Open-source IP cores for space

A processor-level perspective on soft errors in the RISC-V era

Di Mascio, Stefano; Menicucci, Alessandra; Gill, Eberhard; Furano, Gianluca; Monteleone, Claudio

DOI
10.1016/j.cosrev.2020.100349

Publication date
2021

Document Version
Final published version

Published in
Computer Science Review

Computer Science Review 39 (2021) 100349

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Open-source IP cores for space: A processor-level perspective on soft errors in the RISC-V era

Stefano Di Mascio [a,*], Alessandra Menicucci [a], Eberhard Gill [a], Gianluca Furano [b], Claudio Monteleone [b]

[a] Delft University of Technology, 2629 HS Delft, The Netherlands
[b] European Space Agency, 2200 AG Noordwijk, The Netherlands

ARTICLE INFO

Article history:
Received 4 August 2020
Received in revised form 4 December 2020
Accepted 9 December 2020
Available online 24 December 2020

Keywords:
Processors
Fault tolerance
Space

ABSTRACT

This paper discusses principles and techniques to evaluate processors for dependable computing in space applications. The focus is on soft errors, which dominate the failure rate of processors in space. Error, failure and propagation models from literature are selected and employed to estimate the failure rate due to soft errors in typical processor designs. A similar approach can be followed for applications with different radiation environments (e.g. automotive, servers, experimental instrumentation exposed to radiation on ground), by adapting the error models. This detailed white-box analysis is possible only for open-source Intellectual Property (IP) cores and in this work it will be applied to several open-source IP cores based on the RISC-V Instruction Set Architecture (ISA). For these case studies, several types of redundancy described in literature for space processors will be evaluated in terms of their cost-effectiveness and expected final in-orbit behavior. This work provides a comprehensive framework to assess efficacy and cost-effectiveness of redundancy, instead of listing and categorizing the techniques described in literature without assessing their relevance to state-of-the-art designs in space applications.

Contents

* Corresponding author.
E-mail addresses: s.dimascio@tudelft.nl (S. Di Mascio), a.menicucci@tudelft.nl (A. Menicucci), e.k.a.gill@tudelft.nl (E. Gill), gianluca.furano@esa.int (G. Furano), claudio.monteleone@esa.int (C. Monteleone).

https://doi.org/10.1016/j.cosrev.2020.100349

List of abbreviations

AC	Average Criticality
ACE	Architecturally Correct Execution
ALU	Arithmetic-Logic Unit
ASIC	Application-Specific Integrated Circuit
AVF	Architectural Vulnerability Factor
BP	Branch Prediction
CC	Clock Cycle
CI	Cell Interleaving
CL	Criticality Level
COTS	Commercial-Off-The-Shelf
CPI	Cycles Per Instruction
CSR	Control and Status Registers
CU	Constant Utilization
CVF	Cache Vulnerability Factor
CW	Constant Workload
DC	Data Cache
DMR	Double Modular Redundancy
DUE	Detected Uncorrectable Error
ECC	Error Correcting Code
EDAC	Error Detection and Correction
EDC	Error Detecting Code
FDSOI	Fully-Depleted Silicon-On-Insulator
FF	Flip Flop
FI	Fault Injection
FinFET	Fin Field-Effect Transistor
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FRF	Floating Register File
FT	Fault-Tolerant
GCR	Galactic Cosmic Rays
GEO	Geostationary Orbit
GP	General Purpose
HC	High Criticality
HPC	High-Performance Computing
IB	Instruction Buffer
IC	Instruction Cache
ID	Interleaving Distance
IF	Instruction Fetch
II	Instruction Issue
IOD	In-Orbit Demonstration
IP	Intellectual Property
IPC	Instructions Per (clock) Cycle
IRF	Integer Register File
ISA	Instruction Set Architecture
IU	Integer Unit
L1	Level 1
L2C	Level 2 Cache
LC	Low Criticality
LEO	Low Earth Orbit
LET	Linear Energy Transfer
LLC	Last-Level Cache
LSU	Load and Store Unit
M/D	Multiplier and Divider
MBU	Multiple Bit Upset
MCU	Multiple Cell Upset
MD	MBU Dominated
MLP	Memory Level Parallelism
MPEG	Moving Picture Experts Group
MTTE	Mean Time To Event
MTTF	Mean Time To Failure
OBC	On-Board Computer
OoO	Out-of-Order
OS	Operating System
QoS	Quality of Service
RF	Register File
RHBD	Radiation-Hardened By Design
ROB	ReOrder Buffer

RR	Register Rename
RTL	Register Transfer Level
SAA	South Atlantic Anomaly
SBF	Single Bit Flip
SBU	Single Bit Upset
SD	SET Dominated
SECDED	Single Error Correction and Double Error Detection
SED	Single Error Detection
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SOI	Silicon-On-Insulator
SRAM	Synchronous Random Access Memory
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
UT	Unexpected Termination
WB	Write-Back
WT	Write-Through

1. Introduction

Space systems rely on digital electronics for on-board data handling and processing, and processors are key elements (along with memories and interfaces) to achieve such functionalities [1]. When selecting a processor for satellite data systems, typically two choices are available: either a space-grade processor with long flight heritage and well-characterized behavior (e.g. LEON processors [2]), or a proprietary Commercial-Off-The-Shelf (COTS) processor employed as a black box (sometimes after adequate radiation test [3,4]). The latter is preferred to the former when the performance required cannot be met with space-grade processors [5], which typically lag behind their commercial counterparts in terms of performance [6]. The recent availability of open-source Intellectual Property (IP) cores for terrestrial applications , mainly based on the RISC-V Instruction Set Architecture (ISA) [7], allows for a better understanding of their vulnerability, avoiding black-box characterization (typical of proprietary COTS components) and allowing a trade-off between the two approaches. A better modeling of the inner working of processors can both help choosing the best IP core and its configuration. For instance, in [8] the lack of public Register Transfer Level (RTL) models (typical of proprietary processors) is identified as the main issue when trying to characterize the effects of upsets in a microarchitecture (mainly because it is not possible to estimate the exact number of sequential elements). Furthermore, the authors of [9] suggest that the failure rate measured with beam experiments is much larger than the one estimated by Fault Injection (FI) due to unknown proprietary parts of the real physical hardware platform compared to the virtual platform where the FI was carried out.

Once the vulnerability of a processor is estimated, it can be reduced employing redundancy. Redundancy typically comes with significant area, power and performance overhead. Therefore, assessing its cost-effectiveness is crucial. However, the amount and type of optimal redundancy can change drastically depending on the requirements in terms of dependability (i.e. reliability, availability, safety [10]) and performance as well as on the target environment. For instance, in automotive the focus of the standard ISO-26262 [11] is on functional safety. For this reason, several Application-Specific Integrated Circuits (ASICs) for automotive employ two processors executing instructions in lockstep, so that errors can be detected comparing the outputs of the two replicas and the processors are restarted in case of mismatch [3]. A similar approach can reduce availability, as for instance even benign differences at the outputs of the processors will cause a reset. Furthermore, as long as the safety requirements are met, availability is not a primary concern in automotive. This is not the case for space applications, as dependable processors in space are expected to provide a certain service without interruptions over a certain span of time, hence the focus is instead on availability. For example, in the case of a geostationary telecommunication satellite the time span of a mission could be more than 15 years in which the whole space system is expected to provide a certain service 99.9% of the time [12]. Therefore, the unavailability budget for the On-Board Computer (OBC) is even tighter. Furthermore, when the processor is intended for usage in space, the presence of ionizing radiation makes soft errors far more likely and the amount of redundancy must be carefully evaluated as power and area available in space data systems are typically very limited. On the other hand, loss of performance in space data systems can be easily tolerated in most cases. In High-Performance Computing (HPC) the constraints are the opposite, as the amount of loss in terms of performance that can be tolerated is typically very limited [13].

1.1. Objective

The objective of this paper is to introduce readers familiar with processors and typical performance/power/area trade-offs in digital electronics [14] to consider also dependability with quantitative tools, taking as a relevant example the extreme case of space applications. This work develops a *comprehensive* framework at processor-level[1] to assess and mitigate the soft error vulnerability of processors in a *cost-effective* way. The need for this work and its nature of a survey, instead of a completely experimental paper (like for instance [16]), is given by the fact that most of the works in literature describe in great detail specific aspects of the vulnerability of specific hardware structures and how to address soft error vulnerability of specific units in a processor (e.g. register files [17], data [18] and tag [19] array in caches). This sub-processor approach is dictated by the extensive work required to build a relevant test setup and to the number of experiments required to get meaningful statistics. In this paper we will complement these works by putting their results together, using them to develop a comprehensive framework that the reader can reuse and readapt to its own designs or when evaluating an open-source IP core. Although using several extrapolations and approximations, this approach allows the reader to have a complete view of the specific challenges involved in the design of a dependable processor for space and to estimate the effects of a different environment/technology/microarchitecture/redundancy given limited experimental data.

1.2. Scope and related works

The techniques to increase dependability reported in this work are those typically employed for space processors such as LEON [2], TCLS [20] and those developed by Boeing [21]. Therefore, this work can be read as a survey of state-of-the-art techniques to evaluate and design processors for dependable

[1] That is, including caches but excluding peripherals, interconnects, interfaces, off-chip memories and main memory. However, processors are typically included in a System-on-Chip (SoC) together with peripherals and memories. To further extend this framework, the reader can refer to the work in [15], which estimates the impact of other subsystems of SoCs.

space applications. For readers interested in a wider range of applications, there are instead some related works in literature. A survey listing techniques to model and improve reliability of computing systems was published in [22]. From there, additional techniques not included in this work (both because they are not relevant to space processors and for sake of brevity) can be included in our framework. An introduction to the soft error problem in processors was published in [23], covering soft error mitigation techniques at device, circuit, microarchitectural and software level. In this work, we will develop further all the aspects related to the microarchitecture and will establish a model built putting together results from literature. This will give more insights on how to evaluate open-source IP cores and how to enhance their dependability in a cost-effective way. For instance, only 10 out of 132 references of this paper are used also in [23] and some of them are only necessary to introduce the topic (e.g. [10], which proposes a nomenclature for dependable systems). Other comprehensive frameworks were proposed in recent years (2016–2019) [24,25]. The present framework differs for three reasons: it is built from a survey of the literature, it has a wider scope (e.g. comprising definition of threat models from the space environment, and considerations on availability and validation) and it is described step by step to the reader (see Table 15). The reader can therefore implement the framework for its own designs and contribute to its extension in a straightforward way.

1.3. Outline

To introduce the reader to the problem, the first part of this paper follows the error from its generation to the occurrence of the service failure (as shown in Fig. 1). In Section 2.1, typical faults in space processors are identified and an error model is associated to each of them, in Section 2.2 the outcomes of the defined error models are analyzed up to the service interface, and in Section 2.3 the application-dependent effects of errors at the service interface are analyzed.

The second part of the paper follows instead the steps of a typical design flow for a fault-tolerant processor. In Section 3.1 a quantitative model to identify the most vulnerable units of processors is presented and in Section 3.2 it is applied to four different processor designs. Section 4 then analyzes several types of redundancy and discusses their cost-effectiveness. Section 5 discusses aspects related to validation and in-orbit expected behavior. Finally, Section 6 draws conclusions.

2. Identifying and modeling threats

Fig. 1 shows how threats[2] interact with a processor. A failure is a deviation from the expected behavior of the service provided at the service interface [10], and it is caused by one or more deviations from the correct state of the system (errors). The cause of the error is called fault [10]. Changes in the charge stored in nodes due to particle strikes are typical faults in space processors (external faults in Fig. 1), and they are called soft errors as they can be removed simply overwriting them with the correct value [26]. This is not the case for hard errors [27], where the distinction between fault (e.g. defective gate) and error (e.g. wrong result of a calculation) is needed for correct recovery (e.g. to replace a defective unit with a spare unit).

[2] In [10], the term 'threats' refers to faults, errors, and failures.

2.1. Fault and error models

Regardless of the specific threats due to the space environment, processors in space have to be first of all robust against faults common to processors in terrestrial applications.[3] For instance, simulations for a 32 nm ASIC technology show that the data propagation delay of Flip Flops (FFs) increases less than 5% in 5 years of stress conditions due to aging [28]. This can be taken into account during design by applying larger margins on the maximum allowed frequency. Aging and hard faults due to imperfections or wear out can be classified as internal faults in Fig. 1, for which environmental conditions and specific activation patterns are required in order to generate errors. Despite hard errors, soft errors due to radiation typically dominate the failure rate of processors already in terrestrial environments. In [29] the ratio of soft errors to hard errors for Synchronous Random Access Memory (SRAM) arrays in processors ranges from 77 to 735, and in [30] 99.36% of the errors in an SRAM array are soft errors while 0.64% are hard errors. Soft errors in space are even more predominant, as in this case charged particle strikes are more common (outside the Earth atmosphere the flux of particles is higher) and different particles are present (heavy ions and protons instead of neutrons) [31].

Furthermore, our focus in this paper is on faults capable of generating functional errors and we will not consider faults which generate electrical failures like Single Event Latchups [32] and increase of absorbed current due to Total Ionizing Dose (TID) effects [33]. The reason is that those are typically not addressed at microarchitectural level but at technology and electrical level instead.

2.1.1. Upsets

Ionizing particles can change the value stored in a single or more sequential elements. In the first case, the terms Single Event Upset (SEU) or Single Bit Upset (SBU) are employed. In the second case, the term Multiple Bit Upset (MBU) can be used.[4]

The upset rate λ_{ev} mainly depends on the radiation environment (including also shielding), the technology[5] and the choice of the sequential and combinational elements in the processor within the same technology. The upset rate can be either estimated with environmental models or measured on the field [34]. In the first case, a standard approach is to carry out a radiation test composed of several test runs with particles with different Linear Energy Transfer (LET)[6] and measure the respective cross section.[7] Afterwards, tools like SPENVIS [36] are used to calculate the differential LET spectrum which can be obtained from the

[3] In our discussion we do not include systematic failures due to bugs that should not be considered part of dependability but of normal engineering practice (verification).

[4] Sometimes the term Multiple Cell Upset (MCU) is employed instead, while MBU is reserved to cases where the multiple upsets are in the same Error Detection and Correction (EDAC)-protected word. Furthermore, the notation MBU(n) will be employed to indicate MBUs causing n upsets with a single particle strike.

[5] Several factors can be included in the technology. For instance, the error rate per bit on a specific technology depends on the voltage chosen (in [16] decreasing the voltage from 1.2 V to 0.8 V results in an increase of the error rate by a factor 1.5x up to 3x, depending on the radiation source). However, as shown in [16], this does not change the ratio between errors from combinational and sequential logic.

[6] The LET represents the energy loss of the particle when it travels a unit distance in the semiconductor [35]. It is typically normalized to the density of the material and given in MeVcm^2/mg.

[7] The device cross section for a given LET is defined as the quantity that multiplied by the particle flux produces the SEE rate of that flux of particles. It is typically given as cm^2/device or cm^2/b [35].

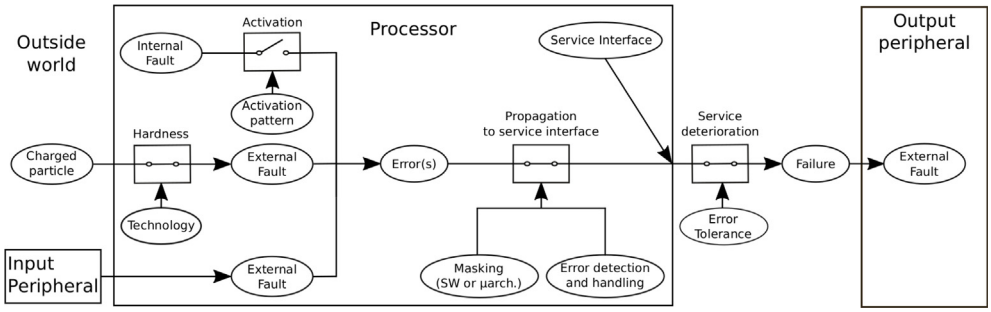

Fig. 1. Typical interactions of threats with a processor providing a service to an output peripheral.

particle differential energy spectra in a certain orbit [35]. The upset rate can be then found with the following integral [35]:

$$\lambda_{ev} = \int_0^{inf} \int_{-1}^{1} \int_0^{2\pi} f(L,\theta,\phi)\, \sigma(L,\theta,\phi)\, d\phi\, dcos(\theta)\, dL \qquad (1)$$

where the differential flux f and the cross section per bit σ depend on the LET L and the incidence and rotation angles (θ and ϕ) [35].

Data from [37] shows for a commercial 28 nm Fully-Depleted Silicon-On-Insulator (FDSOI) SRAM an in-orbit SEU rate of 4.66×10^{-9} upsets/bit/day for solar minimum in Geostationary Orbit (GEO). From data in the same work, an estimation of 5×10^{-7} for *worst week* in GEO and 5×10^{-10} upsets/bit/day for Low Earth Orbit (LEO) can be taken (three orders of magnitude less than GEO worst conditions). Data from [38] show that considering different time spans will have different worst cases, e.g. the upset rate for the worst case of an SRAM array for one week in GEO is one order of magnitude lower than the worst case for 5 min, the latter reaching an upset rate of around 10^{-2} upsets/bit/day (similar values are given in [39], some of them even reaching 10^{-1} upsets/bit/day). Furthermore, the upsets are not homogeneously distributed in a certain orbit. For instance, all reboots in [40] (LEO) due to upsets happened in the South Atlantic Anomaly (SAA) and over the poles, where the level of radiation is higher due to the lower magnetic field shielding. To provide a comparison with processors in terrestrial environment, the upset rates at sea level in [41] is assumed to be 2.7×10^{-11} upsets/bit/day, which is four orders of magnitude less than for the 28 nm FDSOI in GEO (*worst week*).

The radiation environment experienced by the processor depends also on the amount of shielding, which cannot be controlled by the designer of the processor. In [38] it is shown that the reduction of upset rate due to an ideal aluminum sphere going from 0.1 mm to 2.5 mm is of 4 orders of magnitude for a 45 nm Silicon-On-Insulator (SOI) SRAM in the case of trapped protons, typical of LEO [42]. Considering an electronic box in a spacecraft brings the upset rate down of roughly another order of magnitude. However, in [38] it is shown that Galactic Cosmic Rays (GCR) are insensible to shielding depths. This causes a plateau of 8.64×10^{-7} upsets/bit/day for the SRAM technology considered in [38], where adding more shielding does not improve the radiation tolerance of the part which must be addressed exclusively at semiconductor level.

In a similar manner, different technologies exhibit different upset rates in the same radiation environment. A typical Radiation-Hardened By Design (RHBD) SRAM memory based on a 250 nm technology has been reported in [34] to operate in GEO with an average of 1.8×10^{-10} upsets/bit/day. A commercial SRAM based on 65 nm bulk technology in [43] is reported to experience an average of 1.5×10^{-7} upsets/bit/day in LEO, and in GEO would show an even higher upset rate. Space-grade processors are currently based on 65 nm (e.g. GR740 [1]) or even 180 nm (e.g. GR716 [2]) RHBD ASIC technologies, while typical processors for terrestrial application are typically below 28 nm (e.g. [44]). These newer technologies are expected to be more vulnerable: when scaling from 65 nm to 14 nm the upset rate increases from around 10^{-12} to around 10^{-11} upsets/bit/day for planar bulk technologies, while it increases from 10^{-11} to 10^{-10} upsets/bit/day for FDSOI and Fin Field-Effect Transistor (FinFET) technologies [45] (all of them measured at ground altitudes). For all three types of technologies the increase happens when going beyond 28 nm, while from 65 to 28 nm the upset rate is constant or slightly decreasing.

Even in the same technology, different sequential elements composing the processor can have different upset rates. For instance, the OpenSPARC T2 in [46] (65 nm) is mainly composed of SRAM arrays optimized for density (for caches) with an upset rate ranging between 8.58×10^{-13} and 1.14×10^{-12} upsets/bit/day, less-dense and higher-performance SRAM arrays (for register files) with an upset rate per bit of half or less and FFs with an upset rate per bit of one-third or less compared to the SRAM array optimized for density. However, as [47] shows, this is not always the case and several technologies (especially newer ones) show the opposite situation. As a matter of fact, the ratio of the upset rate of FFs to SRAM cells in [47] is 0.44 for 130 nm, 1.96 for 90 nm, 1.75 for 65 nm and 1.15 for 40 nm technologies.

