Evaluation of Fault Mitigation Techniques Based on Approximate Computing Under Radiation

A. Martínez-Álvarez[1], D. González-Montesoro[1], A. Serrano-Cases[1], A. Aponte-Moreno[2], F. Restrepo-Calle[2], Y. Morilla[3], P. Martín-Holgado[3], and S. Cuenca-Asensi[1]

***Abstract*—A software technique based on approximate computing and redundancy is presented to mitigate radiation-induced soft errors in COTS microprocessors. Approximate Computing relies on the capability of certain applications to accept imprecise results to improve efficiency by sacrificing its results in a controlled manner. Our approach avoids the time overhead derived from hardening while preserving the detection and correction rate. Experimental results show that we can detect and correct SDC events improving the cross-section up to 160×, and keeping accuracy under control without compromising performance. In addition, an accuracy-aware layer is included to improve error mitigation and to provide a trade-off between the number of tolerable errors and the necessary accuracy.**

***Index Terms*—Approximate computing, ARM, proton irradiation, fault tolerance.**

I. Introduction

Commercial off-the-shelf (COTS) hardware devices continue gaining popularity when fault tolerance is a requirement. Given the nature of such systems, software techniques define the most evident source of improved reliability. In this context, traditional triple modular redundancy (TMR) techniques are the most common method to provide protection. However, software-based fault tolerance techniques, of any kind, are always applied at a cost. Indeed, they involve several unwanted overheads, such as execution time and memory usage, which must be limited and kept under control, especially when applied to embedded systems.

In cases where accuracy requisites can be relaxed, the usage of approximate computing techniques (AC) can shrink time limits to a minimum. Recently, approximate computing techniques have been employed to design fault-tolerant systems with a reduced time overhead [1], [2]. The AC paradigm improves the performance (and consequently also the time overhead) and energy efficiency of a given software at the cost of introducing some inaccuracy at the output [3]. However, it is essential to ensure that the approximated output remains compatible with precise software results while keeping accuracy under control.

In this paper, we propose and assess, under proton radiation, a modified version of the AC-based fault tolerance technique introduced in [4]. Our technique combines redundancy and approximate computing, both software-based, to further minimize the time overhead inherent to SIHFT (Software Implemented Hardware Fault Tolerance) techniques. They have the advantage that they can be used in COTS microprocessor devices where no hardware modifications can be achieved. An extensive case study is presented where critical computation is defined by the well-known robotic translations `inverse2kj` and `forward2kj`. These algorithms have applications in robotic arms on terrestrial and in space [4]; for instance, they are intended to perform calculations on a robotic arm with two degrees of freedom to return the position of the robotic arm in cartesian coordinates as well as in particular coordinates using trigonometric operations. The benchmarks are hardened using a combination of temporal redundancy and different levels of AC. The device under test (DUT) was a 28nm Xilinx Zynq-7010 System on a Chip (SoC) integrating an ARM Cortex-A9 processor. The system was irradiated with $15.3MeV$ protons. The results show that the technique is perfectly suitable for detecting and correcting single-event faults while keeping the accuracy under control and without examining performance overheads.

II. Background and Related Works

Approximate Computing relies on the ability of certain applications to accept imprecise results to enhance efficiency or improve energy consumption [3]. Approximate Computing techniques are classified into different levels: from Hardware, where energy consumption or circuit area can be reduced, to Software, where the performance of an application can be improved by sacrificing its result in a controlled manner.

Due to the benefits of using AC techniques to reduce energy consumption or improve the performance of a system, researchers have extended the use of approximate computing to the design of fault-tolerant systems to reduce the overheads associated with hardening [2].

Most of the work using Approximate Computing in the design of fault-tolerant systems is at the circuit level. For instance, hardware TMR proposals using approximate logic

The research reported in this paper has been partially supported through the following projects: MultiRad (funded by Région Auvergne-Rhône-Alpes, France); IRT Nanoelec (French National Research Agency ANR-10-AIRT-05 project funded through the Program d'investissement d'avenir); UGA/LPSC/-GENESIS platform and `PID2022-138696OB-C22` (funded by the Spanish Ministry of Science and Innovation).

[1]A. Martínez-Álvarez, D. González-Montesoro, A. Serrano-Cases and S. Cuenca-Asensi are staff members at the Computer Technology Department, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain (e-mail: {amartinez, dgonzalez, aserrano, sergio}@dtic.ua.es.)

[2] A. Aponte-Moreno and F. Restrepo-Calle are with Department of Systems and Industrial Engineering, Universidad Nacional de Colombia, Bogotá D.C., Colombia. (e-mail: jaapontem@unal.edu.co, ferestrepoca@unal.edu.co).

