications," *ACM Comp. Surv.*, vol. 47, no. 2, pp. 37:1–37:34, Jan. 2015.

[2] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," in *IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems (DFT)*, Sept 2007, pp. 87–95.

[3] D. Agiakatsikas, N. T. H. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration Control Networks for TMR Systems with Module-based Recovery," in *Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 88–91.

[4] M. L. Shooman, *Reliability of computer systems and networks: fault tolerance, analysis, and design*. John Wiley & Sons, 2003.

[5] *Product Guide PG036 (v4.1)*, Xilinx Corporation.

[6] D. M. Hiemstra and V. Kirischian, "Single Event Upset Characterization of the Kintex-7 Field Programmable Gate Array Using Proton Irradiation," in *IEEE Radiat. Effects Data Workshop (REDW)*, July 2014, pp. 1–4.

[7] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Trans. on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, Dec 1997.

[8] J. Tonfat, F. Kastensmidt, and R. Reis, "Energy efficient frame-level redundancy scrubbing technique for SRAM-based FPGAs," in *NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, June 2015, pp. 1–8.

[9] *User Guide UG702 (v14.5)*, Xilinx Corporation.

[10] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, Sept 2011, pp. 349–355.

[11] E. Cetin et al., "Overview and Investigation of SEU Detection and Recovery Approaches for FPGA-Based Heterogeneous Systems," *in FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*, pp. 33–46, 2016.