FMER: An Energy-Efficient Error Recovery Methodology for SRAM-based FPGA Designs

Dimitris Agiakatsikas*, Ediz Cetin†, and Oliver Diessel*
*School of Computer Science and Engineering, UNSW Sydney, Australia
†School of Engineering, Macquarie University, Australia
{d.agiakatsikas, o.diessel}@unsw.edu.au, ediz.cetin@mq.edu.au

***Abstract*—This work introduces FMER, that is, a frame- and module-based Configuration Memory (CM) error recovery technique targeting Triple Modular Redundant (TMR) designs that are realized on SRAM-based FPGAs. Module-based CM Error Recovery (MER) is used to reconfigure on-demand the CM of faulty TMR modules, while the remaining CM of the device recovers from soft-errors with periodic scrubbing. We derive reliability, availability and power consumption models of TMR designs that incorporate FMER, MER, blind scrubbing and no recovery at all, and show that FMER is particularly beneficial for missions that require high reliability or availability subject to a low energy budget.**

***Index Terms*—Fault-tolerance, SRAM-based FPGAs, reliability, availability, radiation-induced errors, SEUs, configuration memory error recovery techniques.**

I. Introduction

High radiation environments, such as space, render the memory cells of SRAM-based FPGAs unreliable since their state can be unintentionally changed when they are struck by high energy particles [1], [2]. Such errors are usually called Single Event Upsets (SEUs) and fall into a group of radiation-induced errors, namely Single Event Effects (SEEs) [2]. However, SRAM-based FPGAs are ideal platforms for implementing complex space applications due to their low Non-Recurring Engineering (NRE) costs, their high performance and more importantly for their *in-field reconfigurability* which makes them very flexible; the hardware itself can be upgraded remotely while it operates in space without needing any physical human interaction [3]. These systems may target different or multiple mission requirements, such as high reliability, high availability or high performance [4], [5], but all systems must expend their energy reserves efficiently and strategically in order to reliably accomplish their computational tasks during the mission.

The research community is therefore increasingly developing design techniques and methodologies to enable the implementation of reliable space applications on SRAM-based FPGAs [2], [6], [7]. These techniques not only have to mitigate SEEs that are encountered in classic Application Specific Integrated Circuit (ASIC) designs, such as transient errors in combinational logic or SEUs in the user memory, i.e. Flip-Flops (FFs) or RAM, but also have to mitigate SEUs in the configuration SRAM cells of the FPGAs. Unfortunately, Configuration Memory (CM) errors[1] [8] account for most of the errors in SRAM-based FPGA systems and have a permanent effect on the functionality of the design until the CM is rewritten with the *golden* bitstream.

Most SRAM-based FPGA designs enhance their reliability with Triple Modular Redundancy (TMR) [9]. TMR triplicates a design and an additional sub-circuit, a voter, overrules any erroneous output of a corrupted module (replica). However, the TMR design fails when two or more of its replicas output different results [10]. Such a situation can arise for various reasons, i.e. Common Mode Failures (CMFs) due to a design bug that exists between all replicas [11], or due to an SEU in a shared interconnection resource between the replicas [12], [13]. Nevertheless, even when CMFs are not present in the TMR design, the possibility exists that a succession of soft errors will cause a second or a third replica to fail. In order to avoid such situations, TMR-based designs realized on SRAM-based FPGAs are usually combined with user memory [9], [14] and CM Error Recovery (ER) [15], [16], [17], [18], [19], [20] mechanisms. These mechanisms correct soft errors in the system as fast as possible in order to increase the system's availability as well as its reliability when it is triplicated.

CM ER techniques can be divided into two categories. The first category treats the CM of the FPGA as a memory component and periodically scrubs (checks and/or rewrites) its contents to correct any accumulated SEUs without taking into account the effects of these SEUs on the functionality of the design. In contrast, the second category observes the functionality of the design, which can be corrupted by either soft errors in its user memory or CM, and reconfigures (rewrites) the CM of the device when the design experiences ongoing functional errors. Further, when Module-based configuration memory Error Recovery (MER) is incorporated in the system, the voters in the TMR design not only mask errors from a faulty module but also identify which module is in error, in the minority, so as to trigger partial reconfiguration of the CM of that particular module[21], [22]. MER therefore recovers faster from CM errors than periodic scrubbing or on-demand device CM reconfiguration as it partially reconfigures a subset of the device's CM — the CM of the faulty module.

On the other hand, independent of the utilized CM ER

This work was supported by the Australian Research Council Linkage (LP140100328) and Discovery (DP150103866) Projects funding schemes and an Australian Postgraduate Award (APA) scholarship.

[1]An *error* is a manifestation of a *fault* (or otherwise of an *SEE*) on an entity and therefore it is said that the entity is experiencing a *failure*.

technique used, SEUs in the user memory should also be recovered. These recovery techniques usually involve (i) BRAM triplication with periodic scrubbing of their contents [9], and (ii) voter insertion in the feedback paths of the replicas [23] so that user memory errors are not trapped in cyclic datapaths and so that the state (flip-flops) of a replica can be re-synchronized with its healthy siblings after CM ER has been completed [22]. This work assumes that the above described user memory ER techniques are used, and that non-triplicated BRAMs or Flip-Flops (FFs) of simplex (non-triplicated) components are reset to their initial state after they recover from a CM error.

Nevertheless, despite the importance of CM ER in various applications, especially of those targeting space missions, to our best knowledge, and as stated in [20], energy consumption models for CM ER techniques have not yet been reported in the literature. Only recently, Tomfat et. al [24] proposed a methodology to measure the energy consumption of writing a configuration frame — the smallest portion of CM that can be configured at a time — in order to compare their proposed ER technique with blind scrubbing.

In this work, we provide power consumption models for CM scrubbing, MER and our proposed ER technique, FMER, in addition to the System on Chip's (SoC's) reliability and availability models, so that the trade-off between these properties can be studied and compared. This work shows that FMER is beneficial for missions where periodic device CM scrubbing or MER cannot achieve the mission's Mean Time To Recover (MTTR), availability and/or reliability requirements for a given energy budget.

A preliminary version of this work was presented in [25]. The derivation of the reliability and availability models for the simplex (non-triplicated) and TMR components — described in Sec. III — are new for this article. The proposed dependability models have also been expanded to incorporate sub-systems that are difficult to triplicate (e.g. high-speed transceivers) and are therefore usually implemented without any form of redundancy. A more thorough discussion of the assumptions made in the derivation of the dependability models, as well as a discussion of the CM SEU rate in modern SRAM-based FPGAs, is provided. The implementation of FMER is detailed, while the practicality and applicability of FMER is given through the implementation of various triplicated HLS applications on an Artix-7 200T FPGA. Finally, a related work section has been added to this manuscript.

The paper is organized as follows. Section II states the problem we address in this work. Section III provides information about the configuration architecture of modern FPGAs and its sensitivity to radiation-induced errors. Additionally, models for the dependability[2] and energy consumption of systems built on these devices is given, while section IV evaluates the proposed models. Section V provides the implementation details of FMER, while section VI evaluates FMER on a number of SoCs that implement triplicated HLS benchmarks on an Artix-7 200T FPGA. Section VII presents related work. Final considerations and future work conclude the paper.

[2]This work uses the term *dependability* to describe both the reliability and the availability of a system.

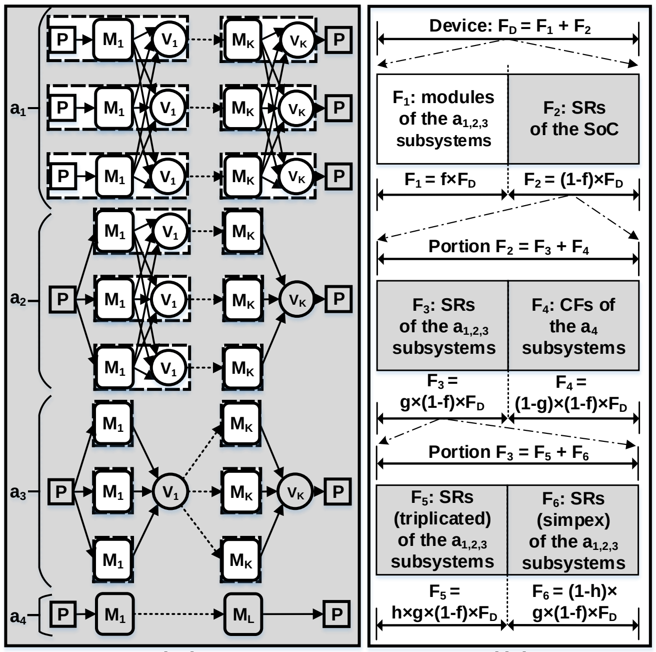
a1
a2
a3
a4
P
M1
V1
MK
VK
ML
Device: $F_D = F_1 + F_2$
F_1: modules of the $a_{1,2,3}$ subsystems
F_2: SRs of the SoC
$F_1 = f \times F_D$
$F_2 = (1-f) \times F_D$
Portion $F_2 = F_3 + F_4$
F_3: SRs of the $a_{1,2,3}$ subsystems
F_4: CFs of the a_4 subsystems
$F_3 = g \times (1-f) \times F_D$
$F_4 = (1-g) \times (1-f) \times F_D$
Portion $F_3 = F_5 + F_6$
F_5: SRs (triplicated) of the $a_{1,2,3}$ subsystems
F_6: SRs (simpex) of the $a_{1,2,3}$ subsystems
$F_5 = h \times g \times (1-f) \times F_D$
$F_6 = (1-h) \times g \times (1-f) \times F_D$
(a)
(b)

Fig. 1. (a) Possible programmable System on Chip (SoC) sub-systems, (b) FPGA-based SoC design formulation

II. Problem Statement

State-of-the-art SRAM-based FPGAs provide a large amount of configurable logic to implement a vast array of space applications on a single chip, thus reducing overall system mass, weight and power consumption. Such SoC designs are usually implemented by integrating a number of independently developed sub-systems, of which, depending upon the dependability requirements of the mission, some sub-systems are triplicated to enhance their reliability. The following paragraphs describe these SoCs in detail and point out the advantages and disadvantages of using CM scrubbing or MER for repairing faults in them. Finally, the benefits of combining these two methods, as investigated in FMER, are outlined.

Consider an FPGA-based SoC design that is composed of a combination of some or all of the a_1, a_2 a_3 and a_4 sub-systems as depicted in Fig. 1 (a), where each sub-system is composed of some different number of K or L components. Of the sub-systems depicted in Fig. 1(a) the most reliable structure is that of sub-system a_1, whose logic is completely triplicated including the voters V and the Input Output (IO) pins P [9]. Although fully triplicated schemes provide better reliability than those with simplex (non-triplicated) IO pins and voters, there are situations in which this triplication is not possible, such as when the number of available pins in the device do not suffice [26]. Such a case is represented by sub-system a_2, in which all the logic of the sub-system is triplicated except for the input pins and the output pins with their associated voters and their interconnection resources. Moreover, as illustrated in sub-system a_3, there are situations in which even the intermediate voters of a sub-system's components are not triplicated in order to decrease the cost of the fully triplicated scheme. Even worse from a reliability perspective, the SoC may include sub-systems, shown as a_4, that are not triplicated at all, due to performance issues, resource unavailability or inflexibility. Examples of such sub-systems are the hardwired high-speed transceivers or clock managers found in modern SoC FPGAs that are difficult to triplicate.

When periodic CM scrubbing [20] is incorporated in the SoC, any erroneous resource affected by an SEU in their CM will recover after a scrub cycle [9] (a complete configuration of the device's CM). On the other hand, if MER [21] is applied to the SoC, any permanent functional error of the design caused by SEUs in the FFs, BRAMs or the CM of a triplicated module will be detected by its voters and be corrected with user memory and CM ER techniques. Therefore, with MER only errors inside the TMR modules can be detected via the component's voters and thus be corrected, i.e. only erroneous resources inside the dashed (white) bounding boxes (BBs) depicted in Fig. 1 (a) recover. In this case, errors in the following resources are not detected or corrected 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, (iv) the routing resources that interconnect the modules, and (v) the simplex a_4 sub-systems. We refer to the resources that are not included in the dashed regions, but appear shaded in Fig. 1(a), as *Support Resources* (SRs). CM scrubbing therefore has higher CM fault-coverage than MER since errors in both the TMR modules and the SRs are corrected by just reconfiguring the configuration frames of the device. However, scrubbing has a considerably higher CM ER latency than MER, which responds rapidly when a voter detects repeated errors in a module [27]. Moreover, scrubbing wastes energy scanning for errors in the CM of the replicated modules, although this information is available from the component's voters.