[3]Y. Morilla and Pedro Martin-Holgado are with the Centro Nacional de Aceleradores (CNA), Centro Nacional de Aceleradores, CSIC, JA, Universidad de Seville, E-41092 Seville, Spain, SPAIN (e-mail: ymorilla@us.es, pmartinholgado@us.es).

circuits [5]–[8] or Quadruple Approximate Modular Redundancy (QAMR) schemes [9]–[11] have been presented. In these works, good results are achieved (fault coverage and reduction of area overheads).

Specific architectural designs have also been presented to reduce the overheads associated with hardening. For example, the use of components with different levels of reliability has been proposed; the most precise cores are responsible for executing the most sensitive parts of the application [12].

On the other hand, there have been works where the multi-core feature of systems is exploited to implement NMR (N Modular Redundancy) with AC. In [13], a proposal called LEXACT is introduced, which relies on executing both exact and approximate versions of a task on different processor cores. The approximate versions have a shorter execution time, allowing multiple approximate tasks to be executed on the same core.

Duplication with comparison (DWC) with approximation for error detection, has been proposed for software solutions applicable to COTS devices [14], [15]. In these cases, the precision is reduced to compensate for the overhead associated with duplication.

In [16], the authors propose a framework called FTxAC, based on Approximate Computing, for the design of fault-tolerant systems with reduced overheads. The proposal was validated through several software-level case studies, demonstrating that program approximation not only reduces overheads but also improves reliability.

Finally, it is worth mentioning that some studies have included experimental evaluations of novel approaches to improve the reliability and efficiency of approximate computing systems. The work presented in [17] deals with the impact of neutron irradiation on approximate computing techniques applied to the data representation of Convolutional Neural Networks (CNNs). The work presented in [18] proposes a software emulator capable of injecting real faults from radiation tests into CNNs, providing a more efficient and versatile method for reliability assessment compared to traditional statistical fault injection methods. Furthermore, [19] proposes Approximate Triple Modular Redundancy (ATMR) to mitigate multi-bit upsets in embedded software and shows that ATMR effectively balances execution time overhead and fault masking rate, offering a promising approach to improve system resilience. The experiments involve exposing an ARM Cortex A9 processor to laser pulses in the data cache memory area to induce bit flips. Lastly, a neutron beam experiment is performed in [20] to assess the vulnerability of a Kepler GPU to transient effects induced by radiation. In addition, a fault injection campaign is performed in the same work to identify critical registers and improve the reliability of the GPU register file by relaxing application accuracy.

III. Approximate Computing Mitigation Approach

The AC mitigation technique proposed in [4] consists of approximating the computational calculation of a given program before applying hardening strategies (ie, traditional TMR in that work) to compensate for overheads associated with fault tolerance. To minimize these overheads, the authors introduced a new AC technique called *simplified iterations*, which is a variation of the *loop perforation* technique.

The conventional *loop perforation* technique involves reducing the number of iterations in a loop. This approach decreases the execution time while compromising the result's precision in a controlled manner. However, the applicability of the *loop perforation* technique is limited to algorithms of iterative nature. Not all algorithms with long loops are suitable candidates for an accurate approximation using this technique. For instance, programs that fulfil identical calculations on a large set of input data may employ loops with as many iterations as there are inputs to execute operations on each of them. Skipping iterations in such programs can result in abandoning calculations on specific inputs, potentially leading to significant approximation errors.

Hence, to uphold an acceptable level of imprecision, the authors suggest refraining from employing the traditional *loop perforation* method, which involves skipping entire iterations. Instead, they propose expediting the computations conducted within the previously ommited iterations by performing operations of reduced complexity that yield results closely approximating the original calculations. To achieve this, a thorough functional analysis of the loop's internal processes is essential. This analysis will aid in devising a function whose approximate calculations can be executed efficiently without significantly deviating from the precise outcomes. This functional analysis is imperative as the calculation's simplification hinges on the operations carried out within the loop's body.

The approximation technique is represented in Figure 1. At the top of the figure, a traditional loop representation is shown, where the calculations are repeated *n* times. At the bottom, the approximate loop is depicted using the *simplified iterations* technique.