The differentiation between FFs and SRAM arrays is also required because FFs have temporal masking, which is not present in SRAM arrays. If we consider an upstream sequential element connected to a downstream element through combinational logic, an upset happening in the upstream element between $t = t_{samp} - T_{prop}$ and $t = t_{samp}$ (where t_{samp} is the sampling instant given by the clock and T_{prop} is the time required for the correct sampling of a signal propagating from the upstream to the downstream element) will not propagate to the sequential elements downstream. A sampling factor can be defined as $SF_{FF} = 1 - \frac{T_{prop}}{T_{clk}}$, where T_{clk} is the clock period for the FFs. This implies that the fraction of temporally masked errors in FFs actually increases with the frequency [16]. Despite this masking, typical models used in literature assume a constant failure rate for FFs when changing frequency [48], while more refined analyses find that there is an increase of the failure rate due to a Single Event Transient (SET) mechanism in the combinational logic between master and slave [49]. Data provided in [49] show that this increase is very small, when considering a single FF the maximum found is 5×10^{-15} errors/bit/day/MHz. Considering a design going from 100 MHz to 1 GHz, the error rate increases by 4.5×10^{-12} errors/bit/day, which is of orders of magnitude less even compared to the less vulnerable technologies for space (around 10^{-10} upsets/bit/day). However, as mentioned in [16], testing shift registers where T_{prop} is close to zero fails to take into account temporal masking, and SF_{FF} is close to one for practical values of frequency. On the other hand, when testing a circuit with both sequential and combinational logic, understanding which of the

two generated the error sampled in an FF to validate the temporal masking model is a daunting task. According to the model in [16], temporal masking instead can have a considerable impact. In [16] an average SF_{FF} of 66.6% is given. When lowering the frequency on the same design the sampling factor increases, until for 100 MHz the sampling factor gets to 96.66%.

Even the same type of sequential element can come in different sizes for the right performance/power/area trade-off. Data from [50] shows that FFs for a 65 nm commercial bulk technology have upset rates ranging between 1.6×10^{-7} upsets/bit/day (fastest FF) and 4.1×10^{-7} upsets/bit/day (slowest FF, 2.56x more vulnerable). Rad-hard (radiation-hardened) versions of the same technology have upsets rates ranging from 8.12×10^{-8} to 1.82×10^{-9} upsets/bit/day (2.24x increase of vulnerability with a 3x increase in drive strength). From [51] it can be seen that a rad-hard version of a FF on commercial technology can achieve a reduction of upset rate of 350x. In [16] several frequency targets (ranging from 100 MHz to 900 MHz) are set when synthesizing a processor, generating implementations with different mix of FFs. This increases vulnerability up to 10% (i.e. $RV_{FF} = 1.1$) taking the less vulnerable as reference. This increase follows a regular pattern, growing with the difference between the target frequency (e.g. 900 MHz) and the real clock frequency (e.g. 100 MHz).

The upset rate λ_{ev} is typically assumed constant [52] (i.e. inter-arrival times of raw errors in a component are independent [52]) and therefore the reliability function is exponential for each sequential element, i.e. $R_b(t) = e^{-\lambda_{ev} \times t}$. The use of the exponential distribution implies that the error rate of a series of elements becomes the sum of the error rates and the probability of not having an upset in the processor is $R_{SEU}(t) = e^{-SER_{SEU} \times t}$, where the Soft Error Rate (SER) due to SEUs is:

$$SER_{SEU} = \lambda_{ev} \times \left(N_{SRAM} \times RV_{SRAM} + N_{FF} \times RV_{FF} \times SV_{FF}\right) \quad (2)$$

where N_{SRAM} and N_{FF} are respectively the number of SRAM cells and FFs. RV_{SRAM} and RV_{FF} are the average vulnerability of respectively SRAM cells and FFs employed relatively to a reference sequential element with event rate λ_{ev}.

When considering MBUs, they can be measured as fraction of the total events. This means that if two events happen, one generating a SBU and one a MBU, the fraction of MBU is 50% regardless of the number of errors due to the MBU. Data from [53] show that for SRAM arrays in a 90 nm ASIC technology 95% of events cause a SBUs, 4% cause a MBU(2) and 1% cause MBU(3). For 65 nm SRAM arrays the situation reported in [53] is quite different: 45% are SBUs, 18% are MBU(2), 10% are MBU(3) and 27% are MBU($\geq$4). As a pessimistic estimation for Ultra Deep Sub Micrometer (UDSM) technologies data from [54] for a 32 nm SRAM array[8] can be taken: in this case the fraction of SBUs is 24%, the fraction of MBU(2) is 52%, the fraction of MBU(3) is 3% and the fraction of MBU($\geq$4) is 21%.

2.1.2. Single event transients (SETs)

A single particle hitting a combinational node is able to cause a transient voltage pulse [55]. This pulse can be latched by the sequential elements downstream and can be either seen as a single error or multiple errors in sequential elements by the user (e.g. software level). Even if the user is not able to distinguish between SETs and upsets, SETs have different generation mechanisms that require different redundancy techniques compared to SBUs and MBUs. As a matter of fact, SETs have additional levels of masking (electrical and logical) [56]. Furthermore, they have a different temporal masking mechanism: if the pulse reaches the sequential element outside from the sampling window, then the spike is not sampled and the error not generated. This implies that the contribution of SETs increases with the increase of the frequency. The reason is that when frequency increases, the sampling window becomes a larger fraction of the total time.

In relatively old technologies (e.g. technology nodes larger than 90 nm), SETs are not predominant as they are attenuated by large capacitance (electrical masking) and the low clock frequencies make the sampling unlikely (temporal masking) [57]. In more recent technologies instead, capacitance is reduced and the clock frequency is higher. For this reason, the probability that a spike is latched increases [57]. In [58] a comparator, an FF chain and an inverter chain are tested to compare the contribution of SETs and SEUs on a 45 nm bulk technology. The chain of inverters in [58] has a depth (12 stages) to emulate the highest electrical masking available typically in designs and accounts only for electrical and temporal masking, while the comparator also account for logical masking. As logical masking depends upon the input combination, in [58] a best, average and worst case are given. The worst case counts around twice the SETs compared to the best case. Furthermore, in [58] errors due to combinational logic (inverter chain) are less than one eighth of errors in sequential elements up to 100 MHz, around half at 500 MHz and uncertainties overlaps for 1 GHz (even if the expected value is still at half the sequential elements). The crossover frequency is around 1.5 GHz for the inverter chain and between 1.7 and 5 GHz for the comparator. However, considering that the vulnerability of FFs decreases with frequency, the contribution of sequential logic would be higher and the crossover frequency lower. This shows how increasing frequency does not necessarily increase the error rate, but certainly increases the relative vulnerability of combinational logic in the design, making optimal redundancy for low frequency not fit for higher frequencies, as it will be shown in Section 4. The SER due to SETs can be written as:

$$SER_{SET} = \lambda_{ev} \times \frac{A_{comb}}{A_b} \times SF_{SET} \times RV_{comb} \quad (3)$$

where A_{comb} is the area of the combinational logic, A_b the area of the reference sequential element associated with λ_{ev}, and SF_{SET} is the sampling factor of SET pulses (indicating how many pulses are actually sampled by the sequential elements downstream). In [59] the overall probability of a SET being latched given a strike is 16.55% for 45 nm, 21.31% for 32 nm, 26.27% for 22 nm and 28.71% for 16 nm. We will consider a best case with $SF_{SET} = 0\%$, an average case with $SF_{SET} = 15\%$ and a worst case with $SF_{SET} = 30\%$. Also in this case we defined a RV_{comb} that keeps into account different frequency targets that will imply the choice of different combinational elements. Data from [16] show that different timing targets (e.g. 100 MHz) can increase the failure rate of combinational logic by 2x compared to the timing target minimizing the failure rate (900 MHz), when running both implementations at the same frequency (100 MHz). It should be noted that in the case of combinational logic, as opposed to sequential elements, smaller gates are more sensitive to SETs [16].

2.1.3. Errors in SRAM-based FPGAs

The correct behavior of processors implemented on SRAM Field Programmable Gate Arrays (FPGAs) is dependent on large configuration memories. An interesting finding in [60] is that the percentage of bit flips in the configuration memory normalized to the resource utilization (fraction of sensitive bits in the configuration memory divided by the fraction of slices utilized in the FPGA) is roughly independent from the specific IP core (ranging from around 3% to around 6%). However, the impact of soft errors on the microarchitecture is similar to those of hard errors (e.g. stuck-at [61]) and therefore they will not be included in this framework.

[8] It is not possible to define worst cases and best cases that will be always such for each type of redundancies explored in the following sections. So as a metric to define the best, average and worst case in Table 1, the total percentage of MBUs is considered.

Table 1
Error models for soft errors identified for space processors (data derived from [16,53,54,59]) for different types of technology defined in Section 2.1.4: Low Criticality (LC), Average Criticality (AC), High Criticality (HC), SET Dominated (SD) and MBU Dominated (MD).

Technology	LC	AC	HC	SD	MD
$SF_{SET\%}$	0%	15%	30%	30%	0%
$SF_{FF\%}$	97%	82%	67%	67%	97%
$SBU_{\%}$	95%	45%	24%	95%	24%
$MBU(2)_{\%}$	4%	18%	52%	4%	52%
$MBU(3)_{\%}$	1%	10%	3%	1%	3%
$MBU(\geq 4)_{\%}$	0%	27%	21%	0%	21%
$MBU(even)_{\%}$	4%	45%	73%	4%	73%

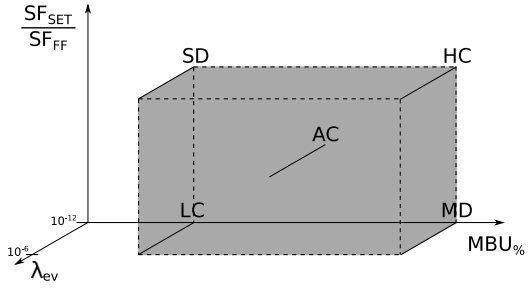

Fig. 2. Technology space considered in this work, delimited by dashed lines. 'Edge' and 'average' technologies in black solid lines.

2.1.4. Model adopted

Given the discussion in previous sections, the *SER* of the processor will be estimated as $SER = SER_{SEU} + SER_{SET}$, which can be rewritten as:

$$SER = \lambda_{ev} \times N_{eq} \tag{4}$$

where N_{eq} is the number of reference sequential elements that would produce the same SER given a certain λ_{ev}. In our model (Eqs. (2) and (3)):

$$N_{eq} = N_{SRAM} \times RV_{SRAM} + N_{FF} \times RV_{FF} \times SV_{FF} + \frac{A_{comb}}{A_b} \times SF_{SET} \times RV_{comb} \tag{5}$$

Finally, the effect of the fraction of MBUs on the final failure rate will be taken into account as described in Section 3.3.3. In Table 1 the parameters of the proposed model for 5 different types of technologies are reported. These parameters describe a three-dimensional space of technologies, as shown in Fig. 2. Four of the selected technologies (LC stands for Low Criticality, MD stands for MBU Dominated, HC stands for High Criticality, and SD stands for SET Dominated) are edges of a solid in this space and one is the average case (AC stands for Average Criticality). As a matter of fact, technologies not only affect λ_{ev} (quantity of events), but with the relative contribution of SEUs, SETs and MBUs (quality of events) they also determine which redundancy is more effective. The rest of the edges of the solid are defined considering only a finite range of λ_{ev} ($10^{-12} - 10^{-6}$), defined according to average values experienced during several missions (Section 2.1.1), while considerations on extreme conditions such as *worst week* and *worst 5 minutes* in GEO will be carried out in Section 4.1.2.

2.2. Error propagation to the service interface

Errors generated by a fault not masked at the technology level can be masked during their propagation to the service interface (even when not considering redundancy) at the microarchitectural level (e.g. the error does not influence the behavior of the processor) and at the software level (e.g. an error which affects a bit in an unused instruction or is used only by a dynamically dead instruction[9]), as shown in Fig. 3. When the error is masked, the application terminates normally and output pins (and files) do not differ from the fault-free execution.[10]

When redundancy is employed, along with the intrinsic microarchitectural and software masking, error detection and handling are also possible. The capability of a processor to avoid an error to turn into a failure is referred to as "fault tolerance" [10]. The possible outcomes of error detection and handling are:

- Correctable error: the error detection and handling mechanism proceeds to correct the error (correction). However, when more errors than expected are present, the correction can be wrong (miscorrection [64]).
- Detected Uncorrectable Error (DUE): the error detection and handling mechanism is able to detect the error and to prevent it from propagating to the service interface [65]. The reaction to a detected DUE (e.g. rollback) may cause penalties in terms of availability.
- Unexpected Termination (UT): its effect on the error propagation is the same as a DUE, but it is typically caused by the Operating System (OS) and software [66] instead of hardware. For instance, a process may terminate abnormally thanks to built-in protections (memory access violation, kernel panic, and arithmetic exception) triggered by an anomalous behavior [67].
- Undetected: in this case the redundancy employed fails at detecting the error during its propagation and no action is taken.

2.3. Service interface and error tolerance

The system service defines the service interface at which the service is to be provided and which outputs of the software (e.g. variables directly mapped to a failure) and hardware (e.g. signals to other subsystems) will be able of propagating the errors. An error, when propagated to the service interface, can generate wrong data, wrong commands or unavailability of the system (Fig. 3). The unavailable state can be expanded in a case where the unavailability is due to the intrinsic vulnerability of the processor (i.e. hang) and a case where it is due to error handling.

2.3.1. Intrinsic error tolerance

In many works, wrong data and wrong commands on the output are both assumed to be a failure, calling this Silent Data Corruption (following the terminology of [65]). However, this is not always the case, as some services are inherently tolerant to wrong data at the service interface. In [68] a system is defined as error tolerant with respect to a service, if the system produces *acceptable* results to the end user according to a certain Quality of Service (QoS) even when errors are propagated to the outputs of the system. The system fails due to insufficient QoS instead when the QoS is below a certain threshold (QoS_{thr}). For instance, in a system providing edge detection for images, the QoS is defined in [69] as the peak Signal-to-Noise Ratio (SNR) when comparing the corrupted and correct images and the QoS_{thr} is set to 10 dB. More complex services have a more complex definition of acceptable quality. For instance, in Moving Picture Experts Group (MPEG) encoding there are three types of frames: *I* frames,

[9] A dynamically dead instruction is an instruction which outputs are not used by any other instructions and that does not actually influence the output of the processor [62].

[10] In [63], masked cases are instead classified in two different categories depending on whether the final architectural status is different from a fault-free execution (referred to as *Output Not Affected*) and those where it is the same (referred to as *Vanished*).

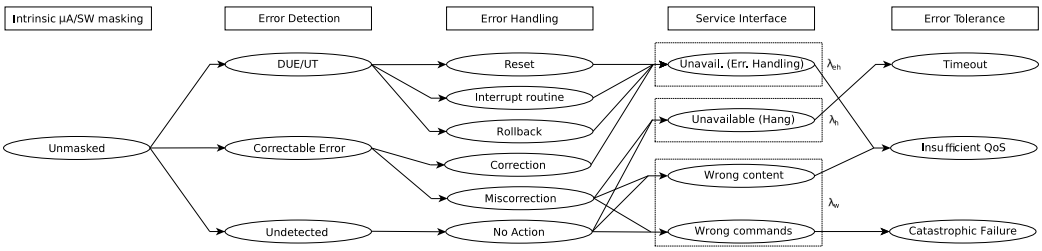

Fig. 3. Propagation of errors to the service interface and effects on the system service.

P frames and *B* frames [69]. In general, the loss of *B* and *P* frames can be compensated by the decoder, while the loss of an *I* frame will result in a substantial quality degradation. In [69] a frame is considered bad if the SNR (compared to the correct frame) is more than 2 dB for *I* frames, 4 dB for *P* frames and 6 dB for *B* frames. The QoS in [69] is then defined as the percentage of good frames and QoS_{thr} is then set to 10% of bad frames. An example of even more complex service is inference for image classification. In this case the QoS_{thr} is defined as the difference in confidence of the top ranked element compared to the top ranked element of the fault-free execution [70]. In addition, the concept of QoS is introduced also for the catastrophic failures, which in this case is when the top ranked element differs from the golden execution. As a matter of fact, a differentiation is done between the case where the top ranked element is at least a 'good candidate' (i.e. one of the first 5) in the fault-free execution and the opposite case.

In [69] it is shown that in order to fully exploit the concept of error-tolerance, control operations (defined as those which can change the control flow in the software and therefore potentially generate wrong commands at the outputs) must be identified and protected. As a matter of fact, catastrophic failures are avoided both for *Susan* (edge detection) and *MPEG* (MPEG encoding) when errors are not injected in control operations (while some other benchmarks have catastrophic failure rates up to 19% even when errors are not injected in control operations). When control operations are protected, more than 100 errors per second had to be injected in *Susan* to show any frame loss due to the SNR being too low. *MPEG* had instead about 2% loss at 10 errors per second. Both error rates are pessimistic for space, as the error rate in this case is several order of magnitude lower (in Section 3.2 the maximum SER found is around three errors per day at the highest upset rate considered). *MPEG* crashes disabling protection for control operation, while for *Susan* disabling protection leads to very poor fidelity of output. This can be attributed to the relatively small number of control instructions (less than 9%) in *Susan* compared to the higher percentage in *MPEG* (around 50%) [69].

2.3.2. Explicit error tolerance

Once models of failures at the service interface are defined, explicit techniques of error tolerance can be employed. One of the most commonly used is the watchdog timer, namely a counter that if not periodically reset by the processor will reset the processor itself [71]. This is represented in Fig. 3 with *Timeout* and it is based on the simple model of *Hang* of the processor at the service interface. However more complex models can be employed, and in [71] also a smart watchdog is proposed. Similarly, in [72] a symptom-based mechanism is employed to reduce the failure rate by 20x over a baseline design without explicit error tolerance.

3. Modeling the vulnerability of processors

Once the models for the threats are defined, the following step is to build a model to identify the most vulnerable parts of the design. A common model in literature is the Architectural Vulnerability Factor (AVF) decomposition [41].

3.1. AVF decomposition

In order to take into account the masking effects due to software and microarchitecture, in [73] the AVF of a unit is defined as the probability that a fault in that unit of the processor will cause a failure at the outputs of the processor. For this reason, the AVF depends on which event of those described in Section 2.3 are considered as failures. In this work, we will use the definitions of failures as indicated in Fig. 3 (at the service interface).

The rate of occurrence of a failure f for the unit i can be modeled as $\lambda_{i,f} = SER_i \times AVF_{i,f}$. In order to have a correct execution, all the units of the AVF decomposition are required to not propagate an error to the outputs of the processor. As a result, units in an AVF decomposition can be thought as a series of components in a reliability block diagram [41]. Assuming that the masking is uniform (therefore not changing the distribution of events) and assuming that failures in different components are independent of each other, the total reliability is given by the product of the reliability of the units composing the processor. The processor-level failure rate for the failure f λ_f is then given by:

$$\lambda_f = \sum_i SER_i \times AVF_{i,f} = SER \times AVF_f \tag{6}$$

As $SER = \lambda_{ev} \times N_{eq}$, the effects of failures on a service for space applications (relatively high λ_{ev} and low N_{eq}) can be sometimes compared to the effects on services for application with lower λ_{ev} and higher N_{eq} (e.g. servers) [41]. Eq. (6) can also be written as $\lambda_f = \lambda_{ev} \sum_i \hat{\lambda}_{i,f}$, where $\hat{\lambda}_{i,f} = N_i \times AVF_{i,f}$ is the failure rate normalized to the upset rate per bit. For failures causing wrong outputs or data, the failure rate λ_w (Fig. 3) is enough to estimate their effect on the service.[11]

The impact on the service interface of failures causing unavailability[12] instead is also determined by the duration of the unavailability $T_{u,i}$ they cause each time they manifest. Different types of events causing unavailability can be observed:

- Timeout (λ_h): these events are due to residual AVF_u not protected by redundancy. We assume they are addressed employing a watchdog timer that triggers a hard reset (power cycle) when it expires. An order of magnitude for $T_{u,h}$ can be found in [74], where it is assumed to last 5 min, as extensive checking (e.g. memory) is required.
- UT ($\lambda_{eh,ut}$): when a process is terminated, a possible solution is to use an interrupt service routine for diagnostic and restart of the process. These have typically lower impact than a reset. The work in [75] shows that a process can be restarted with a latency on the order of 10 ms.

[11] Sometimes, instead of the failure rate, the Mean Time To Failure (MTTF) is employed to indicate how often a failure will happen on average. The use of an exponential reliability function simplifies further the calculations, as $MTTF_w = 1/\lambda_w$.