Our contribution is a hybrid CM ER mechanism, which we refer to as Frame- and Module-based CM Error Recovery (FMER), that combines the advantages of both scrubbing and MER; FMER periodically scrubs the SRs of the SoC until it is interrupted by a reconfiguration request from a faulty replicated module, whereupon it recovers the module by MER before resuming to scrub the SRs. Therefore, FMER achieves the CM fault-coverage of scrubbing alone, but provides lower MTTR in the SoC since errors within modules are recovered immediately after they are detected and only the SRs are recovered with periodic scrubbing. Moreover, the energy expended recovering the SRs via scrubbing is less than that used were the entire device scrubbed. We model and compare the dependability and energy consumption of four identical SoCs that are composed of the sub-systems depicted in Fig. 1(a) and incorporate either (a) FMER, (b) blind scrubbing, (c) MER or (d) NR (no error recovery). Firstly, this requires the derivation of the reliability and availability functions for the components of the a_1, a_2, a_3 and a_4 sub-systems, where each component is repaired either with blind scrubbing or MER, or is left un-recovered depending on the adopted ER technique in the SoC.

We explore the proposed SoC models at various radiation levels and design parameters and show that FMER affords higher reliability and availability to SoCs with lower energy consumption than obtained with classic CM scrubbing or MER.

III. Dependability – Energy Consumption Models

This section provides background on the CM architecture of Xilinx FPGAs and their soft-error vulnerabilities in various operating orbits. It then provides models for the MTTR of CM blind scrubbing and MER, while it formulates the reliability, availability and power consumption models of the SoC, when it incorporates either blind scrubbing, MER, FMER or no recovery (NR) at all. Last, the assumptions made in the derivation of the dependability models are provided. Note that this work presents a new CM ER technique and compares the dependability of four equivalent SoCs that utilize the same user memory ER techniques. Therefore, our dependability analysis does not account for soft-errors in the user memory of the design as their effect on the dependability of each SoC is the same.

A. Configuration Memory Architecture

As already mentioned in Section I, high radiation particles in space can corrupt the CM cells of SRAM-based FPGAs. Therefore, these systems incorporate an external or internal Reconfiguration Controller (RC) that repairs soft-errors in the FPGA's CM. This subsection introduces the CM architecture of Xilinx FPGAs [28], but similar properties and models also apply to Altera SRAM-based FPGAs. In more detail, the CM of Xilinx FPGAs is accessed through blocks of memory referred to as Configuration Frames (CFs). A device includes F_D CFs that are accessed via an I_B (32-, 16- or 8-bit) Internal Configuration Access Port (ICAP) bus when the RC is internal, while other type of configuration access ports are used for the realization of external RCs. A CF is composed of B_F bits and the ideal time, t_F, needed to read or write it depends upon the operating frequency f_{ICAP} of the ICAP primitive, the I_B bus width, and the size of the frame:

$$t_F = \frac{1}{f_{ICAP}} \times \frac{B_F}{I_B} \tag{1}$$

B. Error Susceptibility Of Modern FPGAs

The upset rate of a configuration bit λ_b in Xilinx 7-series, specifically the Kintex-7 family, is given in Table I. The results were calculated using SPENVIS [29] for Geosynchronous Equatorial Orbit (GEO), Global Positioning System (GPS) orbit and Low Earth Orbit (LEO) in order to be used as upset rate references in the following sections. In deriving the figures of Table I, we used the Worst Week, Worst Day and Peak 5-minute CREME96 models [30] and assumed the presence of 2.54 mm of aluminium shielding, while the cross-section of the device was obtained from [31]. This work demonstrates FMER on a Xilinx Artix-7 (XC200T) FPGA and we believe that λ_b for this device is similar to that of Kintex-7, since all Xilinx 7-series FPGAs share a unified architecture and

TABLE I
UPSET RATE PER CONF. BIT PER SECOND, λ_b

Orbit Alt./Incl.	Worst Week	Worst Day	Peak 5-Min.
GEO 35,768 km / 0°	2.16E-11	7.34E-11	2.66E-10
GPS 20,200 km / 0°	1.43E-11	4,84E-11	1.75E-10
LEO (ISS) 400 km 51.60°	3.76E-14	1.10E-13	3.86E-13

are manufactured using TSMC's 28 nm High-K Metal Gate (HKMG) technology process [32].

For example, assuming the worst week GEO model, the CM of the Artix-7 (XC200T) FPGA is expected to upset with rate $\lambda_{device} = \lambda_b \times F_D \times B_F =$ 2.16E-11 $\times\, 18,300 \times 3,232 =$ 0.0013 SEUs/device/sec. ($\approx$ 1 SEU per 13 min). However, the upset rate of the FPGA's CM is different from the failure rate of an implemented design on that specific device. For example, a design that is implemented on this device with utilization $U = 80\%$ of the FPGA's reconfigurable resources, which has an Architectural Vulnerability Factor (*AVF*) = 15% [33] is expected to fail with rate $\lambda_{design} = \lambda_{device} \times$ U $\times$ AVF $= 0.0013 \times 0.8 \times 0.15 \approx 0.00015$ times per second ($\approx$ one failure per 1.8 hours). The AVF depends upon the design itself and denotes the portion of the utilized configuration bits that will lead to observable errors if they are corrupted. These bits that produce errors in the system are referred to as critical SEU bits in this work. Xilinx claims that the AVF is approximately 15% for an average design [33], i.e. results averaged from a number of designs that utilize more than 70% of the FPGA's available reconfiguration resources.

C. Mean-Time-To-Recover Models

This subsection provides the MTTR models of CM *blind scrubbing* and MER that are then used to derive our proposed hybrid ER technique. Note, that we combine blind scrubbing and MER for FMER, however, it is also possible to combine more sophisticated scrubbing techniques with MER to implement FMER, e.g. by combining MER with Single Error Correction Double Error Detection (SECDED) readback scrubbing.

1) Blind scrubbing: The simplest way to recover errors in an SRAM-based FPGA design is to periodically, one-by-one reconfigure the CFs of the device with *golden* CFs stored externally in a radiation-tolerant memory, e.g. flash, or radiation-hardened SRAM when high memory throughput is required [34]. The worst case time to recover from an SEU is when it occurs in the 1^{st} CF of the FPGA and the scrubber has just started to reconfigure the 2^{nd} CF of the device. Thus, the SEU will not be corrected until after a whole scrub cycle or otherwise a complete reconfiguration of the device. Hence, on average the device requires half a scrub cycle to recover from an SEU, in addition to any waiting time *w* that is inserted between the scrub cycles:

$$MTTR = \left(\frac{F_D}{2} t_F\right) + w \tag{2}$$

2) MER: Many fault-tolerant designs inherently include majority voters and/or comparators, i.e. with TMR or Dual Modular Redundant (DMR) schemes, that provide rapid error detection and localization mechanisms in the system. Therefore, the RC may rely on the system's comparators to detect repeated erroneous results from a faulty module and trigger on demand reconfiguration of the module's CFs. The MTTR of a faulty module depends on the error manifestation delay t_P to the system's RC [27], and on the time required to reconfigure the F_M CFs of the faulty module. However, t_P is neglected from the calculation since typically ($t_P \ll F_M t_F$):

$$MTTR = t_P + F_M t_F \approx F_M t_F \tag{3}$$

The reciprocal of (2) gives the rate at which SEUs recover in the system when blind scrubbing (μ_s) is applied, while the reciprocal of (3) gives the rate at which SEUs recover in a faulty module when MER (μ_m) is applied.

D. Hierarchical Dependability Models Of The SoC

By definition, the reliability function $R(t)$ represents the probability that a system has operated (according to its specifications) over the interval ($0 \leq t < T$), where $T \in \mathbb{R} \geq 0$ denotes the mission duration. In contrast, the availability function $A(t)$ is the probability of the system operating correctly at time t. When the system does not incorporate any ER mechanism then $R(t) = A(t)$. Additionally, the steady state availability A is also a useful dependability metric which estimates the probability of the system operating in the long term. Thus, $A = \lim_{t\to\infty} A(t)$.

There are many ways to model the dependability of a system. Models range from simple combinatorial models that hold only under specific assumptions, i.e. when components in the system fail and are repaired independently, to more complex models, like Markov-chain models [35], which are able to capture these dependencies. On the other hand, one can selectively use Markov-chain models for those components for which the accuracy of the dependability modelling would be negatively impacted if the failure or repair dependencies were not modelled. This work uses simple combinatorial models to capture the dependability of the SoC as a whole, while more complex Markov-chain reliability and availability models are used to capture failure and repair dependencies in each subsystem's components.

In more detail, an SoC that is composed of a number of the sub-systems a_1 - a_4 depicted in Fig. 1 (a), in which each sub-system is composed of a different number of components can be viewed as a series logical structure of components with the following combinatorial system reliability and availability functions[35]:

$$\text{R(t)} = \prod_{i=1}^{K} R_i^{type}(t) \times \prod_{j=1}^{L} R_j^{type}(t) \tag{4}$$

$$\text{A(t)} = \prod_{i=1}^{K} A_i^{type}(t) \times \prod_{j=1}^{L} A_j^{type}(t), \tag{5}$$

where variables K and L denote the total number of components that realize the $a_{1,2,3}$ (TMR-based) and a_4 (simplex-based) sub-systems respectively. Moreover, the sub-products

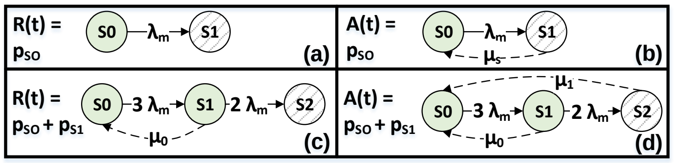

Fig. 2. Markov-chain models for the types of components encountered in sub-systems $a_1 - a_4$

of the K and L components denote the total reliability and availability of the $a_{1,2,3}$ and the a_4 sub-systems respectively. Depending on the properties of a sub-system's component, i.e. whether or not it is simplex or triplicated and whether or not it recovers with MER or CM scrubbing (when ER is applicable in the SoC), then the reliability and availability of each component in the SoC is given with one of the following *type* of function: (a) Simplex & No Recovery (NR), (b) Simplex & Scrub, (c) TMR & NR, (d) TMR & Scrub, and (e) TMR & MER. Note that Eqs. (4) & (5) just state the fact that the SoC works when all components of the $a_{1,2,3,4}$ sub-systems are functional. The following subsections provide the reliability and availability functions for the mentioned (a), (b), (c), (d) and (e) type of components, which are derived with Markov-chain models. Moreover, sub-section III-I provides the assumptions made for the derivations of these models and also discusses their accuracy and flexibility.

E. Reliability and Availability of SoC Components

In this subsection we utilize continuous-time discrete-state Markov chains to derive the dependability functions for the *types* of components encountered in the sub-systems depicted in Fig. 1(a). The derived functions are thereafter substituted into Eqs. (4) & (5) to calculate the overall *R*(*t*) and *A*(*t*) of the SoCs.

Markov chains generate a set of differential equations that are solved in order to derive the probability distribution of the chain's states, where each state represents a distinct behavior of the modeled component. We use Mathematica [36] to apply the methodologies presented in [10], [37] and [35] to find *R*(*t*) and *A*(*t*) of each component. We assume that all components start in state S0 (there are no errors in the components), i.e. that the initial probability distribution of the Markov chains depicted in Fig. 2 is $p_{S0} = 1$ and 0 elsewhere.

1) Simplex & NR: The reliability of a simplex component that has no means of recovering from errors is presented with the two states (S0: functional, S1: failed) of the Markov chain depicted in Fig. 2(a), where λ_m represents the failure rate with which the component (simplex module) transitions to the failed S1 state. The probability distribution of the functional state p_{S0} gives the reliability function of the component:

$$R^a(t) = p_{S0} = e^{-\lambda_m t}, \tag{6}$$

which verifies the well-known reliability function of a non-redundant component [35]. The availability function of the component is also given by Eq. (6), $A^a(t) = R^a(t)$, since the component is never repaired.