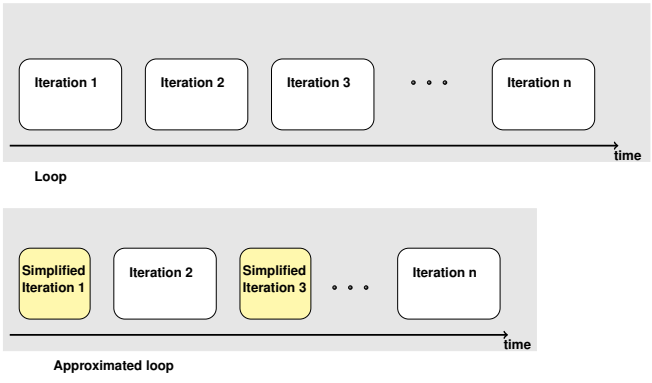

Fig. 1. Simplified iterations technique

The implementation of *simplified iterations* is illustrated in Figure 2. The upper section shows the code structure of a typical for-loop, while the bottom section shows the application of *simplified iterations* on the same loop. The `original_iteration()` function represents the precise calculations, while the approximated ones are represented by the `approximated_calculations()` function.

For loop:

```
for i ← 0 to N do
    original_calculations()
end for
```

Approximated for loop:

```
for i ← 0 to N do
    if Non-approximate iteration then
        original_calculations()
    else
        approximated_calculations()
    end if
end for
```

Fig. 2. Snippet of pseudo-code for *simplified iterations* technique

In this study, the authors propose using the DWCF (Duplication With Comparison) fault mitigation technique [21] in conjunction with the simplified iterations AC technique, instead of traditional TMR. DWCF is an advanced version of conventional DWC that can also correct faults. In DWCF, the program is executed twice, and the results are compared; in addition, a third program execution is triggered only if the result comparison does not match; finally, a second comparison can be performed to correct any faults using a majority voter with the results from the three program executions. Figure 3 shows how DWCF works in its original and approximate versions.

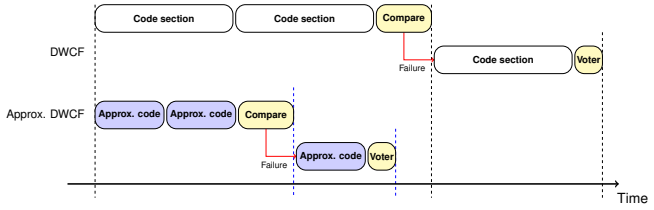

Fig. 3. DWCF with simplified iterations approximation.

The use of AC modules results in a noticeable reduction in execution time. Additionally, the time is further minimized at the end of the first comparison if the results from the first two executions match.

The reduction in execution time depends on the degree of approximation achieved by the AC technique used. In this case (through simplified iterations), the degree of approximation is related to the number of iterations on the original loop that are replaced with simplified calculations, and how much the operations in the iteration can be approximated. Therefore, by changing the number of approximate iterations, it is possible to obtain different levels of approximation for the same algorithm.

Compared to previous works, this study stands out for its applicability to Commercial Off-The-Shelf (COTS) devices, as it is software-based, enabling error detection and correction. Although software-level proposals have been presented, they solely focus on error detection and offer solutions to specific problems. Furthermore, it is crucial to emphasize that the validation of this study was conducted through radiation experiments.

IV. Test bench approximation

The test bench for the radiation experiments was comprised of two selected test programs, `Inversek2j (IN)` and `Forwardk2j (FW)`, which were obtained from the *AxBench* benchmark suite for approximate computing [22]. Both algorithms are used in robot kinematics with two degrees of freedom and are intended to calculate the position of a robotic arm. These tests are essential as robotic arms are not only limited to terrestrial use but also have the potential for operation in space [23], where there is a significant amount of radiation exposure.

`Forwardk2j` (forward kinematics) calculates the position of the end of the arm in space, based on the angles of the joints. In this study, we consider a two-jointed robotic arm with lengths l_1 and l_2. The arm's position, expressed as the Cartesian plane coordinates (x, y), is calculated based on the angles θ_1 and θ_2, as illustrated in the equations 1 and 2.

$$x = l_1 \cdot cos(\theta_1) + l_2 \cdot cos(\theta_1 + \theta_2) \quad (1)$$

$$y = l_1 \cdot sin(\theta_1) + l_2 \cdot sin(\theta_1 + \theta_2) \quad (2)$$

`Inversek2j` (inverse kinematics) calculates the robot joint angles, based on the desired position of the end of the arm in space. The equations 3 and 4 corresponding to the calculation of the angles.

$$\theta_2 = arccos(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 \cdot l_1 \cdot l_2}) \quad (3)$$

$$\theta_1 = arcsin(\frac{y \cdot (l_1 + l_2 \cdot cos(\theta_2)) - x \cdot l_2 \cdot sin(\theta_2)}{x^2 + y^2}) \quad (4)$$

In this work, the input for both programs was a 64-bit floating-point matrix of 500×2 spatial coordinates.