[12] If a system is unavailable for a total $T_{Unavailable}$ during a certain $T_{Mission}$, the unavailability is then defined as $U = \frac{T_{Unavailable}}{T_{Mission}}$ and the availability as $Availability = 1 - U$.

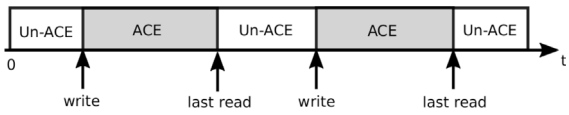

Fig. 4. Fraction of time a location in the RF is in ACE (gray) and un-ACE (white). Between *write* and *last read* an arbitrary number of reads can happen.

- DUE in data without valid copies ($\lambda_{eh,hr}$): in this case, e.g. errors in Write-Back (WB) caches, a DUE requires at least a soft reset (i.e. ending the current processes and booting again). From the work in [76], a penalty of 45 s can be assumed for a soft reset, composed of end time and boot time.
- Rollback to an up-do-date value ($\lambda_{eh,rb}$): when the corrupted data is available in the most up-to-date value, the loss in terms of availability is minimal. For instance, in case of a DUE in a Level 1 (L1) cache with Write-Through (WT) policy the data can be read from the Level 2 Cache (L2C), with a penalty of a cache miss [77]. As can be seen in [77], 150 Clock Cycles (CCs) can be taken as a pessimistic estimation for a cache miss and even in this case, assuming a clock frequency of 100 MHz, the penalty is in the order of microseconds (which is in most cases negligible).
- Correction ($\lambda_{eh,c}$): the latency in this case is very short. For instance, the LEON2FT checks the EDAC code on the Register File (RF) during the execution phase, writes back errors in the RF with the correct value, flushes the pipeline and restarts from the instruction that reads the operand with the error [78]. This procedure causes typically minimum penalty in terms of stalling (in this case just 5 CCs).
- Device-specific rollback ($\lambda_{eh,ds}$): some devices save the old status to rollback to it in case of DUE [79] or they compare the output of three processors and restore the correct status from one of the golden replica [20]. In these cases the penalty in terms of availability is implementation-specific. We will discuss this aspect further in Section 4.

The unavailability due to each type of these events i can be expressed as:

$$U_i = \frac{N_{u,i} \times T_{u,i}}{T_{Mission}} = \frac{(T_{Mission} \times \lambda_{u,i}) \times T_{u,i}}{T_{Mission}} \tag{7}$$

where $N_{u,i}$ is the number of times the events i happened during the mission and $T_{Mission}$ is the total mission time. Therefore, the unavailability of the processor considering all the possible sources i of unavailability is:

$$U = T_{u,h} \times \lambda_h + \sum_i T_{u,eh,i} \times \lambda_{eh,i} = \lambda_{ev} \times \hat{U} \tag{8}$$

3.1.1. Vulnerability in time: ACE analysis

More insights can be gained on the meaning of the AVF by considering how AVF is estimated in [73], i.e. considering the bits required for an Architecturally Correct Execution (ACE). A bit is an ACE-bit when changing its value will cause the error to propagate to the service interface and it is an un-ACE bit otherwise. A bit typically changes from ACE to un-ACE and vice versa during program execution, as shown in Fig. 4 for a bit in a location of the RF.

At any instant in time, the AVF can be expressed as the number of ACE bits in a structure N_{ACE_i} over the total number of bits in the structure N_i: $AVF_i(t) = \frac{N_{ACE_i}(t)}{N_i}$. The average AVF can then be defined as the average number of ACE-bits in a certain timespan. Using Little's law [73], the average number of ACE-bits within a structure (e.g. instruction buffer or execution unit) can be written as the product of the arrival rate (bandwidth B_{ACE_i}) of ACE bits and the average time of persistence in the structure (latency L_i):

$$AVF_i = \frac{N_{ACE_i}}{N_i} = \frac{B_{ACE_i} \times L_i}{N_i} \tag{9}$$

For instance, when considering hardware structures storing or executing instructions, the rate of arrival of ACE bits is given by the number of Instructions Per (clock) Cycle (IPC). The average time these bits spend in the structure depends on the functionality of the block, which may store it for a long time (e.g. memory or buffers) leading to high AVFs or for shorter times (execution units) leading to lower AVF. Furthermore, for functional units like Arithmetic Logic Units (ALU), Eq. (9) shows that the more frequently they are used and the longer is the latency of the operation, the more vulnerable they are. For memories, it shows that the longer the average lifetime and the higher the memory utilization, the higher the AVF is.

Table 2
Features of the cache subsystem common to LE and HE (data derived from [82]). 'Pref.' stands for 'prefetcher'.

Unit	Size	Block size	Associativity	Policy	Prefetching
DC	32 KiB	64B	4-way	WB	Stride pref.
IC	32 KiB	64B	4-way	Read-only	Pref.
L2C	1 MiB	64B	16-way	WB	w/o pref.

3.2. Impact of the microarchitecture on the failure rate

In [7] the authors provided an overview on RISC-V and proposed how to employ the RISC-V ISA in space data systems to address present and future needs. In this roadmap, several 'profiles' of processors were proposed. Here we will analyze four General Purpose (GP) profiles from the point of view of dependability as case studies for our models: GP-LE-1, GP-LE-4, GP-HE-1 and GP-HE-4.[13] The LE-4 can be seen as an implementation equivalent to the state of the art of space-grade components (single-issue, in-order pipeline, quad-core like the GR740 [2]), while the HE-4 can be seen as a possible future space-grade processor. These configuration will be represented by the Rocket (LE) and the BOOM processor (HE) where FI was carried out in [67]. Therefore, for units in Tables 3 and 4 we use values for AVFs from [67]. However, to provide a more comprehensive comparison of the contribution of each block in a realistic design, we also include estimations for one L1 Instruction Cache (IC) per core, one Data Cache (DC) per core, one FPU per core and L2C (one shared among the cores in LE-4 and HE-4). For the Floating Register File (FRF) we use as a pessimistic estimation the same value of the Integer Register File (IRF) of the Rocket, as data from [80] shows for FRF similar contribution to the failure rate compared to the IRF. When considering the functional part of the Floating Point Unit (FPU), [81] shows that in average (over different benchmarks) only 1.76% of errors in FPUs reach the FPU output.[14] For all the profiles we use the same cache configuration, i.e. the baseline of [82] that is reported in Table 2 and

[13] As defined in [7], "LE" stands for Low-End and "HE" stands for High-End. The following digit indicates the number of cores. In the remainder of this paper, "GP" is usually omitted as only GP processors are considered.

[14] Further data shows that AVF for control modules in the FPU is 8.9% while datapath modules have a 1.43%. The large percentage of area dedicated to the datapath in a FPU explains the low average value. Also, this is a pessimistic estimation for the AVF of a FPU in a processor as the service interface is taken at the output of the FPU and not at the output of the processor, thus neglecting the masking effect of the rest of the processor to errors coming from the FPU. These data do not differentiate between types of failure so we assume that the breakdown is similar to the one of the Arithmetic-Logic Unit (ALU) in the HE-1 in terms of AVF_w, AVF_h and $AVF_{eh,ut}$.

Table 3
AVF (from [67,80,81]) and N_{eq} for LE-1 (without caches), decomposed in IRF, Multiplier and Divider (M/D), Instruction Buffer (IB), rest of the Integer Unit (IU), Control and Status Registers (CSR), FRF and FPU.

LE-1	IRF	M/D	IB	IU	CSR	FRF	FPU
AVF_w	3.3%	0.2%	0.5%	2.4%	5.9%	3.3%	1.0%
AVF_h	1.0%	0.1%	0.3%	4.4%	8.2%	1.0%	0.2%
$AVF_{eh,ut}$	12.2%	0.4%	1.1%	4.9%	4.3%	12.2%	0.6%
$N_{eq,LC}$	2.65E+3	2.17E+2	9.9E+1	1.1E+3	1.2E+3	2.8E+3	1.6E+3
$N_{eq,AC}$	2.65E+3	5.72E+2	1.4E+2	1.7E+3	1.4E+3	2.8E+3	5.0E+3
$N_{eq,HC}$	2.65E+3	9.27E+2	1.8E+2	2.2E+3	1.6E+3	2.8E+3	8.5E+3

Table 4
AVF (from [67,80,81]) and N_{eq} for HE-1 (without caches), decomposed in IRF, Register Rename (RR), Instruction Fetch (IF), Instruction Issue (II), Load and Store Unit (LSU), ReOrder Buffer (ROB), BP, ALU, CSR, FRF and FPU.

HE-1	IRF	RR	IF	II	LSU	ROB	BP	ALU	CSR	FRF	FPU
AVF_w	1.9%	2.4%	2.6%	2.4%	1.5%	1.2%	0.8%	1.2%	3.9%	3.3%	1.0%
AVF_h	1.0%	3.3%	1.0%	3.1%	2.4%	2.4%	1.5%	0.4%	0.2%	1.0%	0.2%
$AVF_{eh,ut}$	8.7%	5.7%	7.3%	0.9%	3.7%	0.8%	0.1%	0.7%	5.4%	8.7%	0.6%
$N_{eq,LC}$	4.5E+3	2.9E+3	4.1E+3	7.1E+2	2.1E+3	1.1E+3	2.8E+3	1.9E+3	1.3E+3	3.4E+3	4.8E+3
$N_{eq,AC}$	6.4E+3	4.1E+3	5.6E+3	9.8E+2	2.6E+3	1.2E+3	3.0E+3	3.7E+3	1.5E+3	4.3E+3	7.5E+3
$N_{eq,HC}$	8.4E+3	5.2E+3	7.1E+3	1.2E+3	3.1E+3	1.4E+3	3.1E+3	5.5E+3	1.8E+3	5.1E+3	1.0E+4

Table 5
AVF (from [82]) and N_{eq} (the same for all technologies) for caches. LE-1 and HE-1 have one DC and one IC each. LE-4 and HE-4 are obtained replicating 4 times the respective single-core version and adding a L2C.

Caches	DC_{WT}	DC_{WB}	IC	$L2C_{WB}$
AVF_w	5%	8.8%	0.5%	0.5%
AVF_h	1.3%	2.5%	5%	0.6%
$AVF_{eh,ut}$	2.9%	4.3%	5.2%	1.7%
N_{eq}	5.14E+4	5.7E+2	2.0E+5	2.4E+6

with AVF values reported in Table 5. This will provide the reader with an estimation of how the same size of caches influences the failure rate in different designs (even if higher performance processors may employ larger caches). However, in Section 3.2.1 we will also provide models and considerations on scaling of cache size. For simplicity, in this section we will consider only data arrays and not tag arrays in caches. Even if tag arrays in [83] are reported to be have higher AVF than data arrays[15] (as for instance they have on average an AVF 2.76x higher than data arrays in DC), they typically are smaller (around 7 KiB, i.e. around 9 times smaller than the data array). Therefore, not including tag bits in the model can be expected to underestimate the vulnerability of caches by around 20% according to Eq. (6). Furthermore, using values for caches of a processor with a different ISA does not impact AVF of caches in a significant way, as in [84] the AVF of caches for two different ISAs (ARM and x86) for 10 MiBench benchmarks shows that the difference is small.[16]

Furthermore, we assume same average values of AVF for single and quad-core versions of the same design. As a matter of fact, [85] investigates the changes in AVF in a dual-core processor where each core is running a different thread and it shows that AVF is roughly the same compared to a single core (the change in AVF is within a $+/-2\%$ of the AVF single core value).

Estimations of N_{eq} are obtained with syntheses on Design Compiler on a 65 nm bulk commercial technology targeting 100 MHz and using the code available to the public of the Rocket processor[17] and of the BOOM processor.[18] However, as we do not have access to the memory compiler of the ASIC technology (as it is often the case), we will estimate the size of caches using CACTI [86].

It can be noted from Figs. 5 and 6 that caches are the most vulnerable units in processors, even considering technologies with high SER from combinational logic. This was already shown in [87] with a less refined model. Most of the units have a similar relative contribution to λ_w and U, except the IC which has a similar impact compared to L2C in terms of unavailability but lags behind more than a order of magnitude in terms of λ_w. Most of the units increase their failure rate when moving from LC to SD. However, for a few of those (those with higher percentage of sequential elements like BP), the failure rate decreases due to FF temporal masking (as shown in [16]). Furthermore, microarchitectures impact the failure rate much more in terms of N_{eq} than in terms of AVF. As a matter of fact, the maximum ratio between two different designs in terms of N_{eq} with the same type of technology defined in this section (cacheless LE-1 and the HE-4) is around 100 for each technology, while the maximum ratio of AVFs found in literature due to different microarchitectures is around 4x (in [88]).

3.2.1. Design explorations

In [89] the effect of the processor width and of the number of functional units (e.g ALU and FPU) on the AVF of the functional units is investigated but no clear correlation is found. Looking at data from the literature for IRF and caches (e.g. [82]), we define two models of scaling of the failure rate for an array of sequential elements based on Eq. (9), as shown in Fig. 7:

- Constant Workload (CW): the workload for the array remains constant while increasing the size of the unit, meaning that the failure rate remains constant and the AVF decreases by the same factor as the size was increased.
- Constant Utilization (CU): the relative utilization of the array remains constant while increasing the size of the array, meaning that the AVF remains the same and the failure rate increases of the same amount the size was increased.

As shown in Fig. 7 some units show a behavior similar to CW (e.g. physical register), some lay in between CU and CW (DC on

15 Also [19] shows a high value for tag arrays (32.5%).

16 Intuitively, this is more true for L2C (−4%) and DC (+5%), while the difference is slightly larger for ICs (+24%), which store ISA-specific instructions [84].

17 https://github.com/chipsalliance/rocket-chip.git.

18 https://github.com/riscv-boom/boom-template.git.

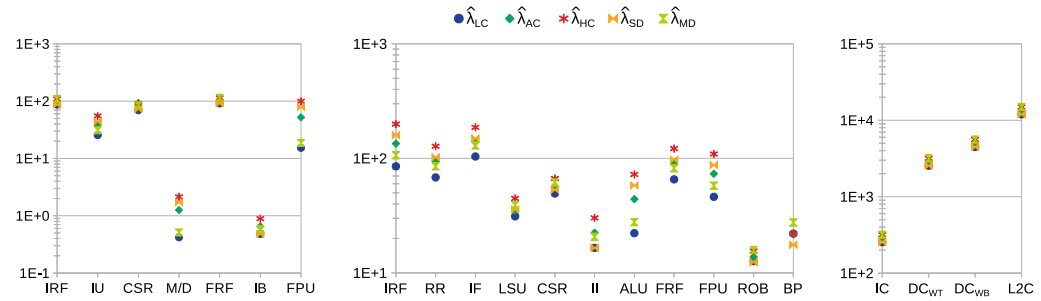

Fig. 5. Normalized failure rate for wrong outputs $\hat{\lambda}_w$ for LE-1 (cacheless), HE-1 (cacheless) and caches (from left to right). Calculations based on Eq. (6).

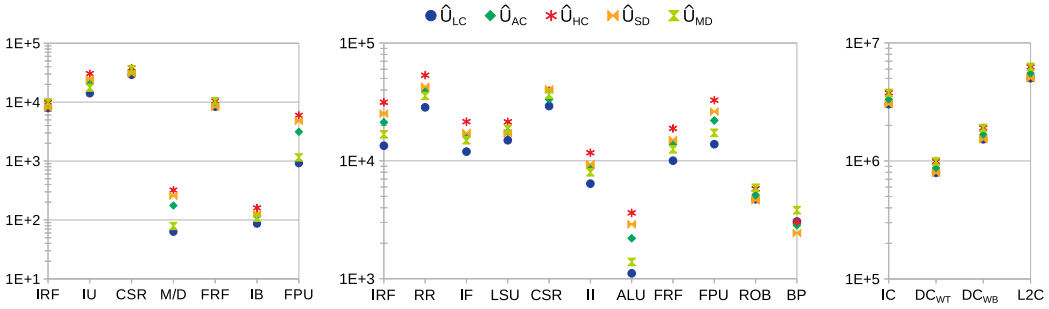

Fig. 6. Normalized unavailability $\hat{U}$ for LE-1 (cacheless), HE-1 (cacheless) and caches (from left to right). Calculations based on Eq. (8).

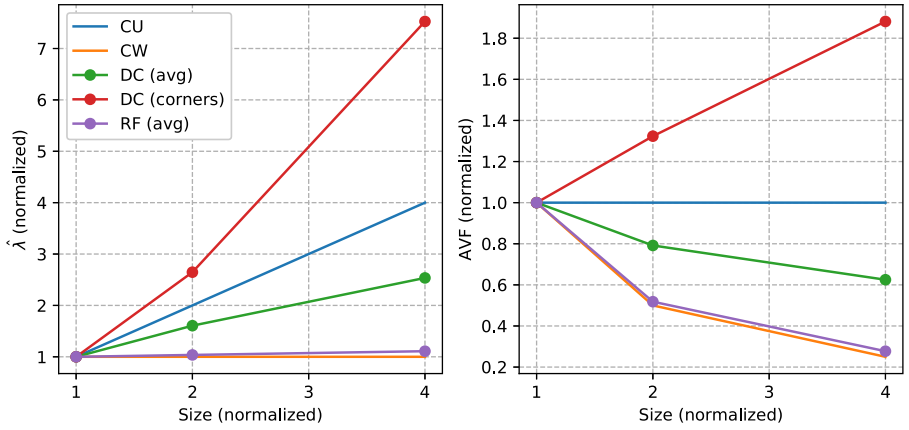

Fig. 7. Effects of size on $\hat{\lambda}$ (normalized to the $\hat{\lambda}$ of the smallest size considered) and AVF for 2x and 4x increases (based on [82]).

average and IC for all benchmarks from [82]) and some other units increase their utilization when their size is increased (DC for the *corners* benchmark in [82]) and in this case we talk about "superlinear" behavior (as done in [90]).

The results in [91] confirm the increase of failure rate of the DC when increasing its size. However, in this case the behavior shown is superlinear (and not in between CW and CU), as increasing its size of 16x (from 16 KiB to 256 KiB) increases its failure rate by 21x. Interestingly, they also show that increasing the size of DC by 16x has an effect on the failure rate of L2, which decreases by around 2x. The work in [92] highlights how cache arrays typically exhibit a superlinear behavior when the cache hit rate increases with the increase of the size (e.g. for the *FFT* and *matrix multiplication* benchmarks), while if the cache hit rate remain constant they typically show a CW behavior. An explanation for this is presented in [90] and reported in Fig. 8 (left). Let us consider a program that reads the variable *A*, then the variable *B* and then again the variable *A*. In a large cache, it is more likely that both A and B will reside in the cache. For this reason reading B does not cause a cache miss and line *A* is not evicted. In a small cache instead, reading *B* is more likely to cause a cache miss and a replacement of *A* with *B*, thus reducing drastically the fraction of time the location stores ACE-bits. The mechanism described happens for both WT and WB policies, while in Fig. 8 (right) it is also shown a mechanism specific of WB caches. As a matter of fact, in WB caches dirty lines also exist and those are always ACE, as they will be eventually written back to main memory. Fig. 8 (right) shows a program which writes *A* and then reads *B* and then does not act on the location until the end of the program, when the dirty lines will be written back. Also in this case, a small cache which substitute *A* with *B* can reduce the fraction of time the location stores ACE-bits considerably.

The previous discussion shows also that the write policy influences the AVF of the L2C: in [82] a value of 7% can be taken for a WB L2 cache (in [84] a similar value is given) and 4.2% for a WT L2C (1 MiB), which implies almost double the SER due to the L2C.

Furthermore, as show in [82], the AVF of the DC is roughly insensitive to the associativity (5 benchmarks out of 8), while some benchmarks (*djpeg* and *smooth*) exhibit a steep variation

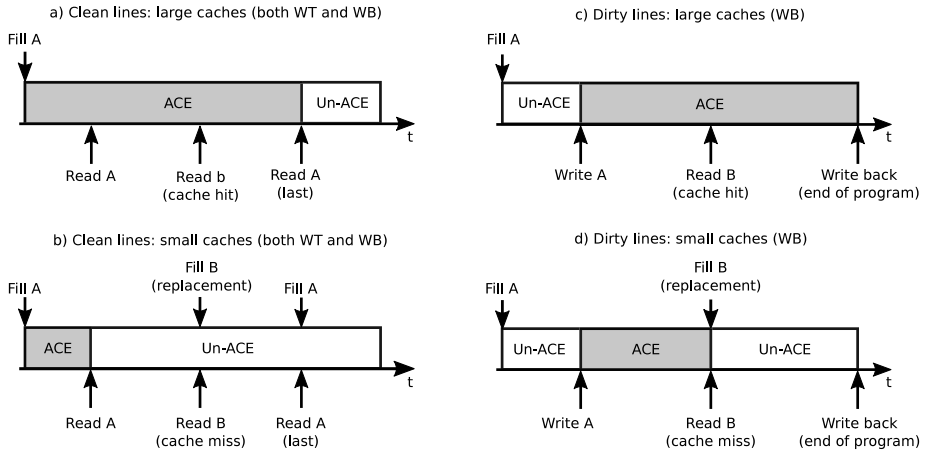

Fig. 8. Examples of superlinear behavior for a location in DC storing the variable *A*, similarly to [90].

of AVF for a specific number of ways, and only one (*search*) shows an increase of AVF with the number of ways. IC instead decreases its AVF when the number of ways is increased [82]. Adding prefetches to the DC leaves substantially unchanged the AVF, while removing prefetchers for IC reduces the AVF, which becomes, on average, 0.67x the baseline AVF [82].