2) Simplex & Scrub: The reliability model of a simplex component that recovers from CM errors via scrubbing is the same as for a simplex component with no recovery at all, i.e. it can be modeled with the Markov chain depicted in Fig. 2 (a). This underscores the fact that blind scrubbing does not have any impact on the reliability of a simplex component. In the exponential reliability model, reconfiguring the component periodically does not have an impact on the occurrence of an SEU event. The reliability of a simplex component with scrubbing is therefore:

$$R^b(t) = R^a(t) = e^{-\lambda_m t}, \tag{7}$$

On the other hand, the availability of a simplex component that is periodically scrubbed is modeled with the Markov chain depicted in Fig. 2 (b). This figure augments the reliability Markov chain of the component shown in Fig.2 (a) with a transition from S1 to S0 with rate μ_s, which represents the ER rate of the component. The component is available when it is in state S0 and therefore the chain is solved for p_{S0} [37]:

$$A^b(t) = p_{S0} = \frac{\mu_s}{\lambda_m + \mu_s} + \frac{\lambda_m e^{-t(\lambda_m+\mu_s)}}{\lambda_m + \mu_s}. \tag{8}$$

The availability function of Eq. (8) is given by two partial rational expressions, whereby the second fraction denotes the transition to the steady state (first term of the equation) of the component. In this paper all *A*(*t*) functions are presented in the above form, i.e. a first term, which captures the steady state of the component, and a second term, which captures the transition to its steady state.

3) TMR & NR: The reliability model of a TMR component without any form of recovery ($\mu_0 = 0$) is illustrated in Fig. 2 (c). States S0 (no faulty modules) and S1 (one faulty module) represent the functional states of the component, while state S2 (two or more faulty modules) represents its failed state. If the three TMR modules of the component are identical, i.e. all modules have on average the same failure rate λ_m, then the chain transitions from S0 to S1 with rate $3\lambda_m$ since in S0 all modules are functional. Similarly, the chain transitions from S1 to S2 with rate $2\lambda_m$ since in S1 one module has already failed. The reliability function is given by summing the probability distribution of states S0 and S1, i.e. the functional states of the component:

$$R^c(t) = p_{S0} + p_{S1} = 3e^{-2\lambda_m t} - 2e^{-3\lambda_m t} \tag{9}$$

Additionally, since the component does not incorporate any recovery mechanism, then the availability of the component is given also by Eq. (9), i.e. $A^c(t) = R^c(t)$.

4) TMR & Scrub or TMR & MER: The reliability model of a TMR component that recovers from SEUs with scrubbing or with MER is given in Fig. 2 (c). The rate at which a module fails, λ_m, in both TMR & Scrub or TMR & MER components is the same, while the rate at which a module recovers, μ_0, depends on the adopted recovery method. As mentioned in subsection (III-C), the average time to recover an error in a module with scrubbing, (μ_s^{-1}), is a function of the device's size, given in (F_D) CFs, and the performance of the RC, i.e. the required time, (t_f), for reconfiguring a CF. In contrast, the mean time to recover a module with MER (μ_m^{-1}) depends on the RC's performance, i.e. t_f, and also on the design itself, i.e. the size of the TMR module that is given in F_M CFs. For this reason, the same reliability Markov chain can be applied to both cases by just substituting the corresponding recovery rate for scrubbing, $\mu_0 = \mu_s$, or for MER, $\mu_0 = \mu_m$, respectively. The probability distribution of S0 and S1 ($p_{S0} + p_{S1}$) gives the reliability of the components:

$$R^d(t) = R^e(t) = \frac{e^{-\frac{1}{2}(at)}\left(a\sinh\left(\frac{bt}{2}\right) + b\cosh\left(\frac{bt}{2}\right)\right)}{b}, \quad (10)$$

where $a = 5\lambda_m + \mu_0$, $b = \sqrt{\lambda_m^2 + 10\lambda_m\mu_0 + \mu_0^2}$ and $\mu_0 = \mu_s$ in the case of scrubbing, or $\mu_0 = \mu_m$ in the case of MER.

The same concept also applies for the derivation of the availability function of the TMR component in which errors recover with scrubbing or with MER. The availability model for the TMR component with scrubbing is given in Fig. 2(d), where $\mu_0 = \mu_1 = \mu_s$ since one, two or three modules of the component recover on average after half scrub cycle. The probability distribution of S0 and S1 gives its availability function:

$$\begin{aligned} A^d(t) = p_{S0} + p_{S1} &= \frac{\mu_s(5\lambda_m + \mu_s)}{ab} \\ &+ \frac{6\lambda_m e^{-bt}\left(-2\lambda_m - \mu_s + \mu_s e^{\lambda_m t} + 3\lambda_m e^{\lambda_m t}\right)}{ab}, \end{aligned} \quad (11)$$

where $a = 2\lambda_m + \mu_s$ and $b = 3\lambda_m + \mu_s$.

Similarly, the availability model for a TMR component that recovers with MER is given with the Markov chain of Fig. 2 (d), where $\mu_0 = \mu_m$ and $\mu_1 = \frac{\mu_m}{3}$ since either one or three modules need to be recovered when the component is in state S1 or S2 respectively. Its availability function is:

$$\begin{aligned} A^e(t) = p_{S0} + p_{S1} &= \frac{\mu_m(5\lambda_m + \mu_m)}{b} \\ &+ \frac{18\lambda^2 e^{-\frac{1}{6}(ct)}\left(c\sinh\left(\frac{\sqrt{a}t}{6}\right) + \sqrt{a}\cosh\left(\frac{\sqrt{a}t}{6}\right)\right)}{\sqrt{a}b}, \end{aligned} \quad (12)$$

where $a = 9{\lambda_m}^2 + 60\lambda_m\mu_m + 4{\mu_m}^2$, $b = 18{\lambda_m}^2 + 5\lambda_m\mu_m + {\mu_m}^2$ and $c = 15\lambda_m + 4\mu_m$.

Taking the limit of (6), (8), (9), (11) and (12), as $t \to \infty$, yields the steady state availability of; (a) simplex & NR, (b) simplex & Scrub, (c) TMR & NR, (d) TMR & Scrub and (e) TMR & MER components respectively:

$$A^a = \lim_{t\to\infty} R^a(t) = 0 \quad (13)$$

$$A^b = \lim_{t\to\infty} A^b(t) = \frac{\mu_s}{\lambda_s + \mu_s} \quad (14)$$

$$A^c = \lim_{t\to\infty} R^c(t) = 0 \quad (15)$$

$$A^d = \lim_{t\to\infty} A^d(t) = \frac{\mu_s(5\lambda_m + \mu_s)}{6{\lambda_m}^2 + 5\lambda_m\mu_s + {\mu_s}^2} \quad (16)$$

$$A^e = \lim_{t\to\infty} A^e(t) = \frac{\mu_m(5\lambda_m + \mu_m)}{18{\lambda_m}^2 + 5\lambda_m\mu_m + {\mu_m}^2} \quad (17)$$

F. FPGA-based SoC Design Formulation

Fig. 1(b) depicts an abstract model of an FPGA-based SoC composed of a number of the sub-systems of type a_1 – a_4 illustrated in Fig. 1(a). As shown at the top of Fig. 1(b), the FPGA's CFs have been divided into two subsets, $F_D = \{F_1, F_2\}$ where:

- $F_1 = F_D - F_2$ CFs are devoted to mapping (implementing) the logic of the *3K* TMR modules (dashed boxes in Fig. 1(a)) of the TMR-based sub-systems a_1 - a_3 that can either be recovered by MER or by scrubbing, or not at all.
- $F_2 = F_D - F_1$ CFs are devoted to mapping the SRs of the SoC, i.e. the CFs of the shaded area in Fig. 1(a) that recover with scrubbing when FMER is employed or when scrubbing is used to recover from CM errors in the SoC.

Moreover, Fig. 1(b) shows F_2 being further subdivided into two subsets, $F_2 = \{F_3, F_4\}$ in the middle of the figure where:

- $F_3 = F_1 - F_4$ CFs are devoted to mapping the SRs for the $a_{1,2,3}$ sub-systems, e.g. their simplex or their triplicated IO pins, voters and interconnection resources.
- $F_4 = F_1 - F_3$ CFs are devoted to mapping the logic for the simplex a_4 sub-systems, e.g. clock managers or complex high-speed transceivers with their IO pins and interconnection resources.

Last, F_3 is further subdivided at the bottom of the figure into two subsets, $F_3 = \{F_5, F_6\}$ where:

- $F_5 = F_3 - F_6$ CFs are devoted to mapping the triplicated SRs of the $a_{1,2,3}$ sub-systems, e.g. triplicated output pins and any triplicated interconnection routing resources between the TMR sub-systems.
- $F_6 = F_3 - F_5$ CFs are devoted to mapping the simplex SRs of the $a_{1,2,3}$ sub-systems, e.g. simplex IO pins or simplex voters and the interconnection resources between them.

The proposed model includes three parameters, $f, g, h \in [0, 1]$ in order to distinguish between F_1 and F_2, or F_3 and F_4, or F_5 and F_6 respectively, i.e.
$F_D = F_1 + F_2 = [f \times F_D] + [(1 - f) \times F_D]$,
$F_2 = F_3 + F_4 = [g \times (1 - f) \times F_D] + [(1 - g) \times (1 - f) \times F_D]$,
and $F_3 = F_5 + F_6 = [h \times g \times (1 - f) \times F_D] + [(1 - h) \times g \times (1 - f) \times F_D]$.

Therefore, the average number of CFs for one TMR module is given by:

$$\overline{F_M} = \frac{F_1}{3 \times K} = \frac{f \times F_D}{3 \times K}, \quad (18)$$

i.e. every TMR component occupies F_1/K CFs on average, whereas each of its three identical modules occupies $(F_1/K)/3$ CFs.

Additionally, it is assumed that every TMR component in the $a_{1,2,3}$ sub-systems require

$$\overline{F_{SS}} = \frac{F_6}{K} = \frac{(1 - h) \times g \times (1 - f) \times F_D}{K} \quad (19)$$

CFs for the implementation of their simplex SRs, while every module in the TMR components require

$$\overline{F_{TS}} = \frac{F_5}{3 \times K} = \frac{h \times g \times (1 - f) \times F_D}{3 \times K} \quad (20)$$

CFs for the implementation of their triplicated SRs.

The remaining CFs of the device, i.e. the F_4 portion implement the simplex modules of the a_4 sub-systems, whereby each module with its interconnections and IO pins occupies:

$$\overline{F_{SysSRs}} = \frac{F_4}{L} = \frac{(1-g) \times (1-f) \times F_D}{L} \tag{21}$$

CFs on average.

Moreover, the total upset rate of the device is modeled as follows:

$$\lambda_D = F_D \times B_F \times \lambda_b \tag{22}$$

Consequently, the average failure rate for the logic implemented with the $\overline{F_M}$, $\overline{F_{SS}}$, $\overline{F_{TS}}$ and $\overline{F_{SysSRs}}$ CFs are:

$$\overline{\lambda_m} = \frac{f \times \lambda_D}{3 \times K} \times U_M \times AVF, \tag{23}$$

$$\overline{\lambda_{SS}} = \frac{(1-h) \times g \times (1-f) \times \lambda_D}{K} \times U_S \times AVF, \tag{24}$$

$$\overline{\lambda_{TS}} = \frac{h \times g \times (1-f) \times \lambda_D}{3 \times K} \times U_S \times AVF, \tag{25}$$

$$\overline{\lambda_{SysSRs}} = \frac{(1-g) \times (1-f) \times \lambda_D}{L} \times U_C \times AVF, \tag{26}$$

respectively. Moreover, the U_M and U_S variables denote the resource utilization of each TMR module and their SRs in the $a_{1,2,3}$ sub-systems respectively, while U_C denotes the resource utilization in the a_4 sub-system, i.e. its modules with their associated IO pins and interconnections.

G. Recovery technique: Impact on SoC reliability and Availability

This subsection formulates the reliability and availability of four SoC implementations: 1) SoC/FMER, 2) SoC/Scrub, 3) SoC/MER and 4) SoC/NR (No Recovery), which follow the model of Fig. 1(b).