Adhering to the AC method of *simplified iterations*, we replaced complex calculations with simpler ones during specific iterations. The extent of approximation varies depending on the functional analysis of each algorithm. Both algorithms depend on trigonometric functions in their kinematic equations, and their computations entail considerable computational resources. Therefore, replacing them with simpler operations results in a significant reduction in execution time. More specifically, we approximated the functions $sine$, $cosine$, $arcsine$, and $arccosine$ by replacing them with straight line segments that are close to the values of the original trigonometric functions.

In the case of the `Forwardk2j` algorithm, the approximations of these trigonometric functions ($sine$ and $cosine$) used in the kinematic equations through line segments are illustrated in Figure 4.

The kinematic equations for `Inversek2j` are based on the inverse trigonometric functions $arcsine$, and $arccosine$. The approximations of these functions are shown in Figure 5.

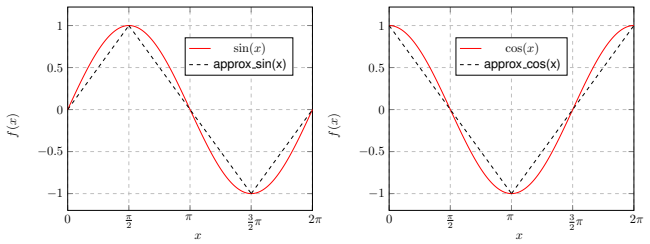

Fig. 4. Approximation of trigonometric functions

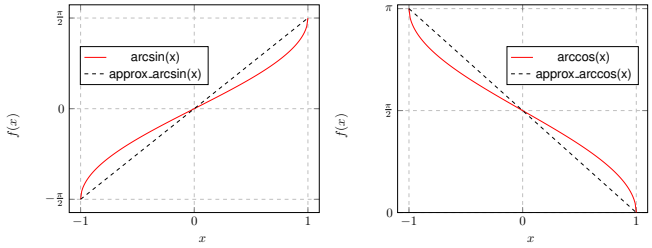

Fig. 5. Approximation of inverse trigonometric functions

For the radiation tests in this work, we selected two approximated versions (A2 and A5), in addition to the precise one (P). In version A2, the precise operations were performed in every third iteration, whereas in version A5, one out of every six iterations utilized original calculations, with the remaining five iterations utilizing approximate operations. With this AC method, a speed-up of $2.81\times$ and $5.06\times$ was obtained for the A2 and A5 versions in `Forwardk2j`. In the case of `Inversek2j`, the speed-up obtained were $2.84\times$ and $5.22\times$ for A2 and A5, respectively.

The inaccuracy of the results due to the approximation was evaluated using the Symmetric Mean Absolute Percentage Error (SMAPE) metric [24], which involves dividing the difference between the exact and approximate results by their sum, as shown in equation 5.

$$\epsilon = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{|P_i - A_i|}{P_i + A_i} \cdot 100 \tag{5}$$

SMAPE solves two problems of the classical Mean Absolute Percentage Error (MAPE) metric: the first is the lack of symmetry, the result is different when exchanging precise and approximate values in the classical equation; the second is that the distortion is maximized with precise results close to zero in MAPE. Therefore, in this work, we have preferred the SMAPE to measure the imprecision of the results.

By employing *simplified iterations*, significantly fewer inaccuracies are achieved in the result compared to those obtained using the traditional *loop perforation* technique.

If *loop perforation* were employed, inaccuracies of 67% and 83.3% would be attained for versions A2 and A5 of the two algorithms, respectively. Conversely, when using *simplified iterations*, results in deviations of 5.8% and 8.1% are obtained for the approximate versions of `Forwardk2j` and 18.6% and 22.2% for `Inversek2j`.

The performance improvement of the approximate versions is reflected in their reduction of overhead in execution time due to the program hardening. In the worst case scenario where all three executions of the `Forwardk2j` algorithm are performed in DWCF, the overhead for the precise and hardened version is $3.02\times$. However, the use of the A2 version reduces the overhead to $1.09\times$, while the A5 version further reduces it to $0.60\times$. Similar results are obtained for the `Inversek2j` algorithm, with an overhead of $3.01\times$ for the precise version and $1.07\times$ and $0.59\times$ for the A2 and A5 versions, respectively.