3.3. Impact of other factors on the failure rate

Several factors impact the failure rate. Fig. 9 summarizes these factors indicating how large is the maximum value compared to the minimum value found of the failure rate when varying a certain factor. The impact of technology (and of the environment) and of microarchitecture was already assessed in Sections 2.1 and 3.2 respectively. The remainder of this subsection quantifies the impacts of other factors.

3.3.1. Dependence on performance and compiler flags

The work in [93] observes a fuzzy correlation between AVF and each performance metric considered (i.e. IPC, branch prediction rate and several cache miss rates) across several SPEC2000 benchmarks. However, in [94] the use of performance throttling is proposed to lower the overall AVF of a processor. Acting both on pipeline resources and cache miss rate, a failure rate reduction up to 35% is achieved.

Another way to change performance is to employ specific compiler flags. In [95], GCC with several combinations of optimization flags for the MiBench benchmark suite are compared in terms of performance and AVF. It is shown that the optimal set of flags for AVF decreases the AVF by around 9% compared to -O3 and of 8% compared to -O2. Among the optimization levels, in [96] -Os is found to be better than the -O0 for around 75% of the benchmarks (on a total of 25 benchmarks considered), while the lowest AVF in average is obtained with -02. Recent work [97] shows that the ratio of the AVF obtained with the worst and best sets of optimization flags is around 2x.

3.3.2. Dependence on software

Error masking in a processor, like performance, depends on the software employed. For this reason, it is crucial that the set of programs employed during the estimation of the AVF is representative of the final application or is general enough to represent a wide spectrum of applications. Most works in literature use the SPEC benchmark suites [67,88], others use EEMBC suites [80] and others MiBench [82,95] for its similarities to the SPEC benchmarks in terms for instance of instruction mix. As a matter of fact, instruction mix of the software can have a significant impact on the failure rate of certain units of the processor. For instance, data from [98] show that moving from benchmarks with low fraction of FPU instructions like *AMG2006* and *UMT* (0.03 and 0.1 per CC) to those with high fraction of FPU instructions like *LINPACK* (0.64 per CC) increases the failure rate of the FRF by 50x.

The variation of AVF on a microarchitecture employing different programs depends also on the microarchitecture itself. For instance, the work in [88] shows how, while for an in-order "small core", the AVF ranges from 8% to 16% (2x maximum increase) for the benchmarks of the SPEC CPU2006, for an Out-of-Order (OoO) "big core" it ranges from around 8% to around 29% (3.63x). Furthermore, the OoO core has for every benchmark a higher AVF compared to the in-order processor (except for *gobmk*). Ref. [88] also shows the Cycles Per Instruction (CPI) stack[19] for each benchmark and notices that there is no simple rule to predict whether a workload has high or low AVF.[20] According to [88], the benchmarks with low AVF have low vulnerability because of their high number of branch mispredictions and instruction cache misses. The benchmarks with high AVF show instead a more complex behavior. Some benchmarks (e.g. *milc*) are memory-intensive: a load operation accessing main memory typically blocks the head of the ROB, which causes the ROB to fill up. This leads to a significant increase of ACE bits while servicing the memory operation.[21] However, some memory-intensive benchmarks (e.g., *mcf* and *libquantum*) have low AVF because of branch mispredictions. Other high-AVF benchmarks (e.g. *zeusmp*) are compute-intensive: high IPC and high Memory Level Parallelism (MLP)[22] is achieved by having high occupancy in various queues. Some benchmarks with high AVF instead experience resource stalls because of DC misses, L2C misses, limited Instruction Level Parallelism (i.e., chains of dependent instructions) which cause the ROB and issue queues to fill up with instructions. Data in [67] show a different trend. In this case, the AVF values of the OoO core are smaller than those of the in-order core for every benchmark, and the trend is also true for the only two benchmarks in common

[19] A CPI stack quantifies the fraction of cycles spent doing useful work, 'lost' cycles because of resource stalls, branch mispredictions, instruction cache misses, Last-Level Cache (LLC) misses and main memory accesses [88].

[20] However the ACE states of caches are not evaluated in this case, as caches are assumed to be protected.

[21] This mechanism is only relevant to OoO processors and does not happen in in-order processors. This explain why the ranges are different.

[22] MLP is the average number of useful long-latency off-chip accesses outstanding when there is at least one such access outstanding [99]. Also in this case, this is a mechanism typical of OoO processors, as simulation results in [99] show that a moderately aggressive OoO issue processor improves MLP over an in-order issue processor by 12%–30%.

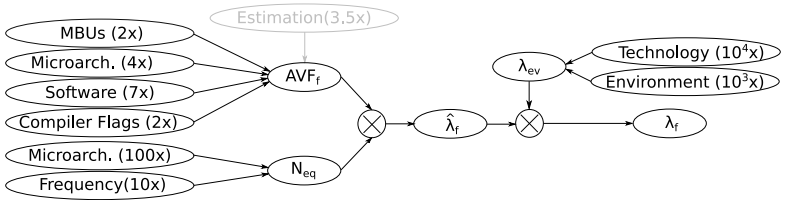

Fig. 9. Factors impacting the failure rate and their relative impact. *Estimation* is in gray because it does not affect the failure rate on the field.

with [88] (*bzip* and *gcc*). Also, the increase in ratio between the minimum and the maximum AVF found is much lower: from 12.4% to 20.6% for the in-order (1.66x) against 7% to 12.3% (1.76x) for the OoO processor. When considering caches, in [82] the AVF for the baseline DC ranges from around 3% to around 23% (7.66x).

3.3.3. Dependence on the fraction of MBUs

In [8] more complex error models compared to the Single Bit Flip (SBF) of [67] are employed to investigate the effects of MBUs on the AVF. The most relevant result is that the AVF value saturates on average around 3 upsets per strike, with an increase of around 10% compared to the value found with the SBF model on average and with a peak of around 25%. To take into account this effect, the AVF employed in Eq. (6) will be then $\overline{AVF} = \alpha_{MU} \times AVF$. We will employ $\alpha_{MU} = 1$ for technologies with a low fraction of MBUs (LC and SD), $\alpha_{MU} = 1.1$ for technologies with an average fraction of MBUs (AC) and $\alpha_{MU} = 1.25$ for technologies with a high fraction of MBUs (HC and MD). The impact of MBUs on the failure rate is limited, as the ratio between the minimum and the maximum value of AVF changing the number of upsets per strike reported in [8] is around 2x. Also in [98], the maximum ratio found when injecting one and four upsets is 2x.

3.3.4. Uncertainty due to the estimation method employed

AVF was originally defined with an ACE analysis implemented on a microarchitectural simulator [73]. In [100], the ACE approach is found to underestimate on average fault masking about 250% compared to FI. The causes identified for such overestimations are: the limited information on the bits (when it cannot be determined whether a bit is in a ACE or un-ACE state, it is assumed in ACE-sate to prove that requirements can be met); limited size of time windows to analyze dead instructions; and Y-bits.[23] The conclusion in [100] is that ACE analysis can be refined until a theoretical threshold, after which is not possible to reduce conservatism further because of Y-bits. However, before this theoretical limit for ACE analysis is reached, ACE analysis becomes intractable due to the increase of complexity. The authors of [102] reject this point of view, arguing that while FI on RTL may provide a more accurate AVF by modeling all low-level masking effects, much of this can be accounted for at the performance level by identifying and modeling those masking effects that significantly impact the AVF and that the Y-bit effect is on the order of 14% on the AVF and that it can be addressed with a more refined ACE analysis [100]. While most of the extended microarchitectural simulators are not available to the public, a modified version of the gem5 simulator capable of assessing soft error vulnerability [103] is available.[24]

FI requires a large number of experiments and either a working hardware platform or a RTL model that can be simulated. However, the results in [104], regarding a dual-core processor design consisting of around 350k sequential elements, show that randomly selecting more than 2.85% elements (10k) for injections provides only marginal improvements in terms of reduction of uncertainty (the standard deviation when considering 10 different groups of FFs saturates).

To inject errors also in microarchitectural resources of a hardware prototype, hardware support is needed. For instance, in [67] faults are injected in a FPGA prototype with an extra XOR at the input of each FF of the processor. The host processor within the FPGA decides which FF to inject (and at which CC) and sends the command to the fault injector connected through a crossbar. Without a similar hardware support, errors could be injected only via software in architectural resources. Another possibility is to simulate an RTL model and inject errors during the simulation, changing the value of a specific signal [105].

3.4. Limitations of the AVF decomposition

Although the use of the AVF decomposition as introduced in Section 3.1 is common [41], there are some limitations in its capability of assessing the vulnerability of processor units.

3.4.1. Sub-unit vulnerability

The AVF decomposition does not provide insights on the homogeneity in terms of vulnerability of a certain structure. The work in [80] provides instead also data for sub-unit vulnerability. In this case, the sequential elements of each unit are grouped in Criticality Levels (CLs), depending on the percentage of times an error in that sequential element propagates to the outputs. For instance, CL0 means that a fault in that element never causes a failure and CL5 indicates that a fault in that element always causes a failure. This classification may allow selecting redundancy more efficiently. For instance, a memory array with a large fraction of CL0 sequential elements can be protected more efficiently with selective information redundancy [106,107] or partial hardware redundancy [108]. While [80] adopts a conservative approach that defines a FF critical if it is critical at least for a benchmark, data from [16] suggest that a considerable part of the critical FFs remains the same among 8 workloads from MiBench (85 out of the top 200 vulnerable FFs of each benchmark), and only a minority (around 13% for each benchmark) are critical in only a single benchmark. For instance, [106] notes that only a few "long-lived" registers (10% for the IRF) are responsible for 40% of the total vulnerability time of the IRF. Based on this consideration, it is proposed to use a cache smaller than the RF to store the ECC of physical registers and replace check bits of "short-lived" registers with those of the "long-lived" ones.

3.4.2. Propagation to specific signals at the service interface

The AVF decomposition does not take into account to which signal of the service interface the errors will propagate. In [80] it is found that errors manifest only in 65% of the outputs, with almost 80% of these errors manifesting in only 20% of the ports.

[23] Y-Bits are bits that can alter the course of execution in the processor without causing a failure, for instance branches for which the behavior of the application is unaffected by whether the particular branch instance is taken or not. Around 40% of the executed branches in SPECint2000 are Y-branch [101].

[24] https://github.com/MPSLab-ASU/gemV.

3.4.3. Propagation time

The AVF decomposition also does not take into account how long it takes for the propagation of the error to the service interface. Data from [80] show that all of the processor components (without considering caches) have a minimum error manifestation time equal or less than 7 CCs, whereas the average error manifestation time for an error in the processor is 1204 CCs. The worst propagation time is 153,287 CCs (for the logic responsible for branch prediction). Errors propagate more quickly when they directly affect the processing data (e.g. ALU and FPU), while storage units like RFs have instead longer propagation times [80]. FRF has longer average propagation times (2950 CCs) compared to IRF (370 CCs), mainly because of more matrix operations and longer latency of the FPU compared to the ALU. The exception are some long-life integer variables, such as indexes in iterative loops (propagation time on the order of the thousands of CCs) [80].

3.4.4. Error accumulation

The AVF decomposition assumes that the software is composed of program loops of period T_L each [41]. In order for the AVF decomposition to produce a negligible error, the product $T_L \times \lambda$ must be small [41]. This means that AVF decomposition is valid when a small number of soft errors occur in a loop iteration. In [109] a model to overcome this limitation is proposed, however it is much more complex. Nevertheless, for each unit a failure rate $\lambda_i = DF \times SER_i$ can be associated, where DF is a more general derating factor. Therefore, the final result of such more detailed model is a failure rate for each unit in the design like those in Figs. 5 and 6, on which the same procedures to apply and validate redundancy can be followed like done in Sections 4 and 5.1.

4. Applying cost-effective redundancy

Given the possible large overhead of redundancy, the concept of cost-effectiveness is introduced in [106] (where a proposed technique is compared to others in terms of area, power and performance overhead) and in [108] (where area and power overhead are considered). To provide a metric for this concept, we define the following cost function:

$$C = \alpha \frac{\Delta T_{ex}}{T_{ex}} + \beta \frac{\Delta A}{A} + \gamma \frac{\Delta P}{P} + \delta \frac{\Delta \lambda_w}{\lambda_w} + \epsilon \frac{\Delta U}{U} \tag{10}$$

where α, β, γ, δ, and ϵ are arbitrary weights depending on the target of the design. We will show how weights can affect the optimal choice for two opposite cases:

1. Focus on dependability (C_d): $\alpha = 0.25$, $\beta = 0.5$, $\gamma = 0.5$, $\delta = 1$, $\epsilon = 1$. This can be seen as the case of an OBC for command and control operations.
2. Focus on performance (C_p): $\alpha = 1$, $\beta = 0.5$, $\gamma = 0.5$, $\delta = 0.25$, $\epsilon = 0.5$. This can be seen as the case of a payload processor for high-performance on-board processing.

T_{ex} is the execution time for a (set of) program(s) employed to evaluate performance or the execution of a certain task. We will use a linear model where $T_{ex} = T_{clk} \times CPI \times N_I$, where T_{clk} is the clock period, CPI the number of CCs per instruction and N_I the number of instructions in a program.[25] However, a decrease in T_{clk} may be partially compensated by an increase in CPI due to the fact that memories are typically slower than processors and for this reason the penalty is less than proportional to the loss in T_{clk}. Furthermore, we do not include latencies in case of DUEs or corrections in the loss of performance, as they are not frequent enough to cause degradation in performance (as opposed to increase of latency even when there are no errors). The terms $\frac{\Delta A}{A}$ and $\frac{\Delta P}{P}$ indicate respectively the relative increase in terms of area and power of the whole processor (keeping the same target and operative frequency). $\frac{\Delta \lambda_w}{\lambda_w}$ and $\frac{\Delta U}{U}$ account for the percentage of errors detected by the redundancy, i.e. its coverage. For instance, the $\Delta\hat{\lambda}_w$ of a certain redundancy technique can be found as:

$$\Delta\hat{\lambda}_w = -\sum_i AVF_{w,i} \left[N_{eq,SEU,i} \left(1 + \frac{\Delta N_{eq,SEU,i}}{N_{eq,SEU,i}}\right) P_{det,comb,i} + N_{eq,SET,i} \left(1 + \frac{\Delta N_{eq,SET,i}}{N_{eq,SET,i}}\right) P_{det,seq,i} \right] \tag{11}$$

where $P_{det,seq,i}$ is the probability of detecting an error in the unit i for sequential logic and $P_{det,comb,i}$ its analogous for combinational logic. Effective redundancy will have a negative $\frac{\Delta \lambda_w}{\lambda_w}$ and will decrease the cost function, but it is mathematically possible to have a positive $\frac{\Delta \lambda_w}{\lambda_w}$, when:

$$\left(1 + \frac{\Delta N_{eq,SEU}}{N_{eq,SEU}}\right) > \frac{1}{P_{det,seq,i}} \tag{12}$$

The case where unavailability increases is instead more common, because of the increase in unavailability from frequent error handling:

$$\Delta\hat{U} = T_{u,h} \times \Delta\hat{\lambda}_h + \sum_j T_{u,eh,j} \times \Delta\hat{\lambda}_{eh,j} \tag{13}$$

where j is the index of the jth type of unavailability due to error handling, $\Delta\hat{\lambda}_h$ can be found with the same formula as Eq. (11) and $\Delta\hat{\lambda}_{eh,j}$ can be found as:

$$\Delta\hat{\lambda}_{eh,j} = \sum_i AVF'_{w,i} \left[N_{eq,SEU} \left(1 + \frac{\Delta N_{eq,SEU}}{N_{eq,SEU}}\right) P_{det,seq,i} + N_{eq,SET} \left(1 + \frac{\Delta N_{eq,SET}}{N_{eq,SET}}\right) P_{det,seq,i} \right] \tag{14}$$

where AVF'_i is the masking factor for all events considering as service interface the point where the redundancy can detect the error in the processor. This is needed because redundancy will react also to errors that manage to propagate to this point and that would be mask in the rest of the propagation to the real service interface if redundancy was not included. This implies that the rate of new error handling events $\Delta\hat{\lambda}_{eh} = \sum_j \hat{\lambda}_{eh,j}$ is larger than the decrease of the rate of other events $-\Delta\hat{\lambda} = -(\Delta\hat{\lambda}_w + \Delta\hat{\lambda}_h + \Delta\hat{\lambda}_{eh,ut})$. An example is given in Section 4.1.3, where $\Delta\hat{\lambda}_{eh}$ is larger than $-\Delta\hat{\lambda}$ by a factor ranging from 1.8x to 13.6x.

In the following subsections we will introduce several types of redundancy for different units of processors and we will evaluate their efficacy and cost-effectiveness for different designs and different technologies/environments. In order to show different types of redundancy, we will apply the cost function to each part of the processor, decomposed in:

1. Cache Arrays (Section 4.1)
2. Register Files (Section 4.2.1)
3. Mixed logic (Section 4.2.2), composed of the remaining combinational and sequential logic

For each of them the most cost-efficient redundancy for different designs, weights and technologies will be assessed. In Section 5 the total effect of applying the most cost-efficient to all the components of the processors will be analyzed. More complex optimization methods can be employed, as done in [25].

[25] It should be noted that increases in CPI are actually more expensive than increases in T_{clk} as the increase in CPI implies that the FT processor is not functionally equivalent to its COTS counterpart even when errors are not detected.

4.1. Choice of redundancy for cache arrays

Memory arrays are typically protected with information redundancy, i.e. information is stored with more bits than strictly required, employing EDAC codes [78]. EDAC codes can be classified according to their capabilities in terms of number of errors that can be detected and corrected in a single protected memory block,[26] which are determined by the minimum distance ('d', i.e. the minimum number of bits that differs) between two valid words of the code ('codewords') [110]. A binary (n, k) linear block code encodes words of k bits using $n = k + r$ bits, with r being the number of check bits [110]. Despite several codes with high correction and detection capability are proposed in literature (e.g. in [18] up to 8-error detection and 9-error correction), implementations typically employ Single Error Detection (SED) codes [78,111–113] or Single Error Correction and Double Error Detection (SECDED) codes [111–114], as in [18] it is shown how with the increase of the minimum distance within the codewords there is an exponential increase in overhead in terms of area and energy per access to the memory block.

SED detects all single errors in an EDAC-protected block [115]. This is often referred to as 'parity', as can be easily implemented adding a zero if the block has an even number of ones or a one if the number is odd, so that all codewords have an even number of ones. Parity is an example of Error Detecting Code (EDC). Parity is also capable of detecting every odd number of errors, while an even number of errors will generate an undetected error. Given its simplicity and low overhead, parity is sometimes used at sub-word level to detect more than one error in one word. For instance in [18] an 8 bit-interleaved parity is described, which for a 64 bit word results in 8 times the overhead in terms of check bits. This approach increases area and power overhead linearly instead of exponentially with the detection capability (even if no correction capabilities are added).

SECDED corrects all single errors and detects all double errors in a memory block [18]. The probabilities of miscorrection and detection for more than two errors in the same word depends upon the specific SECDED code employed. In [64], the (39,32) *Hsiao* code has a miscorrection probability of 59.66% for triple errors, while for the (39,32) *Odd-Weight Column* code it is 58.43%. The miscorrection probabilities for the (72,64) *Hsiao* code and the (72,64) *Odd-Weight Column* code are respectively 56.28% and 54.78%. In [116], the *Odd-Weight Column* code (39,32) miscorrects 1.7% of four errors in a word and the (72,64) *Odd-Weight Column* miscorrects 0.8% of them. These codes are examples of Error Correcting Codes (ECCs).