1) SoC/FMER – Errors in F_1 (TMR modules) are recovered with MER, while in F_2 (SRs) with scrubbing.
2) SoC/Scrub – Both F_1 and F_2 portions are scrubbed or in other words the device is completely scrubbed.
3) SoC/MER – Errors in F_1 are recovered with MER, while in F_2 errors are not recovered.
4) SoC/NR – The device does not incorporate any form of CM ER.

The difference between the above SoC implementations is the incorporated recovery technique and therefore the rate at which their components recover. The average rate at which the modules and the associated SRs fail is the same irrespective of the repair mode. The following provides the *R*(*t*) and *A*(*t*) for the SoC/FMER model.

1) SoC/FMER: The rate μ_m at which the average TMR module recovers with MER in F_1 can be found by substituting Eq. (18) into the reciprocal of Eq. (3):

$$\mu_m = (\overline{F_M} \times t_F)^{-1} \tag{27}$$

Moreover, the rate μ_s at which the F_2 CFs of the SRs recover with scrubbing can be found by replacing F_D with F_2 in the reciprocal of Eq. (2):

$$\mu_s = (\frac{F_2}{2} \times t_F + w)^{-1} \tag{28}$$

TABLE II
RELIABILITY AND AVAILABILITY TERMS OF ALL SOCS

SoC/FMER	SoC/Scrub	SoC/MER	SoC/NR
$R_i^e(t)\|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$	$R_i^d(t)\|_{(23)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$R_i^e(t)\|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$	$R_i^c(t)\|_{(23)\Rightarrow\lambda_m}$
$R_i^d(t)\|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$R_i^d(t)\|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$R_i^c(t)\|_{(25)\Rightarrow\lambda_m}$	$R_i^c(t)\|_{(25)\Rightarrow\lambda_m}$
$R_i^b(t)\|_{(24)\Rightarrow\lambda_s}$	$R_i^b(t)\|_{(24)\Rightarrow\lambda_s}$	$R_i^a(t)\|_{(24)\Rightarrow\lambda_s}$	$R_i^a(t)\|_{(24)\Rightarrow\lambda_s}$
$R_j^b(t)\|_{(26)\Rightarrow\lambda_s}$	$R_j^b(t)\|_{(26)\Rightarrow\lambda_s}$	$R_j^a(t)\|_{(26)\Rightarrow\lambda_s}$	$R_j^a(t)\|_{(26)\Rightarrow\lambda_s}$
$A_i^e(t)\|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$	$A_i^d(t)\|_{(23)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$A_i^e(t)\|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$	$A_i^c(t)\|_{(23)\Rightarrow\lambda_m}$
$A_i^d(t)\|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$A_i^d(t)\|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$	$A_i^c(t)\|_{(25)\Rightarrow\lambda_m}$	$A_i^c(t)\|_{(25)\Rightarrow\lambda_m}$
$A_i^b(t)\|_{(24)\Rightarrow\lambda_s}^{(28)\Rightarrow\mu_s}$	$A_i^b(t)\|_{(24)\Rightarrow\lambda_s}^{(28)\Rightarrow\mu_s}$	$A_i^a(t)\|_{(24)\Rightarrow\lambda_s}$	$A_i^a(t)\|_{(24)\Rightarrow\lambda_s}$
$A_j^b(t)\|_{(26)\Rightarrow\lambda_s}^{(28)\Rightarrow\mu_s}$	$A_j^b(t)\|_{(26)\Rightarrow\lambda_s}^{(28)\Rightarrow\mu_s}$	$A_j^a(t)\|_{(26)\Rightarrow\lambda_s}$	$A_j^a(t)\|_{(26)\Rightarrow\lambda_s}$

Therefore, the reliability of the SoC follows from (4) and is given by the product of two sub-products:

$$\text{R(t)} = \prod_{i=1}^{K} R_i^e(t) R_i^d(t) R_i^b(t) \times \prod_{j=1}^{L} R_j^b(t), \tag{29}$$

whereby the first sub-product denotes the reliability of the $a_{1,2,3}$ sub-systems, while the reliability function of each of their K components depends on their type and is given as follows:

- The reliability function of the TMR modules of the i^{th} TMR component is given by $R_i^e(t)|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$ since they are recovered with MER, while
- the reliability of the i^{th} triplicated SRs associated with the i^{th} TMR component is given by $R_i^d(t)|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$ since they are recovered by scrubbing,
- and that of the i^{th} simplex SRs associated with the i^{th} TMR component is given by $R_i^b(t)|_{(24)\Rightarrow\lambda_m}$ since they recover with scrubbing.

The second sub-product of Eq. (29) denotes the R(t) of the a_4 sub-systems where the reliability of its j^{th} simplex module with its associated pins and interconnection is $R_j^b(t)|_{(26)\Rightarrow\lambda_m}$ as it is scrubbed.

Similarly, the availability of the SoC is calculated as

$$\text{A(t)} = \prod_{i=1}^{K} A_i^e(t) A_i^d(t) A_i^b(t) \times \prod_{j=1}^{L} A_j^b(t), \tag{30}$$

whereby $A_i^e(t)|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$, $A_i^d(t)|_{(25)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$ and $A_i^b(t)|_{(24)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$ are substituted in the first sub-product of the equation, and $A_j^b(t)|_{(26)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$ is substituted in the second sub-product of the equation.

The reliability and availability of the other implementations, namely SoC/Scrub, SoC/MER and SoC/NR are similarly derived and are summarized in Table II. The first four rows in the first column (SoC/FMER) of the table correspond to the $R_i^e(t), R_i^d(t), R_i^b(t)$, and $R_j^b(t)$ terms of Eq. (29) respectively, while the last four rows in the same column correspond to the $A_i^e(t), A_i^d(t), A_i^b(t)$, and $A_j^b(t)$ terms of Eq. (30) respectively. Similarly, the corresponding terms for SoC/Scrub, SoC/MER and SoC/NR are given in the rows of the second, the third and the fourth columns respectively. For example, the

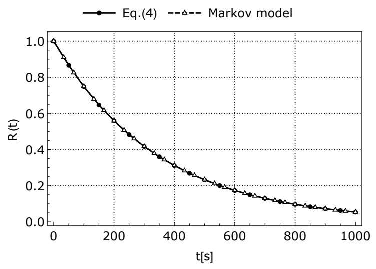

Fig. 3. Comparison of R(t) for t ∈ [0,1000] s between Eq. (4) and the Markov model of [10]

$R_i^e(t)|_{(23)\Rightarrow\lambda_m}^{(27)\Rightarrow\mu_m}$ term of SoC/FMER (shown in the 1st column, 2nd row of Table II) becomes $R_i^d|_{(23)\Rightarrow\lambda_m}^{(28)\Rightarrow\mu_s}$ for the case of SoC/Scrub (shown in the 2ndcolumn, 2nd row of Table II).

H. Recovery technique: Impact on SoC energy consumption

This subsection estimates the energy consumption of each SoC implementation, depending upon which ER technique is used and the mission's length *(T)*.

Denoting with E_F the energy required to reconfigure one CF in the device, then the energy consumption of SoC/Scrub is:

$$E_{Scrub} = \frac{T}{F_D t_F + w} \quad F_D E_F, \tag{31}$$

i.e. the first term, $(\frac{T}{F_D t_F + w})$, denotes the number of executed scrub cycles during the mission and the second term, $(F_D E_F)$, denotes the energy consumption of a scrub cycle.

Similarly, the energy consumption of SoC/MER is:

$$E_{MER} = 3\overline{\lambda_m}T \quad \overline{F_M}E_f, \tag{32}$$

i.e. the first term, $(3\overline{\lambda_m}T)$, represents the expected number of TMR module failures during the mission, while the energy needed to recover a faulty TMR module is given by the second term of the equation, $(\overline{F_M}E_f)$.

The energy consumption of SoC/FMER can be calculated as follows. The RC of the SoC recovers the F_1 CFs (that implement the TMR modules) with MER for $T_{MER} = 3\overline{\lambda_m}T \ \overline{F_M}t_F$ time of the mission. Thus, for the rest of the mission, $T_{Scrub} = T - T_{MER} = T(1 - 3\overline{\lambda_m} \ \overline{F_M}t_F)$ the RC is either scrubbing the F_2 CFs of the SRs or is waiting between scrub cycles. Thus, the energy consumption of SoC/FMER for a mission time T is:

$$\begin{aligned} E_{FMER} &= E_{MER} + E_{Scrub} \\ &= 3\overline{\lambda_m}T \ \ \overline{F_M}E_F + \frac{T(1 - 3\overline{\lambda_m} \ \ \overline{F_M}t_F)}{(1-f)F_D t_F + w} \ \ (1-f)F_D E_F, \end{aligned} \tag{33}$$

where $(1-f)F_D$ denotes the F_2 portion of the device's CFs that is scrubbed periodically.

I. Assumptions

The reliability and availability functions in this section were derived given the following assumptions. Failures in the SoC follow a Poison distribution, while Eqns. (4) and (5) hold if all K+L components fail and recover independently in the SoC. The assumption of independent failures between the SoC's components holds when appropriate design techniques are followed during the implementation of the system. For example, the authors in [13] developed CAD tools that place and route a TMR-based design in a Xilinx FPGA in a way that eliminates the probability of having more than one TMR module failure from a single CM upset. Similar commercial CAD tools that isolate failures between the modules of a SoC are provided in the Xilinx Isolation Design Flow [38].

Additionally, the assumption of independent recovery between the SoC's components holds when the system is completely scrubbed, since a complete reconfiguration of the FPGA is equivalent to having a dedicated repair facility for each TMR module or SR in the SoC, which requires μ_s^{-1} time to recover it. Therefore, the recovery process between all components in the SoC/Scrub is independent. For instance, the authors in [10] used a Markov-chain model, rather than Eq.(4) to calculate the reliability of a TMR-based design. We have found that Eq.(4) yields the same results as the Markov model given in [10] despite being considerably more straightforward to evaluate. This can be seen in Fig. 3, where we plot the reliability of the system for $K = 20$, $\lambda_{\text{device}} = 0.1$ SEUs/dev/s ($\lambda_m = \lambda_{\text{device}}/3K$) and recovery rate $\mu_s = 10\lambda_{\text{device}}$, by using Eq.(4) and the Markov model of [10]. On the other hand, when the SoC incorporates MER an independent reconfiguration process (recovery) for every faulty module holds as long as the repair rate of the module is much larger than its failure rate, $\mu_m \gg \lambda_m$, i.e. when the probability of executing a recovery process in a faulty TMR component, while another TMR component requires repair is negligible [37]. In practice, this is usually the case, since the rate at which modules fail and recover in a TMR-based SoC that incorporates MER is on the order of *hours* and *ms* respectively [27].

IV. Analytical Results

This section explores and compares the reliability, availability and energy consumption of the four SoC implementations presented in Section III, namely SoC/FMER, SoC/Scrub, SoC/MER and SoC/NR. We recall and summarize the most relevant parameters that are frequently used in this section:

- $K \in \mathbb{N}^+$: number of TMR components in the SoC.
- $f \in [0, 1]$: the fraction of device CFs devoted to the $3K$ TMR modules. All CFs are devoted to the SRs when f = 0.
- $w \in \mathbb{R}^+ \cup \{0\}$: waiting time between scrub cycles given in seconds (s).
- $\lambda_b \in \mathbb{R}^+$: upset rate of a configuration bit given in SEUs/bit/s.
- $T \in \mathbb{R}^+$ denotes the mission duration given in hours (hrs) or years (yrs).