4.1.1. Layout solutions

A way to avoid the exponential increase of overhead due to EDAC codes with increased correcting and detecting capability is to apply Cell Interleaving (CI) at layout level to deal with MBUs instead of using codes capable of detecting more than two errors [117]. In CI, memory cells that belong to the same logical EDAC-protected word are physically non-contiguous in the memory array. In this way, a single ionizing particle capable of causing multiple upsets is more likely to cause several single bit errors in different EDAC-protected words. The figure of merit of a certain cell-interleaved memory is the Interleaving Distance (ID), which indicate how many columns in an SRAM array must be involved during a particle strike to have a non-zero probability of two upsets in the same word. In [117] it is shown that an ID of 16 comes with an area increment of 32% and power increment of 25%, and can be deemed enough to avoid MBUs in most technologies (even with conservative estimations). We will assume a 100% increment in power and area for CI in an SRAM array (i.e. the same given in [117] for a 4 KiB SRAM array when increasing ID from 4 to 32). This value is an upper boundary for the acceptable cost of interleaving in terms of area of power for a memory array, as duplication would have a similar cost.

4.1.2. Refreshing

In [52] a model is proposed to quantify accumulation in an EDAC-protected word, from which (assuming that the scrubbing period is small compared to the MTTE[27] for the accumulation[28]) the MTTE for accumulation of two errors in a word of n bits for an array of M words and a scrubbing period T_s is:

$$\begin{aligned} MTTE &= \int_0^{\inf} e^{-\lambda_{ev}\, n\, M\, t} \left(1 + \lambda_{ev}\, n\, T_s\right)^{M \frac{t}{T_s}}\, dt \\ &= \frac{1}{\lambda_{ev}\, n\, M - \frac{M}{T_s} \ln\left(1 + \lambda_{ev}\, n\, T_s\right)} \end{aligned} \tag{15}$$

The accumulation for three SEUs in the same word can be estimated instead using the following equation, derived in a similar way as in [52]:

$$\begin{aligned} MTTE &= \int_0^{\inf} e^{-\lambda_{ev}\, n\, M\, t} \left(1 + \lambda_{ev}\, n\, T_s + \frac{\lambda_{ev}^2\, n^2\, T_s^2}{2}\right)^{M \frac{t}{T_s}} dt \\ &= \frac{1}{\lambda_{ev}\, n\, M - \frac{M}{T_s} \ln\left(1 + \lambda_{ev}\, n\, T_s + \frac{\lambda_{ev}^2\, n^2\, T_s^2}{2}\right)} \end{aligned} \tag{16}$$

Fig. 10 (left) shows the MTTE for one upset, accumulation of two and three upsets in the same word for extreme upset rates, assuming a 10 min refresh rate (which can be seen as a worst case estimation compared to realistic applications as in [118] is shown that typical lifetime in a LLC is in the order of tens of microseconds). Even with this pessimistic assumption, accumulation is in general negligible compared to the contribution of MBUs (the ratio of MTTE of 'accumulation of two upsets' and 'one upset' is around 400 for $\lambda_{ev} = 10^{-2}$ upsets/bit/day and 2E+5 for 'accumulation of three upsets' and 'one upset'). This implies that accumulation will have negligible impact on failures due MBUs, as the latter are much more common (even considering LC, the ratio between two upsets and one upset is 24 and the ratio between three upsets and one upset is 95). The figure on the right instead shows that, even if the sensitivity of the accumulation to the memory size is the same for all events, the MTTE for large memories is small enough to contribute significantly to the failure rate. For instance mass memories like the one described in [119] have a memory scrubber to read and correct locations according to EDAC codes, therefore limiting accumulation to the scrubbing period. Furthermore, it is worth to note that, while memories with words of 64 bits perform slightly better for one upset because (72,64) is more efficient in terms of added cells compared to (39,32) (i.e. the product n × M is slightly smaller for memories carrying the same amount of bites), memories with words of 32 bits perform better for accumulation of two upsets and (by a larger margin) for accumulation of three upsets. This is intuitive, as accumulation becomes more likely when the number of bits in a word increases.

4.1.3. Cost-effective redundancy for cache arrays

Several processors described in literature employ SED in L1 caches and SECDED in L2C (referred to as EDC/ECC), while others have SECDED in both levels (referred to as ECC/ECC) [77]. The

[26] In the rest of this work we will assume EDAC codes applied to words.

[27] We prefer in this case to use Mean Time To Event (MTTE), instead of MTTF, to avoid confusion with the terminology introduced in Section 2.

[28] In this work we will consider the models for accumulation of two and three bits valid if $T_s < 0.1 \times MTTE$.

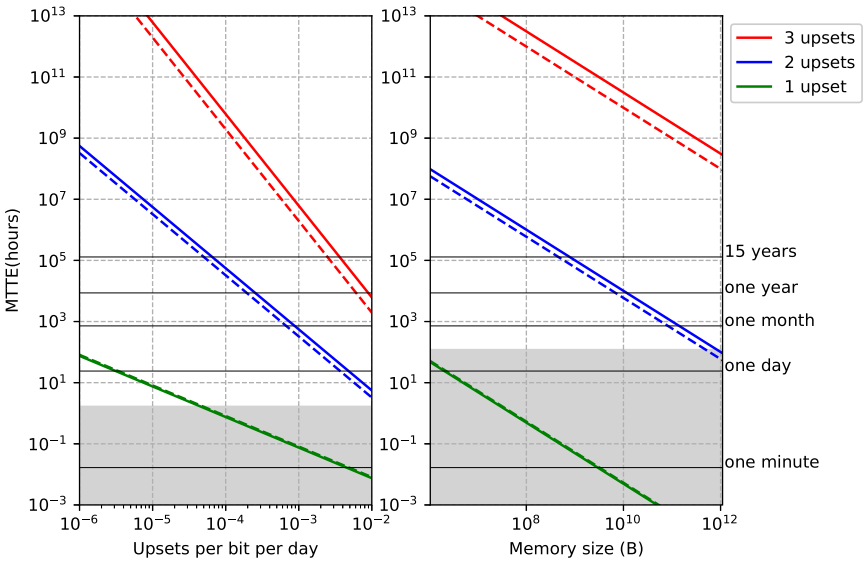

Fig. 10. MTTE for accumulation of errors in the same word when changing upset rate (on the left: memory array size is 32 KiB and refresh rate is 10 min) and memory size (on the right: upset rate is 5E−8 upsets/bit/day and refresh rate is 12 h) according to Eqs. (15) and (16). In both cases the impact of the word length and EDAC code is shown, i.e. solid line for (32,39) and dashed line for (64,72). In gray the range of MTTE where the models for 2 and 3 upsets in the same word are not valid for the selected scrubbing rate.

main argument in favor of the EDC/ECC approach is that it causes a smaller increase in word length, as the increase in word length increases the access latency to the word. For this reason, the latency penalty per access compared to an unprotected L1 cache is less than 1% for the SED, while for the SECDED it is larger (but still below 10%, as the access latency is dominated by the data array's word-line decoder [77]). However, EDC/ECC cannot correct errors in L1 caches, therefore program execution cannot in general resume after detection and a reset is required. This problem is typically mitigated reading the correct, up-to-date value from the next level of the memory hierarchy [113]. In order to make this correction possible, this solution requires WT policy for DC, which incurs in significant performance and power overheads [18]. On average EDC/ECC has a runtime penalty comparable with ECC/ECC (+12% vs. +2% for SPECint in [77]). However for some specific benchmarks the penalty for using EDC/ECC is much higher. For instance, for *bzip2-graphic* the penalty of EDC/ECC over the unprotected version is +157% and +75% for *vortex3*. However, according to the data in [77], SED incurs in less area overhead (virtually none already for L1 caches of 8 KiB), while the SECDED at L1 causes an area overhead between around 50% and 10% depending on the area of the cache (in case of 32 KiB it is around 20%). Using CACTI [86] it can be found that the increase in terms of area for applying ECC in a 1 MiB L2C is around 21%. Regarding power, ECC/ECC has an overhead on the order for 32 KiB of 20% in [77], while using CACTI a 24.46% increase for a cache of 1 MiB. With EDC, also considering the required write to the next level of the memory hierarchy due to the WT policy, it is around 350% [77]. It should be noted that both power and area given previously are at cache subsystem level, thus using them directly in the cost function would overestimate the cost in terms of power and area of cache redundancy (even if caches in most cases consume a large fraction of the power of a processor [120]). To estimate the actual relative increases at processor-level, we model the LE and HE in McPAT [121]. This modeling shows that DC and IC consume respectively 18.95% and 4.53% of the total power consumption for the LE-1 and 17.12% and 4.11% for the HE-1. In the case of LE-4 DCs consume 14.97% of the total power, ICs 3.58% and L2C 18.95% for LE-4. The same fractions for the HE-4 are respectively 14.82%, 3.56% and 11.13%.

Regarding the changes in failure rate and unavailability, limiting our analysis to triple and quadruple errors in the same word, the probability of miscorrection for a SECDED is:

$$P_{SD,misc} = (MBU(3)_{\%} \times P_{misc,3} + MBU(4)_{\%} \times P_{misc,4}) \quad (17)$$

Therefore, the change in failure rate due to the ECC/ECC is:

$$\Delta\hat{\lambda}_w = -[n_c N_{L1}(AVF_{DC_{WB}} + AVF_{IC}) + N_{L2C} AVF_{L2C}](1 - P_{SD,misc}) \quad (18)$$

where N_{L1} is the size of a single L1 cache and n_c is the number of cores. In the case of EDC/ECC the change in failure rate is instead:

$$\begin{aligned}\Delta\hat{\lambda}_w = &-[n_c\, N_{L1}\, (AVF_{DC,WT} + AVF_{IC})(1 - MBU(even)_{\%})] \\ &- N_{L2C}\, AVF_{L2C}\, (1 - P_{SD,misc})\end{aligned} \quad (19)$$

When calculating the change in availability, estimating the $\Delta\hat{\lambda}_{eh,DUE}$ as $-\Delta\hat{\lambda}_w$ is too optimistic, as once a certain location is read, the error handling mechanism will act also on detected errors that would not reach the service interface. To eliminate the fraction of masking due to the propagation from the cache to the outputs of the processor, the Cache Vulnerability Factor (CVF), defined in [122] as the probability of an error in the cache to propagate outside the cache (i.e. being read), can be used instead. In [122] average CVF data for a similar cache configuration are given when running 11 benchmarks from SPEC2000. The CVFs found are 38.03% for $L2C_{WB}$, 57.70% for DC_{WB}, 16.47% DC_{WT}, and 32.05% for IC. The increase in $\lambda_{eh,DUE}$ for ECC/ECC can be then estimated as:

$$\Delta\hat{\lambda}_{eh,DUE} = [n_c N_{L1}(CVF_{DC_{WB}} + CVF_{IC}) + N_{L2C} CVF_{L2C}](1 - P_{SD,misc}) \quad (20)$$

In the case of EDC/ECC the change in $\lambda_{eh,DUE}$ is:

$$\begin{aligned}\Delta\hat{\lambda}_{eh,DUE} = &[n_c\, N_{L1}\, (CVF_{DC_{WT}} + CVF_{IC})\, MBU(odd)_{\%}] \\ &+ N_{L2C}\, CVF_{L2C}(1 - P_{SD,misc})\end{aligned} \quad (21)$$

In Table 6 we compare the cost of applying EDC, ECC and EDC + CI to single core versions and EDC/ECC, ECC/ECC and EDC + CI/ECC to quad-core versions in terms of area, power and performance. In Table 7 the change of reliability shows that while EDC + CI and EDC + CI/ECC have the highest reduction for every technology, for technologies with low fraction of MBUs the improvement they can provide over ECC and ECC/ECC is negligible. The unavailability of quad-core designs with AC technology

Table 6
Relative change (%) in execution time, area and power for redundancy in caches and different designs. Data from [77,117] and modeling in CACTI and McPAT.

Redund.	EDC		ECC		EDC+CI		EDC/ECC		ECC/ECC		EDC+CI/ECC	
Design	LE-1	HE-1	LE-1	HE-1	LE-1	HE-1	LE-4	HE-4	LE-4	HE-4	LE-4	HE-4
$\frac{\Delta T_{ex}}{T_{ex}}$	12	12	2	2	12	12	12	12	2	2	12	12
$\frac{\Delta A}{A}$	∼0	∼0	14	10	72	49	17	9	20	18	31	29
$\frac{\Delta P}{P}$	30	10	5	1	54	20	37	12	7	2	45	16

Table 7
Relative changes in λ_w and U (%), referred to the respective unprotected version with DC_{WB} (according to Eqs. (18), (19), (20), and (21)). The SECDED for ECC is an *Odd-Weight Column* code. In bold the most effective redundancy for each design/technology combination.

Redund.	Design	$\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$	$\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$	$\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$	$\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$	$\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$	$\frac{\Delta U_{LC}}{U_{LC}}$	$\frac{\Delta U_{AC}}{U_{AC}}$	$\frac{\Delta U_{HC}}{U_{HC}}$	$\frac{\Delta U_{SD}}{U_{SD}}$	$\frac{\Delta U_{MD}}{U_{MD}}$
EDC	LE-1	−92	−69	−53	−90	−54	−92	−59	−36	−91	−37
	HE-1	−88	−64	−49	−84	−51	−86	−54	−33	−82	−35
ECC	LE-1	**−94**	−91	−92	−92	−93	−87	−37	−84	−86	−85
	HE-1	**−90**	−85	−85	−85	−89	−82	−34	−77	−78	−80
EDC+CI	LE-1	**−94**	**−93**	**−93**	**−93**	**−94**	**−95**	**−94**	**−94**	**−94**	**−95**
	HE-1	**−90**	**−88**	**−86**	**−86**	**−90**	**−89**	**−87**	**−85**	**−85**	**−89**
EDC/ECC	LE-4	−95	−79	−70	−94	−71	−77	56	−47	−77	−47
	HE-4	−92	−76	−66	−89	−68	−75	54	−50	−73	−46
ECC/ECC	LE-4	**−96**	−93	**−95**	**−95**	**−96**	−75	66	−70	−74	−71
	HE-4	**−93**	−90	−90	**−90**	**−93**	−73	64	−67	−71	−69
EDC+CI/ECC	LE-4	**−96**	**−95**	**−95**	**−95**	**−96**	**−79**	**38**	**−75**	**−78**	**−75**
	HE-4	**−93**	**−91**	**−91**	**−90**	**−93**	**−76**	**37**	**−71**	**−74**	**−73**

Table 8
Cost-effectiveness of several redundancies for caches for several technologies and weights. In bold the most cost-effective solutions for each combination of redundancy, design and technology.

Redund.	Design	$C_{d,LC}$	$C_{d,AC}$	$C_{d,HC}$	$C_{d,SD}$	$C_{d,MD}$	$C_{p,LC}$	$C_{p,AC}$	$C_{p,HC}$	$C_{p,SD}$	$C_{p,MD}$
EDC	LE-1	**−1.66**	−1.09	−0.71	**−1.63**	−0.72	**−0.42**	**−0.19**	−0.04	**−0.41**	−0.05
	HE-1	**−1.66**	−1.11	−0.74	**−1.57**	−0.78	**−0.48**	**−0.26**	−0.12	**−0.45**	−0.13
ECC	LE-1	**−1.66**	−1.13	**−1.61**	**−1.63**	**−1.64**	−0.37	−0.12	**−0.36**	−0.36	**−0.36**
	HE-1	−1.61	−1.09	**−1.51**	−1.52	**−1.59**	−0.38	−0.13	**−0.34**	−0.35	**−0.37**
EDC+CI	LE-1	−1.23	**−1.22**	−1.20	−1.20	−1.23	0.04	0.04	0.05	0.05	0.04
	HE-1	−1.42	**−1.38**	−1.33	−1.33	−1.42	−0.21	−0.19	−0.18	−0.17	−0.21
EDC/ECC	LE-4	−1.42	0.06	−0.87	−1.40	−0.88	−0.23	0.47	−0.02	−0.23	−0.02
	HE-4	**−1.53**	−0.09	−1.03	**−1.48**	−1.00	−0.38	**0.30**	−0.19	**−0.36**	−0.17
ECC/ECC	LE-4	**−1.52**	−0.09	**−1.46**	**−1.51**	**−1.48**	**−0.28**	**0.43**	**−0.25**	**−0.27**	**−0.26**
	HE-4	−1.51	−0.11	**−1.42**	−1.46	**−1.46**	**−0.30**	0.40	**−0.26**	−0.28	**−0.27**
EDC+CI/ECC	LE-4	−1.34	**−0.16**	−1.29	−1.32	−1.30	−0.13	0.45	−0.11	−0.13	−0.12
	HE-4	−1.44	**−0.29**	−1.37	−1.39	−1.41	−0.27	**0.30**	−0.24	−0.25	−0.25

increases compared to a version without redundancy because of DUEs due to a large fraction of MBU(2). In Table 8 the total cost is shown for each technology/design/redundancy combination. EDC is the most cost-effective for both single-core designs and weight factors for technologies with low fraction of MBUs (i.e. LC, SD). When the fraction of MBUs becomes significant, its distribution determines the most cost-effective redundancy. For instance, AC requires EDC + CI and EDC + CI/EDC because most of its fraction of MBUs causes more than two upsets, while ECC and ECC/ECC is enough in most cases for HC as in this case the majority of MBUs causes only two upsets. It should also be noted that in the case of AC the cost of applying EDAC codes to quad-core designs is always positive, implying that not applying EDAC codes would be more cost-effective. However, EDAC codes are typically applied anyway to achieve requirements in terms of $MTTF_w$.

4.2. Choosing the redundancy for the rest of the processor

The rest of the processor can be divided in residual (smaller than caches) SRAM arrays (e.g. RFs) and mixed logic (i.e. composed of FFs and combinational logic). Two main approaches can be found in literature: protecting separately RFs and mixed logic [78] (Sections 4.2.1 and 4.2.2) or protecting them simultaneously [20] (Section 4.2.3).

4.2.1. Choosing the redundancy for the RFs

Similarly to caches, RFs are typically protected with information redundancy. However, as they are smaller than caches, replicating the RF may be a viable solution. For this reason, we compare the effects in the case of SECDED (RF-ECC) and Triple Modular Redundancy (TMR) of the RFs (RF-TMR). In [106] RF-ECC is reported to increase the power of the RF by 100% and the area of 4.9%. Table 9 reports how these estimations increase area and power at processor level for the two designs and Table 10 which relative variations in terms of failure rate and unavailability they produce. As shown in Table 9, RF-TMR is in general more expensive in terms of area and power, although less expensive in terms of performance. In [17], a Double Modular Redundancy (DMR) with parity is proposed as a less expensive version of RF-TMR, which is capable of achieving the same relative change in failure rate and unavailability with lower overhead in terms of area and power. Nevertheless, RF-TMR can be more cost-effective than RF-ECC when the focus is on performance and for some designs (e.g. HE-1), as shown in Table 11.

4.2.2. Choosing the redundancy for mixed logic

To protect the rest of the processor composed of mixed logic, one of the most common approach is the one described in [78]

Table 9
Relative increase (%) in execution time, area and power for redundancies protecting RFs, mixed logic and both simultaneously. Data from [78,106,123–126] and modeling in McPAT.

Redund.	RF-ECC		RF-TMR		FF-TMR		FFD-TMR		C-TMR		C-DMR	
Design	LE-1	HE-1	LE-1	HE-1	LE-1	HE-1	LE-1	HE-1	LE-1	HE-1	LE-1	HE-1
$\frac{\Delta T_{ex}}{T_{ex}}$	4	4	1	1	8	8	45	45	0	0	0	0
$\frac{\Delta A}{A}$	0	1	8	26	9	32	15	53	65	107	28	51
$\frac{\Delta P}{P}$	5	9	9	17	66	65	71	71	60	61	43	44

Table 10
Relative changes (%) in wrong outputs and of unavailability for the LE-1 and HE-1 when adding redundancy to IRF and RFR. In bold the most effective solutions for each combination of redundancy, design and technology.

Redund.	Design	$\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$	$\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$	$\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$	$\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$	$\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$	$\frac{\Delta U_{LC}}{U_{LC}}$	$\frac{\Delta U_{AC}}{U_{AC}}$	$\frac{\Delta U_{HC}}{U_{HC}}$	$\frac{\Delta U_{SD}}{U_{SD}}$	$\frac{\Delta U_{MD}}{U_{MD}}$
RF-ECC	LE-1	**−3.6**	**−3.5**	−1.7	**−3.5**	−3.5	−1.2	−0.3	−1.2	−1.2	−1.2
	HE-1	**−2.9**	−3.7	−4.6	−4.6	−2.9	−1.8	−1.9	−2.9	−2.9	−1.8
RF-TMR	LE-1	**−3.6**	**−3.5**	**−3.5**	**−3.5**	**−3.6**	**−1.4**	**−1.4**	**−1.3**	**−1.3**	**−1.4**
	HE-1	**−2.9**	**−3.8**	**−4.7**	**−4.7**	**−2.9**	**−1.9**	**−2.5**	**−3.0**	**−3.0**	**−1.9**

Table 11
Cost of redundancy in IRF and FRF in the case of SECDED (RF-ECC) and triplication of the RFs (RF-TMR). In bold the most cost-effective solutions for each combination of redundancy, design and technology.

Redund.	Design	$C_{d,LC}$	$C_{d,AC}$	$C_{d,HC}$	$C_{d,SD}$	$C_{d,MD}$	$C_{p,LC}$	$C_{p,AC}$	$C_{p,HC}$	$C_{p,SD}$	$C_{p,MD}$
RF-ECC	LE-1	**−0.01**	**0.00**	**0.01**	**−0.01**	**−0.01**	**0.05**	**0.05**	**0.05**	**0.05**	**0.07**
	HE-1	**0.01**	**0.00**	**−0.02**	**−0.02**	**0.03**	0.15	0.15	0.14	0.14	0.15
RF-TMR	LE-1	0.04	0.04	0.04	0.04	0.04	0.08	0.08	0.08	0.08	0.08
	HE-1	0.04	0.03	0.02	0.02	0.04	**0.08**	**0.07**	**0.07**	**0.07**	**0.08**

for the LEON2FT. This approach uses FFs with Triple Modular Redundancy (FF-TMR), sampling and storing a bit on three different FFs and using a voter on the output to mask upsets and provide the correct value without any CCs of latency. To avoid common failures to the FFs in the TMR, each of the three FFs can have separate clock-trees, so that a SET in one clock-tree can be tolerated even if the data of a complete lane of thousands of registers is corrupted [78]. FF-TMR is applied also in [127], where it provides a 2.5x reduction in wrong commands/content at the outputs (used in conjunction with safe FSMs). The authors of [127] suggest that between 20% and 40% of the errors in the baseline processor are the result of SETs. As a matter of fact, SETs in the combinational logic can still be sampled by the majority of the FFs of a FF-TMR.

However, triplicating both sequential elements and combinational logic (to address also SETs) is reported to increase T_{min} by 60% and area by 326% [128], which is a very high cost. To address also SETs with less overhead, [124] proposes a FF-level TMR with different delays for the three FFs (FFD-TMR) to avoid that a SET is sampled in more than a FF. The area of a FF-TMR cell is [123] is 3.47x larger than a regular FF and consumes 2.7x more power. FFD-TMR cells are instead reported to be about 6x larger than a FF in [124] and 5.2x in [125]. In [125] FFD-TMR cells consume between 3 to 4x more compared to a regular FF, depending on the switching activity. The minimum clock period in the FFD-TMR version is 45% longer than the baseline, showing a substantially larger penalty compared to FF-TMR without delays. As a matter of fact, in [78] FF-TMR increases the minimum period for correct execution of 8% on a 250 nm ASIC technology and the same value is given from different authors in [126] for the same processor using a 65 nm ASIC technology.

Furthermore, in order to minimize the penalty in frequency, the triplicated FFs are typically placed close to each other. In this way, MBUs can cause wrong data to become majority and to be promoted to correct state, state causing data corruption [126].

As the cross section for a triplicated FF is between three to one order of magnitude less compared to the cross section of an unprotected FF in [129], for FF-TMR and FFD-TMR it is assumed that 10% of the events will corrupt more than one of the FFs in MD and HC, 1% of the events will corrupt more than one FF in AC and 0.1% in LC and SD (see Table 12). Furthermore, FFD-TMR is considered to mask all SETs, as [124] report immunity to spikes up to 105 ps.

Despite the optimistic assumptions on the capability of FFD-TMR to mask all the SETs, its cost is so high that FF-TMR is preferable for all designs, technologies and weights considered (the table for the cost-effectiveness is not reported for sake of brevity). This is due to the large area overhead of FFD-TMR for C_d and to the performance overhead of FFD-TMR for C_p. Even considering the weights and type of technology for which FFD-TMR is less expensive (C_d and SD) and reducing the overhead compared to FF-TMR by 50% (e.g. $\frac{\Delta T_{ex}}{T_{ex}} = 0.27$), FFD-TMR is still less cost-effective than FF-TMR. However, the cost of both FF-TMR and FFD-TMR is positive for any design/technology/weight combinations, showing that they are both expensive types of redundancy in general.

To reduce the cost of redundant sequential elements, different designs of sequential elements have been proposed to replace FFD-TMR and FF-TMR. For instance, a DICE-FF cell has a reduction of 61.54% in terms of area, between 40.30% and 48.72% in terms of power (depending on the switching activity) and 15.13% in terms of delay compared to a sequential element of FF-TMR [125]. However, while FF-TMR uses three simple FFs and a voter (and therefore in principle could be implemented in RTL) as a redundant cell, FFD-TMR and other designs require technology-specific adjustment at layout and electrical level within the sequential element. For instance, the DICE-FF requires the design of a custom cell not available in commercial technologies [125].

4.2.3. Protecting simultaneously small SRAM arrays and mixed logic

An alternative to the approach shown in Sections 4.2.1 and 4.2.2 is to replicate entirely the core, excluding large SRAM arrays (i.e. caches) that can be protected efficiently by information redundancy as shown in Section 4.2. In [20] the TCLS is described, a core-level TMR implementation of the ARM Cortex R5. The three cores share an IC and a DC. In [20], this approach is not found to

Table 12
Relative changes (%) in λ_w and U for redundancies protecting mixed logic. In bold the most effective redundancy for each design/technology combination.

Redund.	Design	$\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$	$\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$	$\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$	$\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$	$\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$	$\frac{\Delta U_{LC}}{U_{LC}}$	$\frac{\Delta U_{AC}}{U_{AC}}$	$\frac{\Delta U_{HC}}{U_{HC}}$	$\frac{\Delta U_{SD}}{U_{SD}}$	$\frac{\Delta U_{MD}}{U_{MD}}$
FF-TMR	LE-1	**−2.2**	−1.7	−0.9	−1.5	−1.3	**−3.7**	−3.0	−2.5	−2.5	−2.6
	HE-1	**−7.1**	−6.7	−3.2	−5.1	−4.3	**−8.9**	−7.1	−3.5	−8.2	−6.1
FFD-TMR	LE-1	**−2.2**	**−3.0**	**−2.8**	**−3.8**	**−0.9**	**−3.7**	**−4.1**	**−3.2**	**−4.8**	**−1.5**
	HE-1	**−7.1**	**−8.1**	**−8.1**	**−11.1**	**−2.8**	**−8.9**	**−9.8**	**−7.9**	**−11.7**	**−4.9**

Table 13
Relative changes in λ_w and U (%) for redundancies protecting both small SRAM array and mixed logic for different technologies and designs. In bold the most effective redundancy for each design/technology combination.

Redund.	Design	$\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$	$\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$	$\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$	$\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$	$\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$	$\frac{\Delta U_{LC}}{U_{LC}}$	$\frac{\Delta U_{AC}}{U_{AC}}$	$\frac{\Delta U_{HC}}{U_{HC}}$	$\frac{\Delta U_{SD}}{U_{SD}}$	$\frac{\Delta U_{MD}}{U_{MD}}$
C-TMR	LE-1	**−5.8**	**−6.6**	**−7.4**	**−7.4**	**−5.8**	**−5.1**	**−5.7**	**−6.3**	**−6.3**	**−5.1**
	HE-1	**−10.0**	**−12.2**	**−14.4**	**−14.4**	**−10.0**	**−10.8**	**−12.7**	**−14.6**	**−15.2**	**−10.8**
C-DMR	LE-1	**−5.8**	**−6.6**	**−7.4**	**−7.4**	**−5.8**	4.5	4.6	4.8	4.8	4.5
	HE-1	**−10.0**	**−12.2**	**−14.4**	**−14.4**	**−10.0**	5.8	3.7	2.6	6.9	5.2

Table 14
Comparison of cost-effectiveness of C-TMR, C-DMR and the most cost-effective solution from Table 11 and FF-TMR. In bold the most cost-effective redundancy for each design/technology combination.

Redund.	Design	$C_{d,LC}$	$C_{d,AC}$	$C_{d,HC}$	$C_{d,SD}$	$C_{d,MD}$	$C_{p,LC}$	$C_{p,AC}$	$C_{p,HC}$	$C_{p,SD}$	$C_{p,MD}$
C-TMR	LE-1	0.52	0.50	0.49	0.49	0.52	0.59	0.58	0.57	0.57	0.59
	HE-1	0.63	0.59	0.55	0.54	0.63	0.76	0.75	0.73	0.73	0.76
C-DMR	LE-1	0.34	**0.34**	**0.33**	**0.33**	**0.34**	**0.36**	**0.36**	**0.36**	**0.36**	**0.36**
	HE-1	0.43	0.39	**0.36**	**0.40**	**0.43**	**0.48**	**0.46**	**0.45**	**0.47**	**0.48**
Table 11 & FF-TMR	LE-1	**0.32**	0.35	0.37	0.34	**0.34**	0.48	0.49	0.50	0.49	0.50
	LE-1	**0.36**	**0.37**	0.42	0.36	0.44	0.58	0.59	0.61	0.58	0.60

cause frequency penalties. However, in a previous work [130] a 10% penalty is reported, which shows that, even if not critical as in the case of FFD-TMR, the frequency can actually be penalized. A drawback of this approach is that errors are not masked with zero latency like in the case of FF-TMR and FFD-TMR, even if the T_{eh} can be kept low enough compared to a hard or soft reset. When a discrepancy in the outputs is found, the processor takes 923 CCs to save and 909 CCs to restore the state (with caches enabled), for a total of 1832 CCs. The time required to propagate the error to the service interface does not influence the availability, as correct operation is ensured until the error is propagated to the outputs. The propagation time has instead to be considered for accumulation, as it is possible that the data selected as 'golden' and replicated into all the three cores have latent errors that will manage to reach the outputs of the three processors completely undetected after the state is restored. However, even considering the most vulnerable design/technology combination (HE-1 without caches for HC technologies and $\lambda_{ev} = 10^{-6}$ upsets/bit/day) where we use as T_{prop} the worst-case propagation time [80] (1,204 CCs at 100 MHz, i.e. 12.04 μs), the probability of accumulation of two errors is negligible (8 orders of magnitude less compared to the failure rate due to the cacheless HE-1).

The situation in terms of unavailability is quite different with Core-level Dual Modular Redundancy (C-DMR) (e.g. [3]), as it is not possible to vote to chose a golden version when a mismatch is found and a soft reset is required. Another possibility is to save periodically the status of one of the core [79], but this generates substantial penalties in terms of execution time (ranging from +26% to 548%).

Table 13 and Table 14 show the comparison between C-TMR, C-DMR, and the most cost-effective solution found combining the results from Table 11 and FF-TMR. It shows that protecting separately RFs and mixed logic and replicating the core have in general similar cost-effectiveness, so they are both viable solutions. As a trend, FF-TMR and RF-ECC are more cost-effective for (older) technologies with relative low fraction of MBUs and high masking of SETs (i.e. LC, AC) and when the focus is on dependability. For (newer) technologies higher fraction of MBUs and SETs sampled C-DMR is more cost-effective. In case the focus is on performance (C_p), C-DMR is found as the most cost-effective solution regardless of other parameters. C-TMR is generally the least cost-effective solution because of a larger area and power overhead. It becomes more cost-effective than C-DMR only in case the weight of the availability is higher ($\epsilon = 5$ instead of 1 for C_d).

5. Expected in-orbit behavior and validation

Fig. 11 (left) shows the absolute $MTTF_w$ of LE-4 and HE-4 before and after the most cost-effective solutions according to C_d are employed. It is worth to note that, while from a quantity perspective the vulnerability is roughly the same for all types of technologies and quad-core designs (LE-4 and HE-4), the quality of the vulnerability is so different that applying redundancy produces very different $MTTF_w$ (around one order of magnitude of difference). Fig. 5 shows that this is the case because caches dominate the $MTTF_w$ and have small variations in terms of vulnerability changing type of technology. Also, the comparison between LE-4-FT/HE-4-FT and LE-1-FT/HE-1-FT in Fig. 12 shows that going multicore has a large cost in terms of availability, e.g. for $\lambda_{ev} = 10^{-7}$ upsets/bit/day LE-4-FT and HE-4-FT cannot meet an availability target of 99.999%, while both LE-1-FT and HE-1-FT can meet a 99.99999% target. This findings show that techniques to reduce the vulnerability of L2Cs, e.g. employing a $L2C_{WT}$ to lower the AVF of the [82] and to decrease the $T_{eh,due}$, may be a cost-effective solution to increase the $MTTF_w$ and the availability of quad-core processors.

When the focus is on a target $MTTF_w$ instead of cost-effectiveness, a chart similar to Fig. 11 (right) can be employed to evaluate possible trade-offs. Once the $MTTF_w$ target is set, a combination of microarchitecture and redundancy will be fit only if a technology (for the target environment) exists for which the horizontal line corresponding to the microarchitecture/redundancy is below the oblique line representing the $MTTF_w$ target. Assuming a target of $MTTF_w = 10{,}000$ hr (1.14 years), the combinations below the respective line are:

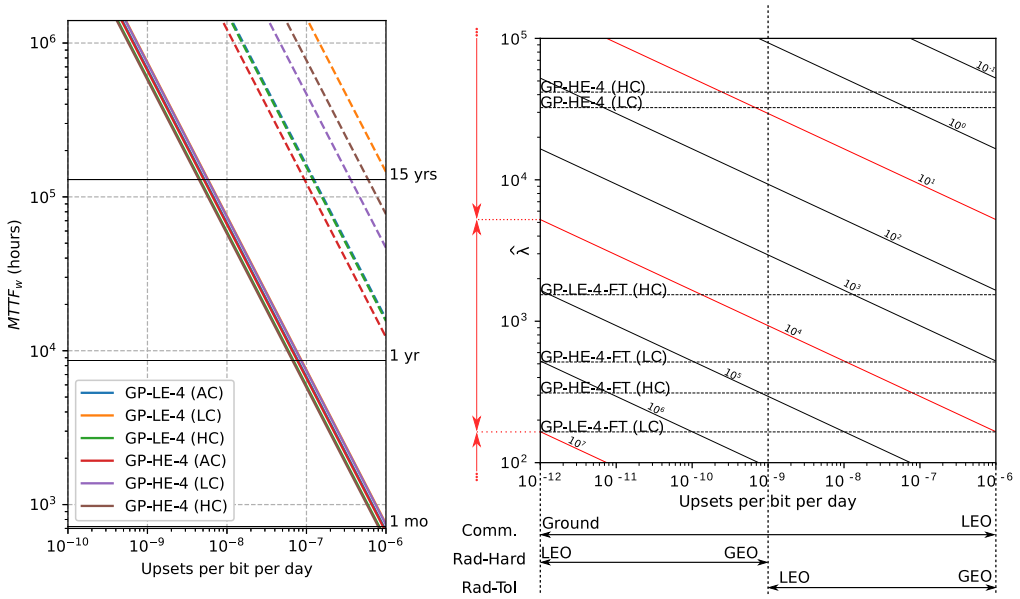

Fig. 11. On the left, $MTTF_w$ for LE-4 and HE-4 in different technologies. Solid lines indicate unprotected versions and dashed lines versions with the most cost-effective redundancy found according to C_d. On the right, $MTTF_w$ in hours for different designs depending on $\hat{\lambda}$ and λ_{ev}. (FT indicates a "Fault-Tolerant" implementation, i.e. employing the most cost-effective redundancies found with C_d). Red lines indicate different classes of designs in terms of $MTTF_w$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

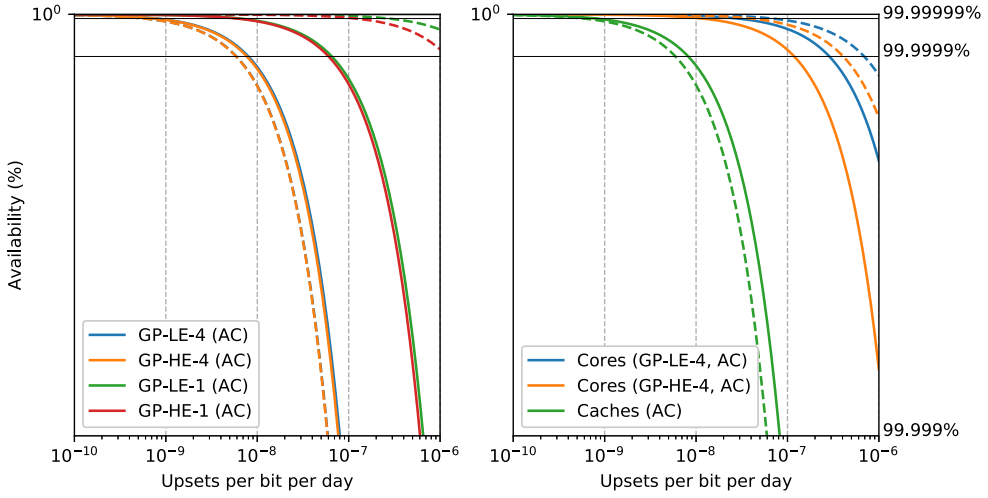

Fig. 12. On the left, availability of LE-4 and HE-4 (AC). On the right, decomposition in core (RFs + mixed logic) and caches. Solid lines indicate unprotected versions and dashed lines versions with the most cost-effective redundancy found for each combination of design/technology. The unavailability has been estimated with Eq. (8).

- HE-4-FT (HC) (C-DMR + EDC/ECC) on RH technology in LEO (HC, $\lambda_{ev} \leq 10^{-10}$ upsets/bit/day)
- HE-4-FT (LC) (FF-TMR + RF-ECC + ECC/ECC) on rad-hard technology in LEO and GEO or rad-tol (radiation-tolerant) in LEO (the latter only up to $\lambda_{ev} = 10^{-8}$ upsets/bit/day).
- HE-4-FT (HC) (C-TMR + ECC/ECC) on rad-hard technology in LEO and GEO or rad-tol in LEO (the latter only up to $\lambda_{ev} = 10^{-7}$ upsets/bit/day).
- LE-4-FT (LC) (FF-TMR + RF-ECC + ECC/ECC) for commercial technologies in LEO, and both rad-hard and rad-tol technologies both in GEO and LEO.

Interestingly, Fig. 11 (right) shows that processors without redundancy achieve low $MTTF_w$ with commercial technology at ground level. This is reflected by the trend of including redundancies in processors for terrestrial applications, mainly in caches [113] and sometimes also in RFs [112]. However, the figure also shows that designs intended for space operate also at $\lambda_{ev} \geq 10^{-10}$ upsets/bit/day and therefore they require more redundancy. Fig. 11 (right) also shows that having a limited range for λ_{ev} implies that it is not possible in general to make a certain design reliable enough to achieve an arbitrary $MTTF_w$ target using a rad-hard technology. Therefore, processors in Fig. 11 (right) can be binned in three classes in terms of $MTTF_w$ using non-overlapping $MTTF_w$ isolines (e.g. 10^7, 10^4 and 10 h, shown in red). For instance, none of the design/redundancy combinations can meet a target $MTTF_w$ higher than 10^7 h and there is a higher level of $MTTF_w$ for which quad-core processors are not fit for any design/redundancy combination and the designer must resort to smaller implementations.

Table 15
Summary of the framework presented and adaptations required for use with different designs.

# - Step	Ref.	Possible adaptations/extensions
1 - Definition of fault and error models	Section 2.1	Use of $SF_{SET\%}$, $SF_{FF\%}$, $SBU_{\%}$, $MBU(n)_{\%}$, and λ_{ev} for specific technology and frequency.
2 - Definition of failure models	Section 2.3 Fig. 3	Use of QoS to distinguish between acceptable and unacceptable behavior.
3 - Estimation of N_{eq} or SER	Section 3.2 Eq. (4)	MCPath can be employed for closed source processors (only total area).
4 - Estimation of AVF	Section 3.3.4	Extended microarchitectural simulators (e.g. [103]) or fault injection on FPGA prototype.
5 - Estimation of $\hat{\lambda}$ and $\hat{U}$	Section 3.1 Eqs. (6)–(8)	Use of alternative derating factors and decompositions to convert SER to $\hat{\lambda}$ (e.g Section 3.4.4).
7 - Application of redundancy	Section 4 Eqs. (11)–(14)	Requires estimation of P_{det}. If AVF' and $\Delta N_{eq}/N$ are unknown, they can be approximated respectively as AVF and 0.
8 - Analysis of cost-effectiveness	Section 4 Eq. (10)	Sensitivity analysis on weights and technical parameters ($\Delta T_{ex}/T_{ex}$, $\Delta A/A$, $\Delta P/P$).
9 - Meet requirements	Section 5 Figs. 11–12	Use of more complex optimization algorithms (e.g. [25]) to minimize cost of achieving $MTTF \leq MTTF_{target}$ and $U \leq U_{target}$.
10 - Validation	Section 5.1	SEE radiation tests (protons, heavy ions) and IOD missions.