We assume that each SoC is implemented on a Xilinx Artix-7 200T 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 base parameters of the model depicted in Fig. 1(b) captures a realistic FPGA-based SoC design

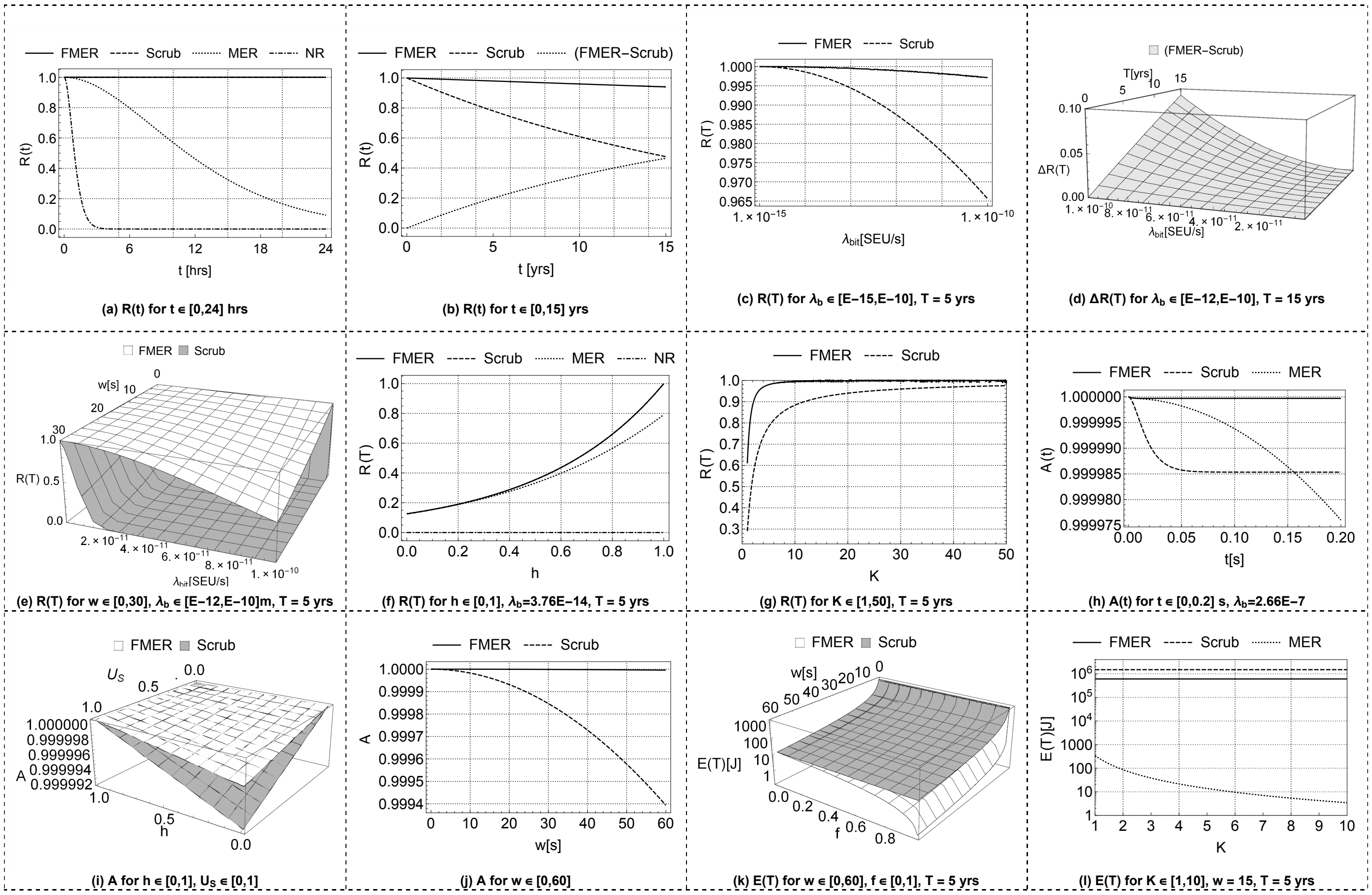

Fig. 4. Reliability, Availability and Energy Consumption Results

that operates in a relatively high radiation environment; λ_b = 2.66E-11 *SEU/bit/s*, w = 0 *s*, f = 0.6, g = 1.0, h = 1.0, *AVF* = 0.15, U_S = 0.1, U_M = U_C = 0.8, K = 5 and T = 5 yrs. However, we explore all possible values of the model's parameters, i.e. we vary λ_b, w, K, etc. We explore the SoC's model from LEO up to GEO radiation levels, $\lambda_b \in$ [3.76E-14, 2.66E-10], which were estimated in Subsection III-B.

The reliability, availability and energy consumption results are captured on the y-axis of 2D and 3D plots in Fig. 4, while the other dimensions are devoted to the model parameters. All plots use the mentioned default values unless otherwise stated. We report energy consumption in Joules (J) and assume that the RC requires on average E_f = 535E-9 J to reconfigure a CF [24]. Although, the authors of [24] conducted their measurements on a Xilinx Virtex-5 FPGA rather than on the utilized Artix-7 FPGA of this work, the assumed E_f does not change the shape of the provided energy results.

A. Reliability Results

Fig. 4(a) shows the reliability of all SoCs for a 24-hour mission (T = 24 *hrs*); as expected SoC/MER is considerably less reliable than SoC/FMER and SoC/Scrub because errors in the SRs are not recovered, while SoC/NR is the least reliable of all implementations since no ER technique is incorporated into the system. On the other hand, SoC/FMER and SoC/Scrub are the only survivors after 24 *hrs* of operation and have negligible difference in reliability ΔR(T) ≈ 43E-6. However, for a longer mission the reliability of SoC/FMER is interestingly higher than that of SoC/Scrub. For example, Fig. 4(b) shows the reliability of SoC/FMER and SoC/Scrub, and their difference in reliability (FMER-Scrub) for a 15-year mission, at the end of which SoC/FMER has R(15 yrs) ≈ 0.94, much higher than SoC/Scrub, which has R(15 yrs) ≈ 0.47.

Nevertheless, when the upset rate λ_b of the SoC's CM decreases, the reliability of SoC/Scrub becomes similar to that of SoC/FMER. For example, Fig. 4(c) shows the reliability at T = 5 yrs and $\lambda_b \in$ [E-15, E-10]. When the CM upset rate is low, SoC/FMER and SoC/Scrub both achieve high reliability since TMR error masking at low CM upset rates allows plenty of time for recovery, i.e. μ of scrubbing is adequate to reduce the probability of having more than one faulty module per TMR component to a negligible level. This can be observed in Fig. 4(c) when $\lambda_b \in$ [E-15, E-13] — the reliability of SoC/FMER and SoC/Scrub at the International Space Station (ISS) orbit at the lower end of this range.

Additionally, the 3D plot of Fig. 4(d) shows the difference in reliability ΔR(T) between SoC/FMER and SoC/Scrub for $T \in$ [1, 15] *yrs* and $\lambda_b \in$ [E-12, E-10]. The figure shows that ΔR(T) $\rightarrow$ 0 as $\lambda_b \rightarrow 0$. These results indicate that SoC/FMER achieves substantially better reliability than SoC/Scub, particularly in higher radiation environments or as the mission time increases. Moreover, FMER should be considered in missions with a tight energy budget, i.e. w in SoC/FMER can be increased to a level so that it achieves the same reliability as SoC/Scrub does, but by consuming less energy. For example, Fig. 4(e) shows the reliability of a mission with $\lambda_b \in$ [E-12, E-10] and $w \in$ [0,30] 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 alone the system is completely scrubbed, while with FMER only a portion of the device is scrubbed, i.e. only the F_2 CFs are scrubbed. Therefore, w can be longer in the case of FMER and still recover errors more effectively than scrubbing the whole device. Moreover, SoC/FMER achieves the reliability of SoC/Scrub with less frequent scrub cycles and is thus able to reduce its energy consumption. For instance, in Fig. 4(e) we observed that both SoC/FMER and SoC/Scrub have R(5 yrs) $\approx$ 0.992 when λ_b = E-11 and when w is 30 s for SoC/FMER and 0.198 s for SoC/Scrub respectively. Using Eqs. (31) and (33) it is calculated that SoC/Scrub consumes 347 times more energy than SoC/FMER ($E_{FMER} \approx$ 20,297 J, $E_{Scrub} \approx$ 7.03E6 J) during the mission, in order to achieve the same reliability as SoC/FMER does.

However, the above results only hold when the SoC is fully triplicated, i.e. when g, h = 1. As shown in Fig. 4(f), the reliability of all SoCs is dramatically reduced, even in low radiation orbits (λ_b = 3.76E-14), when the proportion of simplex components increases in the $a_{1,2,3}$ sub-systems, i.e. when $h \rightarrow 0$. Note that similar results are observed when simplex a_4 sub-systems are included in the SoC, i.e. when $g \rightarrow 0$. In more detail, the reliability of the SoC is reduced due to the unreliability of any included simplex resources. For example, assume a fully triplicated SoC that has R_{SoC}(1 yr) = 0.9999. If an additional simplex component with R_{Simplex}(1 yr) = 0.5 is included in the design, then the overall reliability of the SoC will be R_{SoC}(1 yr) = 0.9999 $\times$ 0.5 $\approx$ 0.5 $\approx$ R_{Simplex}(1 yr).

Nevertheless, when the SoC is fully triplicated, i.e. when $g, h = 1$, then the reliability of the system can be increased by partitioning the design at a finer granularity ($K \rightarrow \infty$), e.g. by triplicating every stage of a processor rather than the processor as a whole. Firstly, as K increases, the number of TMR modules in the system increase and the likelihood of multiple errors affecting the one component is reduced. Moreover, the average number of CFs per TMR module decreases, and thus the overall MTTR in SoC/FMER (not in SoC/Scrub) decreases since less CFs have to be reconfigured per faulty TMR module with MER according to Eq. (3). The improvement in reliability as K increases is captured in Fig. 4(g), which depicts the reliability of SoC/FMER and SoC/Scrub for $K \in$ [1,50]. As can be observed the reliability of SoC/FMER improves faster than SoC/Scrub as K increases.

B. Availability Results

Achieving high availability in an FPGA-based SoC is easier than achieving high reliability. The ratio between the CM upset rate and ER rate in modern SRAM-based FPGAs makes them attractive SoC candidates for high-availability space missions [3].

For example, Fig. 4(h) depicts the transition to steady state availability of SoC/FMER, SoC/Scrub and SoC/MER for an extremely high CM upset rate, λ_b = 1,000 $\times$ 2.66E-10 (1,000x the Peak-5-Min GEO λ_b), which is more likely to be encountered in high-energy physics experiments [39] than in space. As can be observed, even with this extremely high λ_b the steady state availability of SoC/Scrub is 4 nines, while the steady state availability for SoC/FMER is much higher. In contrast, the availability of SoC/MER does not reach a steady state since the SRs never recover when MER alone is applied to the SoC.

Nevertheless, SoC/FMER and SoC/Scrub achieve high availability even when $h \rightarrow 0$ and U_S = 1, i.e. when the SRs are not fully triplicated and are highly utilized. This is shown in Fig. 4(i) where in the worst case (h = 0, U_S = 1), the steady state availability of SoC/FMER and SoC/Scrub is 0.999997 and 0.999991 respectively. However, FMER can be used to achieve high availability with less energy consumption than when scrubbing alone is used. Fig. 4(j) shows the steady state availability of SoC/FMER and SoC/Scrub as w is varied. SoC/FMER achieves 5 nines availability when w = 60 s, while SoC/Scrub approximately 3 nines for the same w, however $E_{FMER} \approx$ 10,163 J, while $E_{Scrub} \approx$ 25,369 J, i.e. SoC/FMER achieves higher availability than SoC/Scrub with 2.5 times less energy. Note that similar results are obtained when the SoC incorporates simplex a_4 sub-systems, i.e. when $g \rightarrow 0$.

C. Energy Consumption Results

We found that SoC/FMER and SoC/Scrub energy consumption decreases geometrically as w increases. Fig. 4(k) illustrates the energy consumption for both systems in logarithmic scale for $w \in$ [0,60], $f \in$ [0,1]. We observe that E_{FMER} is always less than E_{Scrub} for equal values of w and for $f >$ 0. When $f = 0$ 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 system is partitioned at finer granularity which means faulty TMR modules can be localized and corrected more precisely as K increases, thus the RC reconfigures less CFs per fault. This is shown in Fig. 4(l), in which we plot the energy consumption of SoC/FMER, SoC/Scrub and SoC/MER against a logarithmic energy scale for $K \in$ [1,10] and w = 1 s. We observe that E_{MER} decreases as K in SoC/MER increases. In the case of SoC/FMER the energy consumption that is expended 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 $\overline{F_M}$. Last, we observe that E_{MER} is less than E_{FMER} and E_{Scrub} since SoC/MER does not involve periodic scrubbing as it only reconfigures the CFs of faulty TMR modules when error are detected.

V. Implementation of FMER

FMER can be implemented with any internal or external RC that is able to reliably configure the CM of the FPGA with CFs located in an external memory. In principle, FMER can be realized by: (i) constraining the placement of modules to specific regions of the FPGA, which are referred to as Pblocks in this work, and (ii) generating lists of frame addresses (FADs) for each Pblock and for the SRs of the design, so that the CFs of the SR FAD list are periodically scrubbed, and the CFs of any corrupted Pblock are reconfigured on-demand. In order to generate the lists of FADs for the Pblocks and SRs we execute the following steps: (i) generate a FAD list for the whole device, (ii) generate a FAD list for each Pblock, and (iii) create a FAD list for the SRs by subtracting the Pblock FAD lists from the device FAD list.

A. Generating FAD lists and CFDATA

To the best of our knowledge, two methodologies have been described in the literature to create a FAD list for a Xilinx device. One way is to extract the FAD from the bitstream with custom bitstream manipulation tools [17], [40], which can be implemented using academic CAD frameworks such as Rapidsmith [41]. However, to use this method, the bitstream has to have a format in which the FAD for each CF in the bitstream can be associated with its configuration data (CFDATA). This bitstream format can be obtained from the Xilinx Vivado design suite by enabling the CRC-per-frame flag or the debug flag during its generation. Another way to generate the FAD list for the device is to readback the CM of the FPGA and capture the Frame Address Register (FAR) as it auto-increments [42]. We used the first method to extract the FAD list for the device from its bitstream and the FAD list for each Pblock from its partial bitstream.

The CFDATA for each FAD is obtained from Vivado's *.ebc* file. This file is generated from Vivado when the essential bits flag is enabled during bitstream generation. The *.ebc* file contains the CFDATA, in ASCII representation, as obtained from reading back the CM of the device [42], [28]. The *.ebc* file is converted into binary form and is stored in the external memory of an SoC, together with the FAD lists and the bitstream of the design. The device is initially configured with a complete bitstream, and while it is in operation, CM upsets recover with CFDATA from the *.ebc* file. The same *.ebc* file can also be used to recover CFs using blind scrubbing or MER. For example, the Xilinx Soft Error Mitigation (SEM) controller utilizes this file to implement read-back CM scrubbing [43].

VI. Practicality and applicability of FMER

In order to demonstrate the practicality and applicability of FMER in real-world, fault-tolerant SoCs, we implemented TMR versions of various High-Level Synthesis (HLS) applications on a Nexys Video board, which hosts a Xilinx Artix-7 XC7A200T FPGA. We tested each design with all three error recovery methods – blind scrubbing, MER and FMER – and all ER techniques were implemented using the internal TMR RC presented in [44]. The reliability, availability and power consumption of the three techniques were compared for a 2-year LEO mission.

The following subsections provide a description of the SoC implementations and outline how we derived the dependability and energy consumed during error recovery for each technique. Experimental results are presented at the end of the section.

A. Benchmarks and implementation of the SoCs

We implemented a range of different SoCs with applications from the CHStone, DWARV and Bambu HLS benchmark suites [45], [46] that fit onto the XC7A200T device when triplicated. These benchmarks come from various application domains, such as communications, encryption, compression, arithmetic, compute and media.

We used the TLegUp HLS tool [47] to generate TMR Register-Transfer Level (RTL) Verilog code for each application. Each triplicated HLS application consists of 3 modules (i.e. the 3 replicas of the TMR application), while each module incorporates a 3-bit health status port to report which modules of the TMR design, if any, are corrupted. Additionally, each HLS application includes three 1-bit input ports: clock, reset and start, as well as three output ports: finish (1-bit) and result (32-bit). Test vectors for each application are provided by the benchmark suite and are stored on chip to perform functional verification during their operation.

The RC of [44] was used to implement each error recovery technique and to control the ports of the application in order to verify its operation. All designs were synthesized, floorplanned [48] and implemented with Vivado 2017.2.

All three modules of the triplicated RC were placed into one Pblock, which was located in one of the ten available clock regions of the device. The three modules of the RC were placed in a single Pblock because the authors in [44] found that the RC achieves higher performance and reliability with this arrangement. The remaining resources of the device were used to create three further Pblocks, with each one hosting a module of the triplicated application.

B. Utilized configuration frames, essential bits and resources

With the essential bits flag enabled, Vivado generates a mask for the *.ebc* file, which is called an *.ebd* file or essential bits (Ebits) file. This "mask" indicates which bits in the CM may produce a functional error when upset [43], [40], [17]. We analyzed the *.ebd* file with custom tools [40] implemented using the Rapidsmith CAD framework [49] in order to count the Ebits in each Pblock and the SRs of each SoC. The number of Ebits and CFs of the Pblocks and SRs were used to estimate the dependability and energy consumption of each SoC. Note that the number of CFs of each Pblock and the SRs is equal to their FAD list size.

The CFs, Ebits and resource utilization of each SoC are listed in Table III. The sub-columns f and $1-f$ under the F_D column denote the fractions fF_D and $(1-f)F_D$ of the device's CM devoted to the Pblocks (including the RC Pblock) and the SRs respectively. Columns 2-4 of the table report the number of CFs and Ebits of the 1st, 2nd, and 3rd HLS application Pblocks (PBs), respectively, while column 4 shows the same information for the SRs of each SoC. The Ebits of the SRs are further divided into bits located in the configuration (Cfg), IO Block (IOB) and clock (Clk) resource frames of the device. Note that bits for the ICAP are included under the Cfg column. The bits allocated to the remaining resource frames of the SRs are shown in the "Rem" field of the table. These configuration bits realize the routing of the triplicated nets between the

TABLE III
NUMBER OF CFS, ESSENTIAL BITS (EBIT) AND RESOURCE UTILIZATION OF THE PBLOCKS AND SRS FOR EACH SOC

SoC	F_D		1st HLS PB		2nd HLS PB		3rd HLS PB		Support Resources (SRs)						Resource Util.		
										Ebit(K)							
	f	**1-f**	**CFs**	**Ebit(K)**	**CFs**	**Ebit(K)**	**CFs**	**Ebit(K)**	**CFs**	**Cfg**	**IOB**	**Clk**	**Rem.**	**Total**	**Slice (K)**	**BM**	**DSP**
aes	0.23	0.77	1,052	942.80	1,052	1094.72	1,052	949.49	14,122	1.52	0.35	38.77	15.19	55.83	3.88	12	0
aesdec	0.24	0.76	1,124	913.08	1,124	1079.09	1,124	910.81	13,906	1.55	0.35	20.00	19.93	41.83	3.49	12	0
bell	0.14	0.86	344	332.81	600	332.78	600	390.31	15,734	1.70	0.34	4.12	32.07	38.23	0.94	6	0
dfadd	0.23	0.77	1,032	728.43	1,032	730.14	1,032	751.34	14,182	1.69	0.34	17.92	9.39	29.35	5.94	18	0
dfmul	0.19	0.81	924	550.72	924	522.96	580	507.91	14,850	1.44	0.34	16.51	7.37	25.65	4.14	18	48
gsm	0.31	0.69	1,920	1446.63	1,200	1438.75	1,560	1377.18	12,598	1.50	0.34	33.18	223.81	258.83	5.93	21	177
mips	0.12	0.88	380	440.28	408	431.20	380	478.11	16,110	3.58	0.36	12.35	2.13	18.42	1.89	12	12
mmult	0.09	0.91	236	157.88	236	181.71	236	143.88	16,570	1.44	0.34	10.59	11.33	23.71	0.54	12	15
motion	0.43	0.57	2,280	2900.22	2,316	2868.96	2,280	3252.83	10,402	1.70	1.59	55.13	58.02	116.43	7.64	23	0
satd	0.16	0.84	416	354.50	708	384.65	708	364.26	15,446	1.62	0.34	4.10	12.52	18.58	1.73	12	0
sha	0.30	0.70	1,484	1803.28	1,484	1802.68	1,484	2238.30	12,826	1.63	2.06	10.55	56.67	70.92	6.79	30	0
GMean	**0.20**	**0.77**	**807**	**705.13**	**860**	**734.69**	**838**	**732.07**	**14,138**	**1.78**	**0.64**	**18.44**	**19.44**	**43.89**	**2.99**	**15**	**35**

Pblocks of the SoC. Column 6 provides the post-routing resource utilization of each HLS design in terms of number of slices, BRAM16 (BM) and DSP48 (DSP), in order to show the relationship between utilized CFs and resources, respectively. The bottom row of the table provides the geometric mean (GMean) of the experimental results. Information for the RC is excluded from the table, since the same RC is instantiated in all SoCs; the RC is realized with 1062 CFs and 1476.51K Ebits, as averaged over all SoC implementations. Note that the numbers of Ebits for the RC change slightly between each case study due to the heuristic nature of the algorithms used in the CAD tools.

The SoC with the largest fraction of CM devoted to Pblocks is *motion*, with f = 0.43, while the smallest is *mmult*, with f = 0.09. *Motion* utilizes 14 times more slices than *mmult*, which is also reflected in the considerably higher number of Ebits for this design. All SoCs have on average f = 0.20. The analytical results of section IV showed that FMER achieves higher dependability with less energy consumption as $f \to 1$. Therefore, the benefits of FMER would have been more pronounced if the SoCs had greater f.

C. Dependability and energy consumption

In the following, we outline the derivation of the dependability and energy consumption for each error recovery technique.

The SoCs include three simplex subsystems, namely the ICAP, the clock (i.e. clock buffers, clock manager etc.) and IO (i.e. clock input pin) subsystems, that are implemented with the *Cfg*, *IOB* and *Clk* Ebits of Table III, respectively. The SoCs also include three TMR subsystems: (i) the HLS application, which is implemented with the 1st, 2nd and 3rd HLS PB Ebits of Table III, (ii) the RC, which is implemented with 1476.51K Ebits, and (iii) the interconnection nets between the modules of the application and the RC, which are implemented with *Rem.* Ebits of Table III.

1) Reliability of the SoCs: The reliability of the simplex SR subsystems in each SoC is:

$$R_{\text{simplex}}(t) = R_{cfg}(t) \times R_{IOB}(t) \times R_{clk}(t), \tag{34}$$

where $R_{cfg}(t)$, $R_{IOB}(t)$, and $R_{clk}(t)$ are the reliabilities of each of the simplex subsystems, and the reliability of each subsystem is given by Eq. (7). The Ebits of each subsystem (i.e. *Cfg.*, *IOB* and *Clk* in Table III) multiplied by the failure rate of a configuration bit, λ_b, gives its failure rate, λ_m, which is then used in Eq. (7).

The reliability of the TMR subsystems in each SoC is:

$$R_{\text{TMR}}(t) = R_{app}(t) \times R_{RC}(t) \times R_{inet}(t), \tag{35}$$

where $R_{app}(t)$, $R_{RC}(t)$ and $R_{inet}(t)$ is the reliability of the application, RC, and the triplicated interconnection nets between them, respectively, and the reliability of each subsystem is given by Eq. (10), except for SoC/MER, where $R_{inet}(t)$ is given by Eq. (9).

The average Ebits of the 1st, 2nd and 3rd Pblock of Table III are used to calculate the average failure rate, λ_m, of each module in the HLS application. On the other hand, all three modules of the TMR RC are placed in one Pblock, which is implemented with 1476.51K Ebits. Therefore, the average failure rate per RC module is $\frac{1476.51\text{K}}{3}\lambda_b$ upsets/s. Similarly, the average failure rate of 1/3 of the triplicated interconnection nets, which are modelled as belonging to one module in our dependability analysis, is $\frac{\text{Rem.}}{3}\lambda_b$ upsets/s.

The average recovery rate, μ_s, for all TMR subsystems in SoC/Scrub is given by the reciprocal of Eq. (2). In SoC/MER and SoC/FMER, the average number of CFs of the three Pblocks of the HLS application is used in the reciprocal of Eq. (3) to calculate the recovery rate, μ_m, of each module of the HLS application.

In contrast, all three modules of the RC are reconfigured whenever any module fails, since all RC modules have been placed into a single Pblock. The recovery rate for the RC is calculated by the reciprocal of Eq. (3), with F_m = 1062 CFs, i.e. the total number of CFs for the RC Pblock.