5.1. Validation

The most common method to validate a processor for usage in space is radiation testing [78,131]. The main advantage of radiation testing is that it can reproduce exactly the physical mechanisms that will be experienced by the device in space. For this reason, radiation testing can be used both to validate the design and as a validation of the error models employed to select the redundancy (e.g. fraction of MBUs and of SETs sampled).

Sometimes FI is proposed as a validation method. However, FI is not capable of validating the fault models (e.g. percentage of MBUs) as the model of fault injected is arbitrary. On the other hand, radiation testing typically have difficult controllability and observability [3]. Therefore, it is hard to pinpoint where the error was generated. Furthermore, in order to achieve meaningful statistics in a limited time, sometimes the flux of particles employed during radiation testing is several order of magnitude higher than in space. This can produce artifacts, as, for instance, the probability of accumulation of two errors in the scrubbing period is much larger than in space. A field example is provided in [132], where the beam flux had to be throttled down in order to allow error handling in caches to complete successfully and to allow the logging of errors.

Several works in literature (e.g. [9,132]) compare failure rates from FIs and simulations to failure rates during beam testing. The most severe underestimation found in literature is of a factor 20x [51] compared to data from radiation tests. However, this value has been found by simply multiplying the AVF by the population of FFs, so this can be seen as an upper boundary of the possible underestimation. For instance, in [9] the underestimation of the AVF compared to radiation tests is of 11x and the expected failure rate on the field lies in between these two values. This suggests the adoption of a safety factor of at least 10 when setting a target in terms of $MTTF_w$ and to prefer radiation testing for validation, as it provides a worst case estimation. However, it is shown in [132] a similar AVF for FI, protons tests and neutrons tests (respectively 5.02%, 4.35%, 2.65%) when the flux is tuned down enough.

Typically after radiation test the processor is validated in space with an In-Orbit Demonstration (IOD) mission. Data of the behavior of processors in space are not common in literature. Data from [40] shows in-orbit statistics for six identical LEO satellites. The average number of reset is 4.67 reboots per year for each satellite, with an $MTTF_{DUE}$ of 2.57 months. This reflects roughly our models for a LE-1 (typically employed as OBC [7]) for technologies with λ_{ev} ranging from 10^{-8} to 10^{-7} upsets/bit/day (rad-tol technology in LEO).

5.2. Summary

A summary of the framework (containing the description of each step, the references to sections, figures and equations in the paper and possible adaptations or extensions) is reported in Table 15.

6. Conclusion

This paper provides readers familiar with processors with a framework to evaluate the fitness of a microarchitecture for the space environment or any other environment where failure rates are dominated by soft errors. This framework allows to include considerations on soft errors when selecting and configuring an open-source IP core like most of those based on the RISC-V ISA.

Models from literature were introduced and further developed to evaluate the vulnerability of different processor units and evaluate the cost-effectiveness of redundancy in terms of penalties in area, performance, power and availability for several case studies. However, the framework can be easily adapted to different designs and data for a specific technology can be employed to model a specific implementation. Furthermore, the reader is provided also with tools to find the microarchitecture/redundancy/technology combinations which meet specific $MTTF_w$ and availability requirements.

From the models developed, technology and microarchitecture are the factors impacting the most on the dependability of a processor. Furthermore, this work also highlights that estimations of AVF are not the only concern when characterizing the dependability of processors, as other parameters influence the final dependability of the design (e.g. total area, the ratio between sequential and combinational area, temporal masking, etc.) in a comparable way. Caches are shown to be the most vulnerable structures (especially in multi-core processors) and therefore information redundancy in caches is typically very cost-efficient. However, it can be expensive in terms of availability for particular distributions of MBUs for which the number of uncorrectable errors is high. Furthermore, scrubbing has low efficacy in caches (as opposed to when dealing with large external memories), as accumulation in caches has negligible effects compared to MBUs.

Work is still required to characterize the SER in space of ASIC technologies below 28 nm (for instance in terms of fraction of MBUs for unprotected FFs and FF-TMR) and some specific relationships between AVF and microarchitectural choices (for instance the effect of different microarchitectures on the AVF of caches). Furthermore, at the moment of writing no validated extended microarchitectural simulators to estimate soft error vulnerability supporting the RISC-V ISA are available to the public.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: This work has been partially funded by Cobham Gaisler AB.

Acknowledgments

This work was supported by the European Space Agency under the NPI Program, Cobham Gaisler AB, and Delft University of Technology.

References

[1] G. Furano, A. Menicucci, Roadmap for on-board processing and data handling systems in space, in: Dependable Multicore Architectures at Nanoscale, Springer, 2018, pp. 253–281.
[2] J. Andersson, M. Hjorth, F. Johansson, S. Habinc, LEON Processor devices for space missions: First 20 years of LEON in space, in: 2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT), 2017, pp. 136–141, http://dx.doi.org/10.1109/SMC-IT.2017.31.
[3] G. Furano, S. Di Mascio, T. Szewczyk, A. Menicucci, L. Campajola, F. Di Capua, A. Fabbri, M. Ottavi, A novel method for SEE validation of complex socs using low-energy proton beams, in: 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2016, pp. 131–134, http://dx.doi.org/10.1109/DFT.2016.7684084.
[4] F. Bezerra, D. Dangla, F. Manni, J. Mekki, D. Standarovski, R.G. Alia, M. Brugger, S. Danzeca, Evaluation of an alternative low cost approach for SEE assessment of a soc, in: 2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS), 2017, pp. 1–5, http://dx.doi.org/10.1109/RADECS.2017.8696219.
[5] M. Pignol, COTS-Based applications in space avionics, in: 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010), 2010, pp. 1213–1219, http://dx.doi.org/10.1109/DATE.2010.5456992.
[6] R. Hillman, G. Swift, P. Layton, M. Conrad, C. Thibodeau, F. Irom, Space processor radiation mitigation and validation techniques for an 1, 800 MIPS processor board, in: Proceedings of the 7th European Conference on Radiation and its Effects on Components and Systems, 2003. RADECS 2003, 2003, pp. 347–352.
[7] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, C. Monteleone, Leveraging the openness and modularity of RISC-V in space, J. Aerosp. Inf. Syst. 16 (11) (2019) 454–472, http://dx.doi.org/10.2514/1.I010735.
[8] M. Maniatakos, M.K. Michael, Y. Makris, Investigating the limits of AVF analysis in the presence of multiple bit errors, in: 2013 IEEE 19th International on-Line Testing Symposium (IOLTS), 2013, pp. 49–54, http://dx.doi.org/10.1109/IOLTS.2013.6604050.
[9] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, P. Rech, Demystifying soft error assessment strategies on ARM cpus: Microarchitectural fault injection vs. Neutron beam experiments, in: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2019, pp. 26–38, http://dx.doi.org/10.1109/DSN.2019.00018.
[10] A. Avizienis, J.. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (1) (2004) 11–33, http://dx.doi.org/10.1109/TDSC.2004.2.
[11] ISO, Road vehicles – functional safety, 2011.
[12] B. Kosinski, K. Dodson, Key attributes to achieving > 99.99 satellite availability, in: 2018 IEEE International Reliability Physics Symposium (IRPS), 2018, pp. 6A.3–1–6A.3–10, http://dx.doi.org/10.1109/IRPS.2018.8353620.
[13] D. Oliveira, P. Navaux, P. Rech, Increasing the efficiency and efficacy of selective-hardening for parallel applications, in: 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019, pp. 1–6, http://dx.doi.org/10.1109/DFT.2019.8875300.
[14] T.-J. Kwon, J. Sondeen, J. Draper, Design trade-offs in floating-point unit implementation for embedded and processing-in-memory systems, in: 2005 IEEE International Symposium on Circuits and Systems, 2005, pp. 3331–3334, http://dx.doi.org/10.1109/ISCAS.2005.1465341.
[15] H. Cho, E. Cheng, T. Shepherd, C. Cher, S. Mitra, System-level effects of soft errors in uncore components, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (9) (2017) 1497–1510, http://dx.doi.org/10.1109/TCAD.2017.2651824.
[16] M. Ebrahimi, A. Evans, M.B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, R. Seyyedi, Comprehensive analysis of sequential and combinational soft errors in an embedded processor, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (10) (2015) 1586–1599, http://dx.doi.org/10.1109/TCAD.2015.2422845.
[17] M. Fazeli, A. Namazi, S.G. Miremadi, An energy efficient circuit level technique to protect register file from MBUs and SETs in embedded processors, in: 2009 IEEE/IFIP International Conference on Dependable Systems Networks, 2009, pp. 195–204, http://dx.doi.org/10.1109/DSN.2009.5270337.
[18] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, J. Hoe, Multi-bit error tolerant caches using two-dimensional error coding, in: 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), 2007, pp. 197–209, http://dx.doi.org/10.1109/MICRO.2007.19.
[19] S. Wang, J. Hu, S.G. Ziavras, Replicating tag entries for reliability enhancement in cache tag arrays, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (4) (2012) 643–654, http://dx.doi.org/10.1109/TVLSI.2011.2111469.
[20] X. Iturbe, B. Venu, E. Ozer, S. Das, A triple core lock-step (TCLS) ARM cortex-r5 processor for safety-critical and ultra-reliable applications, in: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), 2016, pp. 246–249, http://dx.doi.org/10.1109/DSN-W.2016.57.
[21] J. Ballast, T. Amort, M. Cabanas-Holmen, E.H. Cannon, R. Brees, C. Neathery, S. Fischer, W. Snapp, A method for efficient radiation hardening of multicore processors, in: 2015 IEEE Aerospace Conference, 2015, pp. 1–11.
[22] S. Mittal, J.S. Vetter, A survey of techniques for modeling and improving reliability of computing systems, IEEE Trans. Parallel Distrib. Syst. 27 (4) (2016) 1226–1238, http://dx.doi.org/10.1109/TPDS.2015.2426179.
[23] T. Li, J.A. Ambrose, R. Ragel, S. Parameswaran, Processor design for soft errors: Challenges and state of the art, ACM Comput. Surv. 49 (3) (2016) http://dx.doi.org/10.1145/2996357.
[24] E. Cheng, S. Mirkhani, L.G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M.R. Stan, K. Lilja, J.A. Abraham, P. Bose, S. Mitra, CLEAR: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores, in: Proceedings of the 53rd Annual Design Automation Conference, in: DAC '16, Association for Computing Machinery, New York, NY, USA, 2016, http://dx.doi.org/10.1145/2897937.2897996.
[25] A. Savino, A. Vallero, S. Di Carlo, Redo: Cross-layer multi-objective design-exploration framework for efficient soft error resilient systems, IEEE Trans. Comput. 67 (10) (2018) 1462–1477.
[26] R. Baumann, Soft errors in advanced computer systems, IEEE Des. Test Comput. 22 (3) (2005) 258–266, http://dx.doi.org/10.1109/MDT.2005.69.
[27] M.-L. Li, P. Ramachandran, S.K. Sahoo, S.V. Adve, V.S. Adve, Y. Zhou, Understanding the propagation of hard errors to software and implications for resilient system design, ACM Sigplan Not. 43 (3) (2008) 265–276.
[28] V.G. Rao, H. Mahmoodi, Analysis of reliability of flip-flops under transistor aging effects in nano-scale CMOS technology, in: 2011 IEEE 29th International Conference on Computer Design (ICCD), IEEE, 2011, pp. 439–440.
[29] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, S. Gurumurthi, Feng shui of supercomputer memory positional effects in DRAM and sram faults, in: SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013, pp. 1–11.
[30] T. Siddiqua, V. Sridharan, S.E. Raasch, N. DeBardeleben, K.B. Ferreira, S. Levy, E. Baseman, Q. Guan, Lifetime memory reliability data from the field, in: 2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2017, pp. 1–6, http://dx.doi.org/10.1109/DFT.2017.8244428.
[31] J.L. Barth, C. Dyer, E. Stassinopoulos, Space, atmospheric, and terrestrial radiation environments, IEEE Trans. Nucl. Sci. 50 (3) (2003) 466–482.
[32] N.A. Dodds, Single Event Latchup: Hardening Strategies, Triggering Mechanisms, and Testing Considerations, Vanderbilt University, 2012.
[33] S. Di Mascio, A. Menicucci, G. Furano, T. Szewczyk, L. Campajola, F.D. Capua, A. Lucaroni, M. Ottavi, Towards defining a simplified procedure for COTS system-on-chip TID testing, Nuclear Engineering and Technology 50 (8) (2018) 1298–1305, http://dx.doi.org/10.1016/j.net.2018.07.010, URL http://www.sciencedirect.com/science/article/pii/S1738573318300585.
[34] A.L. Bogorad, J.J. Likar, R.E. Lombardi, S.E. Stone, R. Herschitz, On-orbit error rates of RHBD SRAMs: Comparison of calculation techniques and space environmental models with observed performance, IEEE Trans. Nucl. Sci. 58 (6) (2011) 2804–2806, http://dx.doi.org/10.1109/TNS.2011.2167242.

[35] L. Edmonds, C. Barnes, L. Scheick, An Introduction to Space Radiation Effects on Microelectronics, Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2000.

[36] D. Heynderickx, B. Quaghebeur, J. Wera, E.J. Daly, H.D.R. Evans, New radiation environment and effects models in the european space agency's space environment information system (SPENVIS), Space Weather 2 (10) (2004) 1–4, http://dx.doi.org/10.1029/2004SW000073.

[37] V. Malherbe, G. Gasiot, D. Soussan, J. Autran, P. Roche, On-orbit upset rate prediction at advanced technology nodes: a 28 nm FD-SOI case study, IEEE Trans. Nucl. Sci. 64 (1) (2017) 449–456, http://dx.doi.org/10.1109/TNS.2016.2634604.

[38] J.A. Pellish, M.A. Xapsos, C.A. Stauffer, T.M. Jordan, A.B. Sanders, R.L. Ladbury, T.R. Oldham, P.W. Marshall, D.F. Heidel, K.P. Rodbell, Impact of spacecraft shielding on direct ionization soft error rates for sub-130 nm technologies, IEEE Trans. Nucl. Sci. 57 (6) (2010) 3183–3189, http://dx.doi.org/10.1109/TNS.2010.2084595.

[39] H. Michel, H. Guzmán-Miranda, A. Dörflinger, H. Michalik, M.A. Echanove, SEU fault classification by fault injection for an FPGA in the space instrument SOPHI, in: 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2017, pp. 9–15, http://dx.doi.org/10.1109/AHS.2017.8046353.

[40] C. Fong, S. Yang, C. Chu, C. Huang, J. Yeh, C. Lin, T. Kuo, T. Liu, N.L. Yen, S. Chen, Y. Kuo, Y. Liou, S. Chi, Formosat-3/cosmic constellation spacecraft system performance: After one year in orbit, IEEE Trans. Geosci. Remote Sens. 46 (11) (2008) 3380–3394, http://dx.doi.org/10.1109/TGRS.2008.2005203.

[41] X. Li, S.V. Adve, P. Bose, J.A. Rivers, Architecture-level soft error analysis: Examining the limits of common assumptions, in: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), 2007, pp. 266–275, http://dx.doi.org/10.1109/DSN.2007.15.

[42] R. García Alía, M. Brugger, E. Daly, S. Danzeca, V. Ferlet-Cavrois, R. Gaillard, J. Mekki, C. Poivey, A. Zadeh, Simplified SEE sensitivity screening for COTS components in space, IEEE Trans. Nucl. Sci. 64 (2) (2017) 882–890, http://dx.doi.org/10.1109/TNS.2017.2653863.

[43] B.D. Sierawski, R.A. Reed, K.M. Warren, A.L. Sternberg, R.A. Austin, J.M. Trippe, R.A. Weller, M.L. Alles, R.D. Schrimpf, L.W. Massengill, D.M. Fleetwood, G.W. Buxton, J.C. Brandenburg, W.B. Fisher, R. Davis, Cubesat: Real-time soft error measurements at low earth orbits, in: 2017 IEEE International Reliability Physics Symposium (IRPS), 2017, pp. 3D–1.1–3D–1.6, http://dx.doi.org/10.1109/IRPS.2017.7936291.

[44] F. Zaruba, L. Benini, The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm fdsoi technology, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27 (11) (2019) 2629–2640.

[45] G. Hubert, L. Artola, D. Regis, Impact of scaling on the soft error sensitivity of bulk, FDSOI and finfet technologies due to atmospheric radiation, Integration 50 (2015) 39–47, http://dx.doi.org/10.1016/j.vlsi.2015.01.003, URL http://www.sciencedirect.com/science/article/pii/S0167926015000048.

[46] D.L. Weaver, OpenSPARC Internals: OpenSPARC T1/T2 CMT throughput computing, Sun Microsystems, 2008.

[47] A. Dixit, A. Wood, The impact of new technology on soft error rates, in: 2011 International Reliability Physics Symposium, 2011, pp. 5B.4.1–5B.4.7, http://dx.doi.org/10.1109/IRPS.2011.5784522.

[48] S. Buchner, M. Baze, D. Brown, D. McMorrow, J. Melinger, Comparison of error rates in combinational and sequential logic, IEEE Trans. Nucl. Sci. 44 (6) (1997) 2209–2216, http://dx.doi.org/10.1109/23.659037.

[49] S. Jagannathan, T.D. Loveless, B.L. Bhuva, N.J. Gaspard, N. Mahatme, T. Assis, S.. Wen, R. Wong, L.W. Massengill, Frequency dependence of alpha-particle induced soft error rates of flip-flops in40-nm CMOS technology, IEEE Trans. Nucl. Sci. 59 (6) (2012) 2796–2802, http://dx.doi.org/10.1109/TNS.2012.2223827.

[50] STMicroelectronics, A 65nm hardened ASIC technology for Space applications, URL https://indico.esa.int/event/165/contributions/1218/attachments/1205/1425/05b_-_KIPSAT_-_Presentation.pdf.

[51] S. Clerc, F. Abouzeid, G. Gasiot, J. Daveau, C. Bottoni, M. Glorieux, J. Autran, F. Cacho, V. Huard, L. Dugoujon, R. Weigand, F. Malou, L. Hili, P. Roche, Space radiation and reliability qualifications on 65nm CMOS 600mhz microprocessors, in: 2013 IEEE International Reliability Physics Symposium (IRPS), 2013, pp. 6C.1.1–6C.1.7, http://dx.doi.org/10.1109/IRPS.2013.6532051.

[52] R.M. Goodman, M. Sayano, The reliability of semiconductor RAM memories with on-chip error-correction coding, IEEE Trans. Inform. Theory 37 (3) (1991) 884–896.

[53] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, F. Ruckerbauer, Investigation of increased multi-bit failure rate due to neutron induced SEU in advanced embedded SRAMs, in: 2007 IEEE Symposium on VLSI Circuits, 2007, pp. 80–81, http://dx.doi.org/10.1109/VLSIC.2007.4342774.

[54] M. Raine, G. Hubert, M. Gaillardin, P. Paillet, A. Bournel, Monte Carlo prediction of heavy ion induced MBU sensitivity for SOI SRAMs using radial ionization profile, IEEE Trans. Nucl. Sci. 58 (6) (2011) 2607–2613, http://dx.doi.org/10.1109/TNS.2011.2168238.

[55] M.J. Gadlage, R.D. Schrimpf, J.M. Benedetto, P.H. Eaton, D.G. Mavis, M. Sibley, K. Avery, T.L. Turflinger, Single event transient pulse widths in digital microcircuits, IEEE Trans. Nucl. Sci. 51 (6) (2004) 3285–3290, http://dx.doi.org/10.1109/TNS.2004.839174.

[56] R.C. Harrington, J.S. Kauppila, K.M. Warren, Y.P. Chen, J.A. Maharrey, T.D. Haeffner, T.D. Loveless, B.L. Bhuva, M. Bounasser, K. Lilja, L.W. Massengill, Estimating single-event logic cross sections in advanced technologies, IEEE Trans. Nucl. Sci. 64 (8) (2017) 2115–2121, http://dx.doi.org/10.1109/TNS.2017.2718517.