According to Eq. (29), the total reliability of SoC/Scrub, SoC/MER and SoC/FMER is:

$$R(t) = R_{\text{simplex}}(t) \times R_{\text{TMR}}(t), \tag{36}$$

where $R_{\text{simplex}}(t)$ and $R_{\text{TMR}}(t)$ are the corresponding reliabilities of the simplex and TMR subsystems of each SoC.

2) Availability of the SoCs: Similar to the Xilinx SEM controller [43], the RC outputs a heartbeat that stops when a fatal failure in the SoC occurs. Fatal failures may occur in the RC when one or more simplex SR subsystems fail or when two or more modules of the RC fail. For example, the heartbeat will stop when a clock resource, the ICAP or the clock input pin of the SoC fails. Fatal failures are recovered by a complete reconfiguration of the FPGA. On average it takes half the period of the heartbeat (T_{HB}), plus the latency of a complete reconfiguration of the FPGA ($T_{\text{reconfig.}}$) to detect a heartbeat stop and to recover the SoC:

$$T_{\text{fatal-recovery}} = \frac{T_{\text{HB}}}{2} + T_{\text{reconfig.}} \tag{37}$$

Therefore, the availability of the simplex subsystems of all SoCs is:

$$A_{\text{simplex}}(t) = A_{cfg}(t) \times A_{IOB}(t) \times A_{clk}(t), \tag{38}$$

where $A_{cfg}(t)$, $A_{IOB}(t)$, and $A_{clk}(t)$ are given by Eq. (8) and $\mu_m = (T_{\text{fatal-recovery}})^{-1}$.

Similarly, the availability of the triplicated subsystems in each SoCs is:

$$A_{\text{TMR}}(t) = A_{app}(t) \times A_{RC}(t) \times A_{inet}(t), \tag{39}$$

where $A_{app}(t)$ and $A_{inet}(t)$ are given by Eq. (11) for SoC/Scrub, and Eq. (12) for SoC/FMER. In SoC/MER, $A_{app}(t)$ and $A_{inet}(t)$ are given by Eqs. (12) and (9), respectively. Lastly, $A_{RC}(t)$ in SoC/Scrub is calculated by adding the probability distributions p_{S0} and p_{S1} in the Markov model of Fig.2 (d), when $\mu_1 = \mu_s$, i.e. the scrub rate of the device, and $\mu_2 = (T_{\text{fatal-recovery}})^{-1}$, i.e. the rate at which the device recovers from a stopped heartbeat. Similarly, $A_{RC}(t)$ in SoC/MER and SoC/FMER is calculated by adding p_{S0} and p_{S1} of the same Markov model, where $\mu_1 = (1062 \times t_F)^{-1}$, i.e. the recovery rate of the RC Pblock, and $\mu_2 = (T_{\text{fatal-recovery}})^{-1}$.

The total availability of each SoC is the product of their simplex and TMR subsystem availabilities:

$$A(t) = A_{\text{simplex}}(t) \times A_{\text{TMR}}(t) \tag{40}$$

3) Energy consumption of CM ER in the SoCs: The CM ER energy consumption of SoC/Scrub is given by Eq. (31), while for SoC/MER it is:

$$E = E_{\text{HLS-app}} + E_{\text{RC}}, \tag{41}$$

where both $E_{\text{HLS-app}}$ and E_{RC} are given by Eq. (32). However, $\overline{F_M}$ in the E_{RC} part is equal to 1062 CFs.

The energy consumption of SoC/FMER is given by Eq. (33), except for T_{MER}, which is derived as follows. The RC and the modules of the HLS application in these SoCs recover with MER for $T_{MER} = T_{\text{HLSapp}} + T_{\text{RC}}$ time of the mission. Both T_{HLSapp} and T_{RC} are equal to $3\overline{\lambda_m} T \overline{F_M} t_F$, however in T_{RC}, $3\overline{\lambda_m}$ and $\overline{F_M}$ are substituted with 1,476.51K and 1062, respectively.

4) Calculation of "w" in SoC/Scrub and SoC/FMER: Xilinx suggests scrubbing at a rate at least 10 times faster than the expected CM upset rate[50]. Therefore, the scrub rate, μ_s, of SoC/Scrub should be:

$$\mu_s = k\lambda_D, k > 10 \tag{42}$$

where λ_D is as in Eq. (22) and k determines how many times faster to scrub than the CM upset rate. By setting μ_s equal to the reciprocal of Eq. (2) in Eq. (42) we get:

$$(\frac{F_D}{2} \times t_F + w)^{-1} = k\lambda_D \tag{43}$$

Solving for w in Eq. (43) sets:

$$w = \frac{1}{k\lambda_D} - \frac{F_D t_F}{2}, \tag{44}$$

such that the scrubbing rate is k times higher than the CM upset rate. Values for w in SoC/FMER were also calculated with Eqs. (22) and (44), but by substituting F_D with the number of CFs for the SRs.

D. Experimental Results

The reliability, availability and energy consumption of all SoCs were calculated with the following parameters: (i) T = 2 years, i.e. the mission time, (ii) λ_b = 1.10E-13, i.e. the configuration bit upset rate of the worst day value for LEO from Table I, (iii) t_F = 16.56 us, i.e. the time required for the RC to read a CF from the external SPI flash of the Nexys Video board and to write it to the CM of the FPGA, (iv) T_{HB} = 100 ms, i.e. the RC heartbeat period, (v) $T_{\text{reconfig.}}$ = 400ms, i.e. the time the device takes to download the bitstream of a design into its CM, and (vi) k = 100, i.e. the scrub rate was set to 100 times the CM upset rate.

TABLE IV
R(T), A(T) AND ENERGY CONSUMPTION FOR A 2-YEAR LEO MISSION

SoC	R(T)	A(T) [# of 9s]			E [Joules]		
	Any	Srub	MER	FMER	Scrub	MER	FMER
aes	0.76	8.67	2.47	8.70	396	0.017	236
aesdec	0.86	8.92	2.24	8.96	396	0.018	229
bell	0.96	9.49	1.85	9.52	396	0.008	293
dfadd	0.87	8.98	2.88	9.01	396	0.014	238
dfmul	0.88	9.02	3.08	9.04	396	0.010	261
gsm	0.79	8.71	0.45	8.76	396	0.030	188
mips	0.91	9.13	3.59	9.15	396	0.008	307
mmult	0.92	9.21	2.72	9.21	396	0.006	325
motion	0.67	8.41	1.37	8.54	396	0.081	128
satd	0.96	9.49	2.63	9.52	396	0.008	282
sha	0.91	8.94	1.39	9.15	396	0.037	195
Gmean	**0.86**	**8.99**	**2.00**	**9.05**	**396**	**0.015**	**237**

Table IV provides the reliability, R(T), availability, A(T), in number of nines [8] and energy consumption, E, in Joules for all SoCs. It is important to note that SoC/Scrub, SoC/MER and SoC/FMER have equal R(T). Our experimental and analytical results show that R_{simplex}(T) in Eq. (36) determines the total R(T) of the SoC. For example, the R_{simplex}(T) and R_{TMR}(T) of *aes* SoC/Scrub in Table IV is 0.76 and 0.99 respectively.

By substituting these values into Eq. (36) we obtain the total R(T) of the SoC, i.e. R(T) = 0.76 $\times$ 0.99 = 0.75 $\approx$ R_{simplex}(T). The *satd* and *motion* applications use the lowest and highest number of Ebits respectively for the implementation of their simplex SRs (*Total - Rem.* in Table III); this is reflected in their reliability, where R(T) = 0.96 for *satd* and R(T) = 0.67 for *motion*. All SoCs achieve on average R(T) = 0.86.

Further, all SoC/Scrub and SoC/FMER designs achieve more than 8 nines A(T). SoC/FMER has on average a slightly higher A(T) than SoC/Scrub, because the TMR subsystems of the SoC recover faster with FMER. In fact, the A(T) of SoC/FMER would have been much greater than SoC/Scrub if the designs where fully triplicated; the availability of the SoCs is mostly determined by the availability of their simplex subsystems, not by their triplicated subsystems, whereby their reliability depends significantly on the error recovery technique used. On average, SoC/MER has approximately 7 nines less A(T) than SoC/Scrub and SoC/FMER, since the triplicated interconnection nets between the Pblocks are not recovered from SEUs. In the long term, the availability of SoC/MER becomes zero when T $\rightarrow \infty$, except if a heartbeat is included in the application to trigger a reconfiguration of the device when the interconnection nets between the Pblocks of the SoC fail.

The energy consumption of SoC/Scrub depends only on the FPGA used, i.e. it depends on F_D and λ_D, which determines μ_s. Therefore, SoC/Scrub requires 396 Joules to recover CM upsets for the 2-year LEO mission. In contrast, the energy used to recover with SoC/MER and SoC/FMER depends on the device and on the circuit design, since the Ebits of each Pblock determines how often a fault in the Pblock will trigger its reconfiguration. SoC/MER and SoC/FMER will have consumed, respectively, 19,812 and 1.68 times less energy than SoC/Scrub during the 2-year mission. Of all SoCs, *mmult* and *motion* SoCs have the lowest and highest f. This is reflected in the energy consumed recovering from SEUs, where *mmult* and *motion* SoC/MER have the lowest and highest consumption, respectively. On the other hand, *motion* SoC/FMER consumes the lowest amount of energy to recover from CM upsets of those applications studied, since f is large and not many CFs of the SRs have to be recovered with periodic scrubbing.

VII. Related Work

A number of authors have proposed novel techniques for recovering soft-errors in the CM of SRAM FPGAs [20], [2]. Both blind and readback and compare scrubbing were first introduced in [51]. The latter recovery technique involves the readback and comparison of the FPGA's CFs with golden CFs stored in external memory, in order to detect and re-write only corrupted CFs. Xilinx embedded Error Correction Codes (ECC) in each CF from Virtex-5 FPGAs onwards in order to reduce the need for external memory to perform CM error recovery. Xilinx also added hardwired logic for reading the FPGA's CFs and applying SECDED combined with CRC mechanisms to correct and detect single- and multi-bit CM upsets, respectively [28]. However, demand for faster and more energy-efficient CM error recovery techniques led to a number of interesting proposals, some of which are described in the following paragraph.

Sari et al. [17] reduces both the time and energy expended to recover CM errors by placing the design in a highly utilized Pblock, in order to gather the design's Ebits into a small chip area, and scrub only the CFs that contain at least one Ebit, also called essential CFs (ECFs). The authors in [52] use a deadline-aware scrubbing scheme which dynamically chooses the frame to commence scrubbing with in real-time FPGA systems so as to reduce the number of missed deadlines. Tonfat et al. [24] introduced a customized design flow that places and routes the three modules of a TMR design in a way that all modules have identical CFDATA and MBUs can be corrected by using information from the TMR scheme. A novel technique that uses a lightweight error detection code and erasure codes to detect and correct MBUs in CFs, respectively, is presented in [53]. Bolchini et al. [18] investigated how the number of partitions and the location of inserted voters affect the size and recovery time of modules in a TMR design that incorporates MER. Cetin et al. [16] proposed a scalable tokenring network to transfer to a RC the health status from the modules of TMR designs that incorporate MER. A follow up paper [27] showed that the most reliable and scalable solution for the implementation of such a network is to use the ICAP to readback CFs containing the health status of these modules.

Our work considers the advantages of scrubbing and MER and proposes FMER as a way to reliably and efficiently recover CM errors in SRAM FPGA SoCs of current and future space missions.

VIII. Conclusions And Future Work

In summary, this work proposes FMER, an energy efficient error recovery (ER) technique that targets TMR-based FPGA SoC designs. To demonstrate the efficacy of this method the reliability, availability and energy consumption of implementations that incorporate either FMER, blind scrubbing, MER or no recovery were modelled and compared. It was shown that MER was the most energy efficient recovery technique. However, it has the lowest configuration memory (CM) fault coverage compared with FMER or scrubbing alone, since with MER errors in the support resources (SRs) are not recovered. Moreover, it was shown that in SoC/Scrub unnecessary energy is wasted refreshing the contents of the FPGA's CM even when it is not corrupted. The provided results, demonstrate that SoC/FMER consumes less energy than SoC/Scrub while it always achieves higher reliability and availability than SoC/Scrub and SoC/MER, especially in high radiation environments or on long missions.

The implementation details of FMER were presented, while 11 different TMR-based SoCs that incorporated blind scrubbing, MER and FMER were implemented on an Artix-7 200T FPGA. On a 2-year LEO mission at high radiation levels, the energy consumption of SoC/FMER was found to be on average 1.68 times less than that of SoC/Scrub, while reliability and availability of the two techniques was almost identical.

Future work involves a more thorough analysis of the Ebits and CFs of the SoCs, which will help us understand how the essential bits are distributed in the CM of the device. We believe that most real-life TMR-based FPGA designs do not use many resources for the implementation of their SRs and therefore only a small fraction of the SR CFs are used. This suggests that the scrubbing time and energy consumption of recovering upsets in the CM of the SRs can be further reduced when only their essential CFs are scrubbed. However, we believe that there would be no reason to do so in Pblocks since they are usually highly utilized and most of their CFs are essential.

References

[1] Quinn, H., Graham, P., Morgan, K., Baker, Z., Caffrey, M., Smith, D., Wirthlin, M., and Bell, R., "Flight experience of the Xilinx Virtex-4," *IEEE Trans. on Nuclear Science*, vol. 60, no. 4, pp. 2682–2690, 2013.

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

[3] ——, "Availability analysis for satellite data processing systems based on SRAM FPGAs," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 52, no. 3, pp. 977–989, June 2016.

[4] Esposito, S., Albanese, C., Alderighi, M., Casini, F., Giganti, L., Esposti, M. L., Monteleone, C., and Violante, M., "COTS-Based High-Performance Computing for Space Applications," *IEEE Trans. on Nuclear Science*, vol. 62, no. 6, pp. 2687–2694, Dec 2015.

[5] *TA 8: Science Instruments, Observatories, and Sensor Systems, 2015 NASA Technology Roadmaps*, 2015.

[6] Kastensmidt, F. L., Carro, L., and da Luz Reis, R. A., *Fault-tolerance techniques for SRAM-based FPGAs*. Springer, 2006, vol. 1.

[7] Quinn, H., Morgan, K., Graham, P., Baker, Z., Caffrey, M., Roussel-Dupree, D., Howes, W., Johnson, E., Johnson, J., Krone, J. *et al.*, *Improving Fault Tolerance of SRAM-Based FPGAs in Harsh Radiation Environments*. CRC Press, 2015, vol. 48.

[8] Koren, I. and Krishna, C. M., *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[9] Carmichael, C., *Triple module redundancy design techniques for Virtex FPGAs Application Note (XAPP197)*, Xilinx Inc., July 2006.

[10] McMurtrey, D., Morgan, K. S., Pratt, B., and Wirthlin, M. J., *Estimating TMR reliability on FPGAs using Markov models*, unpublished, 2008.

[11] Lala, J. H. and Harper, R. E., "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 25–40, Jan 1994.

[12] *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools) Application Note (XAPP1222)*, Xilinx Inc., September 2016.

[13] Sterpone, L. and Violante, M., "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Trans. on Computers*, no. 6, pp. 732–744, 2006.

[14] Li, Y., Nelson, B., and Wirthlin, M., "Reliability Models for SEC/DED Memory With Scrubbing in FPGA-Based Designs," *IEEE Trans. on Nuclear Science*, vol. 60, no. 4, pp. 2720–2727, Aug 2013.

[15] Martin, Q. and George, A., "Scrubbing optimization via availability prediction (SOAP) for reconfigurable space computing," in *IEEE Conf. on High Performance Extreme Computing (HPEC)*, Sept 2012, pp. 1–6.

[16] Cetin, E., Diessel, O., Gong, L., and Lai, V., "Towards bounded error recovery time in FPGA-based TMR circuits using dynamic partial reconfiguration," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–4.

[17] Sari, A. and Psarakis, M., "Scrubbing-based SEU mitigation approach for Systems-on-Programmable-Chips," in *Int. Conf. on Field-Programmable Technology (FPT)*, Dec 2011, pp. 1–8.

[18] Bolchini, C., Miele, A., and Santambrogio, M. D., "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.

[19] Heiner, J., Sellers, B., Wirthlin, M., and Kalb, J., "FPGA partial reconfiguration via configuration scrubbing," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2009, pp. 99–104.

[20] Herrera-Alzu, I. and Lopez-Vallejo, M., "Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers," *IEEE Trans. on Nuclear Science*, vol. 60, no. 1, pp. 376–385, Feb 2013.

[21] Bolchini, C., Miele, A., and Sandionigi, C., "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Trans. on Computers*, vol. 60, no. 12, pp. 1744–1758, Dec 2011.

[22] Cetin, E., Diessel, O., and Gong, L., "Improving Fmax of FPGA circuits employing DPR to recover from configuration memory upsets," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2015, pp. 1190–1193.

[23] Johnson, J. M. and Wirthlin, M. J., "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Field-Programmable Gate Arrays (FPGA)*, 2010.

[24] Tonfat, J., Kastensmidt, F., and Reis, R., "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.

[25] Agiakatsikas, D., Cetin, E., and Diessel, O., "FMER: A hybrid configuration memory error recovery scheme for highly reliable FPGA SoCs," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[26] Rollins, N. H., *Hardware and Software Fault-Tolerance of Softcore Processors Implemented in SRAM-Based FPGAs*. Brigham Young University, 2012.

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

[28] *7 Series FPGAs Configuration User Guide (UG470)*, Xilinx Inc., March 2018.

[29] Heynderickx, D., Quaghebeur, B., Speelman, E., and Daly, E., "ESA's SPpace ENVironment Information System (SPENVIS): a WWW interface to models of the space environment and its effects," *American Institute of Aeronautics and Astronautics AIAA*, vol. 371, 2000.

[30] Tylka, A., Adams, J., Boberg, P., Brownstein, B., Dietrich, W., Flueckiger, E., Petersen, E., Shea, M., Smart, D., and Smith, E., "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.

[31] Lee, D. S., Wirthlin, M., Swift, G., and Le, A. C., "Single-Event Characterization of the 28 nm Xilinx Kintex-7 Field-Programmable Gate Array under Heavy Ion Irradiation," in *IEEE Radiation Effects Data Workshop (REDW)*, July 2014, pp. 1–5.

[32] Mehta, N., *Xilinx redefines power, performance, and design productivity with three new 28 nm FPGA families: Virtex-7, Kintex-7, and Artix-7 devices White Paper (WP373)*, Xilinx Inc., October 2012.

[33] Derek, C. and Crabill, E., *UltraScale Devices Maximize Design Integrity with Industry-Leading SEU Resilience and Mitigation White Paper (WP462)*, Xilinx Inc., February 2015.

[34] Quinn, H., Graham, P., Morgan, K., Baker, Z., Caffrey, M., Smith, D., and Bell, R., "On-Orbit Results for the Xilinx Virtex-4 FPGA," in *IEEE Radiation Effects Data Workshop*, July 2012, pp. 1–8.

[35] Sahner, R. A., Trivedi, K., and Puliafito, A., *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Springer Science & Business Media, 2012.

[36] Maeder, R. E., *The Mathematica® Programmer*. Academic Press, 2014.

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

[38] Hallett, E., *Developing Secure and Reliable Single Device Designs with Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs Using the Isolation Design Flow Application Note (XAPP1086)*, Xilinx Inc., February 2015.

[39] Wirthlin, M., Takai, H., and Harding, A., "Soft error rate estimations of the Kintex-7 FPGA within the ATLAS Liquid Argon (LAr) Calorimeter," *Journal of Instrumentation*, vol. 9, no. 01, p. C01025, 2014.

[40] Sari, A., Agiakatsikas, D., and Psarakis, M., "A Soft Error Vulnerability Analysis Framework for Xilinx FPGAs," in *Proc. of ACM/SIGDA Int. Symp. on Field-programmable Gate Arrays (FPGA)*. ACM, 2014, pp. 237–240.

[41] Lavin, C., Padilla, M., Lamprecht, J., Lundrigan, P., Nelson, B., and Hutchings, B., "RapidSmith: do-it-yourself CAD tools for Xilinx FPGAs," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*. IEEE, 2011, pp. 349–355.

[42] Stoddard, A. G., "Configuration scrubbing architectures for high-reliability FPGA systems," Master's thesis, Brigham Young Uni., 2015.

[43] *Soft Error Mitigation Controller Product Guide (PG036)*, Xilinx Inc., April 2017.

[44] Gong, L., Kroh, A., Agiakatsikas, D., Nguyen, N. T., Cetin, E., and Diessel, O., "Reliable SEU monitoring and recovery using a programmable

configuration controller," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–6.

[45] Hara, Y., Tomiyama, H., Honda, S., and Takada, H., "Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis," *Journal of Information Proces.*, vol. 17, pp. 242–254, 2009.

[46] Nane, R., Sima, V.-M., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y. T., Hsiao, H., Brown, S., Ferrandi, F., Anderson, J., and Bertels, K., "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, 2016.

[47] Lee, G., Agiakatsikas, D., Wu, T., Cetin, E., and Diessel, O., "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 129–132.

[48] Rabozzi, M., Durelli, G. C., Miele, A., Lillis, J., and Santambrogio, M. D., "Floorplanning automation for partial-reconfigurable FPGAs via feasible placements generation," *IEEE Trans.s on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 151–164, Jan 2017.

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

[50] *Correcting Single-Event Upsets in Virtex-II Platform FPGA Configuration Memory Application Note (XAPP779)*, Xilinx Inc., February 2007.

[51] *Correcting single-event upsets through Virtex partial configuration Application Note (XAPP216)*, Xilinx Inc., June 2010.

[52] Pereira-Santos, L., Nazar, G. L., and Carro, L., "Repair of FPGA-Based Real-Time Systems With Variable Slacks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 2, pp. 19:1–19:20, Jan. 2018.

[53] Ebrahimi, M., Rao, P. M. B., Seyyedi, R., and Tahoori, M. B., "Low-Cost Multiple Bit Upset Correction in SRAM-Based FPGA Configuration Frames," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 932–943, March 2016.

Dimitris Agiakatsikas is a Ph.D student at the University of New South Wales, Australia. He received a B.Sc. in Electronics from the Technological Educational Institute of Athens, Greece and a M.Sc. in technology of Embedded Systems from the University of Piraeus, Greece. Before commencing his Ph.D studies, he was an Electronics Engineer at the National Observatory of Athens, Greece. His research interests include fault-tolerant computing, computer-aided design for fault-tolerant FPGA-based systems and dependability modelling.

Ediz Cetin (S'96 – M'02) received the B.Eng. (Hons.) degree in control and computer engineering, and the Ph.D. degree in unsupervised adaptive signal processing for wireless receivers from the University of Westminster, London, U.K., in 1996 and 2002, respectively.

From 2002 to 2011, he was with the University of Westminster, initially as a postdoctoral research fellow and subsequently, from 2006 to 2011, as a Senior Lecturer. From 2011 to 2017, he was a Senior Research Associate with the Australian Centre for Space Engineering Research, University of New South Wales (UNSW), Sydney, NSW, Australia. He is currently a Senior Lecturer with the School of Engineering, Macquarie University, Sydney, NSW, Australia. His research interests include interference detection and localization, fault-tolerant reconfigurable circuits, adaptive techniques for RF impairment mitigation for communications and global navigation satellite system receivers, and design and low-power implementation of digital circuits. To date, he has authored or co-authored more than 60 technical publications, a book chapter, and holds two patents in the areas of communications and GNSS receivers.

Dr Cetin is a member of the Institution of Engineering and Technology (IET) and serves as the Chair of the IET New South Wales (NSW) Local Network, as well as the Chair of the Institute of Electrical and Electronics Engineers NSW Circuits and Systems/Solid-State Circuits/Photonics/Electron Devices joint chapter.

Oliver Diessel is an Associate Professor in the School of Computer Science & Engineering, at the University of New South Wales in Australia. He gained B.E., B.Math., and Ph.D. degrees from the University of Newcastle, Australia. His research interests encompass the design and application of dynamically reconfigurable systems and technology, including modelling, design methods, and run-time support for such computer systems. He has co-authored over 70 publications on these topics.