[57] R.C. Baumann, Radiation-induced soft errors in advanced semiconductor technologies, IEEE Trans. Device Mater. Reliab. 5 (3) (2005) 305–316, http://dx.doi.org/10.1109/TDMR.2005.853449.

[58] N.N. Mahatme, S. Jagannathan, T.D. Loveless, L.W. Massengill, B.L. Bhuva, S.. Wen, R. Wong, Comparison of combinational and sequential error rates for a deep submicron process, IEEE Trans. Nucl. Sci. 58 (6) (2011) 2719–2725, http://dx.doi.org/10.1109/TNS.2011.2171993.

[59] J. Velamala, R. LiVolsi, M. Torres, Y. Cao, Design sensitivity of single event transients in scaled logic circuits, in: 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), 2011, pp. 694–699.

[60] A. Ramos, J.A. Maestro, P. Reviriego, Characterizing a RISC-V SRAM-based FPGA implementation against single event upsets using fault injection, Microelectron. Reliab. 78 (2017) 205–211, http://dx.doi.org/10.1016/j.microrel.2017.09.007, URL http://www.sciencedirect.com/science/article/pii/S0026271417304493.

[61] C. Bernardeschi, L. Cassano, A. Domenici, L. Sterpone, Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs, in: 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012, pp. 115–120, http://dx.doi.org/10.1109/DFT.2012.6378210.

[62] J.A. Butts, G. Sohi, Dynamic dead-instruction detection and elimination, ACM SIGPLAN Not. 37 (10) (2002) 199–210.

[63] H. Cho, S. Mirkhani, C. Cher, J.A. Abraham, S. Mitra, Quantitative evaluation of soft error injection techniques for robust system design, in: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), 2013, pp. 1–10.

[64] M. Richter, K. Oberlaender, M. Goessel, New linear SEC-ded codes with reduced triple bit error miscorrection probability, in: 2008 14th IEEE International on-Line Testing Symposium, 2008, pp. 37–42, http://dx.doi.org/10.1109/IOLTS.2008.27.

[65] S. Mukherjee, J. Emer, S.K. Reinhardt, The soft error problem: an architectural perspective, in: 11th International Symposium on High-Performance Computer Architecture, 2005, pp. 243–247, http://dx.doi.org/10.1109/HPCA.2005.37.

[66] G.S. Rodrigues, F. Rosa, Á.B. de Oliveira, F.L. Kastensmidt, L. Ost, R. Reis, Analyzing the impact of fault-tolerance methods in ARM processors under soft errors running linux and parallelization APIs, IEEE Trans. Nucl. Sci. 64 (8) (2017) 2196–2203, http://dx.doi.org/10.1109/TNS.2017.2706519.

[67] H. Cho, Impact of microarchitectural differences of RISC-V processor cores on soft error effects, IEEE Access 6 (2018) 41302–41313.

[68] M.A. Breuer, Multi-media applications and imprecise computation, in: 8th Euromicro Conference on Digital System Design (DSD'05), 2005, pp. 2–7, http://dx.doi.org/10.1109/DSD.2005.58.

[69] D.D. Thaker, D. Franklin, J. Oliver, S. Biswas, D. Lockhart, T. Metodi, F.T. Chong, Characterization of error-tolerant applications when protecting control data, in: 2006 IEEE International Symposium on Workload Characterization, 2006, pp. 142–149, http://dx.doi.org/10.1109/IISWC.2006.302738.

[70] G. Li, S.K.S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, S.W. Keckler, Understanding error propagation in deep learning neural network (dnn) accelerators and applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 8, http://dx.doi.org/10.1145/3126908.3126964.

[71] J. Perez Acle, M.S. Reorda, M. Violante, Implementing a safe embedded computing system in SRAM-based fpgas using IP cores: A case study based on the altera NIOS-II soft processor, in: 2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS), 2011, pp. 1–5, http://dx.doi.org/10.1109/LASCAS.2011.5750278.

[72] N.J. Wang, S.J. Patel, Restore: symptom based soft error detection in microprocessors, in: 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005, pp. 30–39, http://dx.doi.org/10.1109/DSN.2005.82.

[73] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, T. Austin, A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, in: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, in: MICRO 36, IEEE Computer Society, Washington, DC, USA, 2003, p. 29, URL http://dl.acm.org/citation.cfm?id=956417.956570.

[74] D.S. Kim, S.M. Lee, J.-H. Jung, T.H. Kim, S. Lee, J.S. Park, Reliability and availability analysis for an on board computer in a satellite system using standby redundancy and rejuvenation, J. Mech. Sci. Technol. 26 (7) (2012) 2059–2063, http://dx.doi.org/10.1007/s12206-012-0512-6.

[75] F.M. David, J.C. Carlyle, E.M. Chan, D.K. Raila, R.H. Campbell, Exception handling in the choices operating system, in: Advanced Topics in Exception Handling Techniques, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 42–61, http://dx.doi.org/10.1007/11818502_3.

[76] I. Joe, S.C. Lee, Bootup time improvement for embedded linux using snapshot images created on boot time, in: The 2nd International Conference on Next Generation Information Technology, 2011, pp. 193–196.

[77] N.N. Sadler, D.J. Sorin, Choosing an error protection scheme for a microprocessor's l1 data cache, in: 2006 International Conference on Computer Design, 2006, pp. 499–505, http://dx.doi.org/10.1109/ICCD.2006.4380862.

[78] J. Gaisler, A portable and fault-tolerant microprocessor based on the SPARC v8 architecture, in: Proceedings International Conference on Dependable Systems and Networks, 2002, pp. 409–415, http://dx.doi.org/10.1109/DSN.2002.1028926.

[79] Á.B. de Oliveira, G.S. Rodrigues, F.L. Kastensmidt, N. Added, E.L.A. Macchione, V.A.P. Aguiar, N.H. Medina, M.A.G. Silveira, Lockstep dual-core ARM A9: Implementation and resilience analysis under heavy ion-induced soft errors, IEEE Trans. Nucl. Sci. 65 (8) (2018) 1783–1790, http://dx.doi.org/10.1109/TNS.2018.2852606.

[80] X. Iturbe, B. Venu, E. Ozer, Soft error vulnerability assessment of the real-time safety-related ARM cortex-r5 CPU, in: 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2016, pp. 91–96, http://dx.doi.org/10.1109/DFT.2016.7684076.

[81] M. Maniatakos, P. Kudva, B.M. Fleischer, Y. Makris, Low-cost concurrent error detection for floating-point unit (FPU) controllers, IEEE Trans. Comput. 62 (7) (2013) 1376–1388, http://dx.doi.org/10.1109/TC.2012.81.

[82] S. Tselonis, M. Kaliorakis, N. Foutris, G. Papadimitriou, D. Gizopoulos, Microprocessor reliability-performance tradeoffs assessment at the microarchitecture level, in: 2016 IEEE 34th VLSI Test Symposium (VTS), 2016, pp. 1–6, http://dx.doi.org/10.1109/VTS.2016.7477300.

[83] X. Fu, T. Li, J. Fortes, Sim-SODA: Unified framework for architectural level software reliability analysis, in: Workshop on Modeling, Benchmarking and Simulation, 2006, 2006.

[84] A. Chatzidimitriou, D. Gizopoulos, Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy, in: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016, pp. 69–78, http://dx.doi.org/10.1109/ISPASS.2016.7482075.

[85] L. Duan, L. Peng, B. Li, Predicting architectural vulnerability on multithreaded processors under resource contention and sharing, IEEE Trans. Dependable Secure Comput. 10 (2) (2013) 114–127, http://dx.doi.org/10.1109/TDSC.2012.87.

[86] N. Muralimanohar, R. Balasubramonian, N.P. Jouppi, CACTI 6.0: A tool to model large caches, HP Lab. 27 (2009) 28.

[87] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, C. Monteleone, On the criticality of caches in fault tolerant processors for space, in: 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019, http://dx.doi.org/10.1109/DFT.2019.8875424.

[88] A. Naithani, S. Eyerman, L. Eeckhout, Optimizing soft error reliability through scheduling on heterogeneous multicore processors, IEEE Trans. Comput. 67 (6) (2018) 830–846, http://dx.doi.org/10.1109/TC.2017.2779480.

[89] L. Duan, Y. Zhang, B. Li, L. Peng, Comprehensive and efficient design parameter selection for soft error resilient processors via universal rules, IEEE Trans. Comput. 63 (9) (2014) 2201–2214, http://dx.doi.org/10.1109/TC.2013.24.

[90] A. Biswas, C. Recchia, S.S. Mukherjee, V. Ambrose, L. Chan, A. Jaleel, A.E. Papathanasiou, M. Plaster, N. Seifert, Explaining cache SER anomaly using DUE AVF measurement, in: HPCA - 16 2010 the Sixteenth International Symposium on High-Performance Computer Architecture, 2010, pp. 1–12, http://dx.doi.org/10.1109/HPCA.2010.5416629.

[91] Y. Cheng, A. Ma, Y. Tang, M. Zhang, Accurate vulnerability estimation for cache hierarchy, in: The 7th International Conference on Networked Computing and Advanced Information Management, 2011, pp. 7–14.

[92] M. Kooli, G. Di Natale, A. Bosio, Memory-aware design space exploration for reliability evaluation in computing systems, J. Electron. Test. 35 (2) (2019) 145–162, http://dx.doi.org/10.1007/s10836-019-05785-0.

[93] X. Fu, J. Poe, T. Li, J.A.B. Fortes, Characterizing microarchitecture soft error vulnerability phase behavior, in: 14th IEEE International Symposium on Modeling, Analysis, and Simulation, 2006, pp. 147–155, http://dx.doi.org/10.1109/MASCOTS.2006.18.

[94] W. Zhang, T. Li, Managing multi-core soft-error reliability through utility-driven cross domain optimization, in: 2008 International Conference on Application-Specific Systems, Architectures and Processors, 2008, pp. 132–137, http://dx.doi.org/10.1109/ASAP.2008.4580167.

[95] T.M. Jones, M.F. O'boyle, O. Ergin, Evaluating the effects of compiler optimisations on AVF, in: Workshop on Interaction Between Compilers and Computer Architecture (INTERACT-12), 2008.

[96] G.E. Medeiros, F.T. Bortolon, R. Reis, L. Ost, Evaluation of compiler optimization flags effects on soft error resiliency, in: 2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI), 2018, pp. 1–6, http://dx.doi.org/10.1109/SBCCI.2018.8533246.

[97] A. Serrano-Cases, Y. Morilla, P. Martín-Holgado, S. Cuenca-Asensi, A. Martínez-Álvarez, Nonintrusive automatic compiler-guided reliability improvement of embedded applications under proton irradiation, IEEE Trans. Nucl. Sci. 66 (7) (2019) 1500–1509, http://dx.doi.org/10.1109/TNS.2019.2912323.

[98] C. Cher, M.S. Gupta, P. Bose, K.P. Muller, Understanding soft error resiliency of blue gene/q compute chip through hardware proton irradiation and software fault injection, in: SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 587–596, http://dx.doi.org/10.1109/SC.2014.53.

[99] Y. Chou, B. Fahs, S. Abraham, Microarchitecture optimizations for exploiting memory-level parallelism, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, in: ISCA '04, IEEE Computer Society, Washington, DC, USA, 2004, p. 76, URL http://dl.acm.org/citation.cfm?id=998680.1006708.

[100] N.J. Wang, A. Mahesri, S.J. Patel, Examining ACE analysis reliability estimates using fault-injection, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, in: ISCA '07, ACM, New York, NY, USA, 2007, pp. 460–469, http://dx.doi.org/10.1145/1250662.1250719, URL http://doi.acm.org/10.1145/1250662.1250719.

[101] N. Wang, M.F. and, Y-branches: when you come to a fork in the road, take it, in: 2003 12th International Conference on Parallel Architectures and Compilation Techniques, 2003, pp. 56–66, http://dx.doi.org/10.1109/PACT.2003.1238002.

[102] A. Biswas, P. Racunas, J. Emer, S. Mukherjee, Computing accurate AVFs using ACE analysis on performance models: A rebuttal, IEEE Comput. Arch. Lett. 7 (1) (2008) 21–24, http://dx.doi.org/10.1109/L-CA.2007.19.

[103] K. Tanikella, Y. Koy, R. Jeyapaul, K. Lee, A. Shrivastava, Gemv: A validated toolset for the early exploration of system reliability, in: 2016 IEEE 27th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), 2016, pp. 159–163.

[104] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, P. Sanda, Statistical fault injection, in: 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN), 2008, pp. 122–127, http://dx.doi.org/10.1109/DSN.2008.4630080.

[105] H. Ziade, R. Ayoubi, R. Velazco, A survey on fault injection techniques, International Arab Journal of Information Technology Vol. 1, No. 2, July (2004) 171–186.

[106] P. Montesinos, W. Liu, J. Torrellas, Using register lifetime predictions to protect register files against soft errors, in: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), IEEE, 2007, pp. 286–296.

[107] A.-G. Ma, Y. Cheng, Z.-C. Xing, Accurate and simplified prediction of AVF for delay and energy efficient cache design, J. Comput. Sci. Tech. 26 (3) (2011) 504–519, http://dx.doi.org/10.1007/s11390-011-1150-7.

[108] J.A. Blome, S. Gupta, S. Feng, S. Mahlke, Cost-efficient soft error protection for embedded microprocessors, in: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, in: CASES '06, ACM, New York, NY, USA, 2006, pp. 421–431, http://dx.doi.org/10.1145/1176760.1176811, URL http://doi.acm.org/10.1145/1176760.1176811.

[109] X. Li, S.V. Adve, P. Bose, J.A. Rivers, Softarch: an architecture-level tool for modeling and analyzing soft errors, in: 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005, pp. 496–505, http://dx.doi.org/10.1109/DSN.2005.88.

[110] R.W. Hamming, Error detecting and error correcting codes, The Bell Syst. Tech. J. 29 (2) (1950) 147–160.

[111] H.Q. Le, W.J. Starke, J.S. Fields, F.P. O'Connell, D.Q. Nguyen, B.J. Ronchetti, W.M. Sauer, E.M. Schwarz, M.T. Vaden, IBM POWER6 microarchitecture, IBM J. Res. Dev. 51 (6) (2007) 639–662, http://dx.doi.org/10.1147/rd.516.0639.

[112] M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, A. Wynn, Ultrasparc T2: A highly-treaded, power-efficient, SPARC SOC, in: 2007 IEEE Asian Solid-State Circuits Conference, 2007, pp. 22–25, http://dx.doi.org/10.1109/ASSCC.2007.4425786.

[113] N. Quach, High availability and reliability in the itanium processor, IEEE Micro 20 (5) (2000) 61–69, http://dx.doi.org/10.1109/40.877951.

[114] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, H.-U. Zurek, The arm triple core lock-step (TCLS) processor, ACM Trans. Comput. Syst. 36 (3) (2019).

[115] I. Alam, C. Schoeny, L. Dolecek, P. Gupta, Parity++: Lightweight error correction for last level caches, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2018, pp. 114–120, http://dx.doi.org/10.1109/DSN-W.2018.00048.

[116] S. Cha, H. Yoon, Efficient implementation of single error correction and double error detection code with check bit pre-computation for memories, JSTS: J. Semicond. Technol. Sci. 12 (4) (2012) 418–425.
[117] P. Reviriego, J.A. Maestro, S. Baeg, S. Wen, R. Wong, Protection of memories suffering MCUs through the selection of the optimal interleaving distance, IEEE Trans. Nucl. Sci. 57 (4) (2010) 2124–2128, http://dx.doi.org/10.1109/TNS.2010.2042818.
[118] H.-J. Tsai, C.-C. Chen, K.-H. Yang, T.-C. Yang, L.-Y. Huang, C.-H. Chung, M.-F. Chang, T.-F. Chen, Leveraging data lifetime for energy-aware last level non-volatile sram caches using redundant store elimination, in: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), 2014, pp. 1–6, http://dx.doi.org/10.1145/2593069.2593153.
[119] M. Focardi, S. Pezzuto, R. Cosentino, G. Giusi, M. Pancrazzi, V. Noce, R. Ottensamer, M. Steller, A.M.D. Giorgio, E. Pace, P. Plasson, G. Peter, I. Pagano, The instrument control unit of the ESA-PLATO 2.0 mission, in: H.A. MacEwen, G.G. Fazio, M. Lystrup, N. Batalha, N. Siegler, E.C. Tong (Eds.), Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave, 9904, SPIE, 2016, pp. 962–976, http://dx.doi.org/10.1117/12.2231658.
[120] F.A. Endo, D. Couroussé, H.-P. Charles, Micro-architectural simulation of embedded core heterogeneity with gem5 and mcpat, in: Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, in: RAPIDO '15, ACM, New York, NY, USA, 2015, pp. 7:1–7:6, http://dx.doi.org/10.1145/2693433.2693440, URL http://doi.acm.org/10.1145/2693433.2693440.
[121] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures, in: 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009, pp. 469–480.
[122] W. Zhang, Computing cache vulnerability to transient errors and its implication, in: 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05), 2005, pp. 427–435, http://dx.doi.org/10.1109/DFTVS.2005.23.
[123] R. Yamamoto, C. Hamanaka, J. Furuta, K. Kobayashi, H. Onodera, An area-efficient 65 nm radiation-hard dual-modular flip-flop to avoid multiple cell upsets, IEEE Trans. Nucl. Sci. 58 (6) (2011) 3053–3059, http://dx.doi.org/10.1109/TNS.2011.2169457.
[124] M.M. Ghahroodi, E. Ozer, D. Bull, SEU and SET-tolerant ARM Cortex-R4 CPU for space and avionics applications, in: Proc. of the Workshop on Manufacturable and Dependable Multi-Core Architectures at Nanoscale, 2013.
[125] K. Kobayashi, K. Kubota, M. Masuda, Y. Manzawa, J. Furuta, S. Kanda, H. Onodera, A low-power and area-efficient radiation-hard redundant flip-flop, DICE acff, in a 65 nm thin-BOX FD-SOI, IEEE Trans. Nucl. Sci. 61 (4) (2014) 1881–1888, http://dx.doi.org/10.1109/TNS.2014.2318326.
[126] C. Bottoni, B. Coeffic, J. Daveau, L. Naviner, P. Roche, Partial triplication of a SPARC-v8 microprocessor using fault injection, in: 2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS), 2015, pp. 1–4, http://dx.doi.org/10.1109/LASCAS.2015.7250415.
[127] A. Evans, C.U. Ortega, K. Marinis, E. Costenaro, H. Laroussi, K.-V. Obbe, G. Magistrati, V. Ferlet-Cavrois, Heavy-ion micro beam and simulation study of a flash-based fpga microcontroller implementation, IEEE Trans. Nucl. Sci. 64 (1) (2017) 504–511, http://dx.doi.org/10.1109/TNS.2016.2633401.
[128] X. He, X. Huang, Y. Li, TMR-Based soft error tolerance techniques in ASIC design, in: 2016 6th International Conference on Advanced Design and Manufacturing Engineering (ICADME 2016), Atlantis Press, 2017, http://dx.doi.org/10.2991/icadme-16.2016.76.
[129] D.L. Hansen, E.J. Miller, A. Kleinosowski, K. Kohnen, A. Le, D. Wong, K. Amador, M. Baze, D. DeSalvo, M. Dooley, K. Gerst, B. Hughlock, B. Jeppson, R.D. Jobe, D. Nardi, I. Ojalvo, B. Rasmussen, D. Sunderland, J. Truong, M. Yoo, E. Zayas, Clock, flip-flop, and combinatorial logic contributions to the seu cross section in 90 nm ASIC technology, IEEE Trans. Nucl. Sci. 56 (6) (2009) 3542–3550, http://dx.doi.org/10.1109/TNS.2009.2031972.
[130] B. Venu, E. Ozer, X. Iturbe, A. Robinson, A fail-functional automotive CPU subsystem architecture for mitigating single point of failures, in: Proceedings of the IEEE International Workshop on Automotive Reliability and Test, 2016.
[131] F. Sturesson, J. Gaisler, R. Ginosar, T. Liran, Radiation characterization of a dual core LEON3-FT processor, in: 2011 12th European Conference on Radiation and Its Effects on Components and Systems, 2011, pp. 938–944, http://dx.doi.org/10.1109/RADECS.2011.6131334.
[132] P.N. Sanda, J.W. Kellington, P. Kudva, R. Kalla, R.B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, C.R. Jones, Soft-error resilience of the IBM POWER6 processor, IBM J. Res. Dev. 52 (3) (2008) 275–284, http://dx.doi.org/10.1147/rd.523.0275.