V. Radiation Experiments

The device under test (*DUT*) selected for the irradiation assessment was the Zynq Board, equipped with a 28-nm CMOS Xilinx ZYNQ XC7Z010 system-on-chip (SoC) [25]. This SoC integrates a dual-core 667 MHz 32-bit ARM Cortex A9 microprocessor with a 13-stage instruction pipeline, branch prediction, and support for two levels of cache. It also has 256 KiB of built-in on-chip memory (OCM) and 512 MiB of external DDR memory. In this work, only one processor was used having its L1 data cache disabled. The *DUT* was controlled by an external computer, the *RaspberryPi 3 Model B* acting as a control computer (*CC*), the main task of which was to supervise the status of the *DUT*, receive and log all the information generated in the *DUT* and reboot it when necessary. The *DUT* was configured to send a state message every 5-10s in the absence of errors and the *CC* is programmed to receive them. In case there are no messages incoming for twice the message time, then the *CC* automatically resets and reprograms the *DUT*. In the next picture (figure 6) you can see the aforementioned setup assembled to perform the experiments.

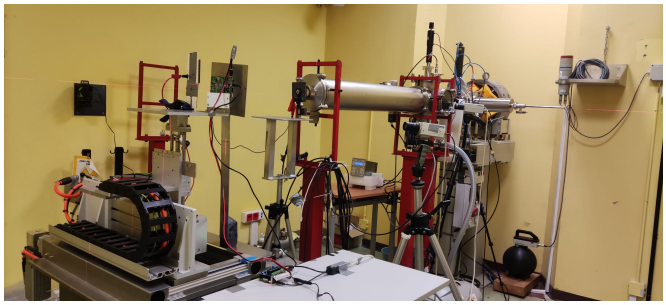

Fig. 6. Experimental set-up established inside the radiation chamber

A. AC+DWCF technique assessment.

An irradiation experiment was conducted using the selected test bench. Both programs are tested using unhardened (`_UH_`) and hardened (`_H_`) versions, and three levels of approximation: full precise or accurate computing (`_P`) and the two aforementioned approximated versions: `_A2` and `_A5`. As an example of the notation used in this paper, `FW_UH_P` means that we have irradiated an unhardened version (UH) of `forwardk2j` (FW) with full precision (P). All compilations were performed using the GCC compiler from the Linaro project (version `arm-eabi-gcc v7.2-2017.11`).

TABLE I
PROTON BEAM TEST RESULTS

Bench	$\Phi \cdot 10^{11}$ (p^+/cm^2)	#events			$\sigma^{upper\ limit}_{lower\ limit} \cdot 10^{-10}\ (cm^2)$					$MWTF \cdot 10^{12}$		
		#SDC	#Det / Det&Rec	#Hang / Invalid Status	SDC	Improvement (SDC)	Det / Det&Rec	Hang + Invalid Status	Total**	SDC	Hang + Invalid Status	Total**
FW_UH_P	2.07	99	–	16 / 1	$4.79\ ^{5.94}_{3.78}$	×1	–	$0.82\ ^{1.32}_{0.47}$	$5.62\ ^{6.87}_{4.49}$	0.745	4.34	0.635
FW_H_P	3.13	0	3 / 75	13 / 4	$*0.03\ ^{0.18}_{0.00}$	**×150**	$0.10\ ^{0.28}_{0.02}$ / $2.40\ ^{3.05}_{1.83}$	$0.54\ ^{0.88}_{0.31}$	$0.54\ ^{0.88}_{0.31}$	66.8	3.93	3.71
FW_UH_A2	3.56	139	–	12 / 4	$3.90\ ^{4.71}_{3.17}$	×1	–	$0.45\ ^{0.73}_{0.25}$	$4.35\ ^{5.21}_{3.56}$	1.20	10.4	1.08
FW_H_A2	3.76	0	6 / 63	18 / 11	$*0.03\ ^{0.15}_{0.00}$	**×146**	$0.16\ ^{0.35}_{0.06}$ / $1.68\ ^{2.18}_{1.25}$	$0.77\ ^{1.12}_{0.51}$	$0.77\ ^{1.12}_{0.51}$	102	3.52	3.40
FW_UH_A5	3.19	105	–	17 / 4	$3.24\ ^{3.99}_{2.56}$	×1	–	$0.65\ ^{1.00}_{0.39}$	$3.88\ ^{4.72}_{3.13}$	1.57	7.86	1.31
FW_H_A5	2.57	0	8 / 52	19 / 5	$*0.04\ ^{0.28}_{0.00}$	**×83.1**	$0.31\ ^{0.62}_{0.13}$ / $2.02\ ^{2.69}_{1.47}$	$0.94\ ^{1.40}_{0.59}$	$0.94\ ^{1.40}_{0.59}$	72.3	3.01	2.89
IN_UH_P	2.61	89	–	13 / 8	$3.41\ ^{4.27}_{2.66}$	×1	–	$0.81\ ^{1.24}_{0.49}$	$4.22\ ^{5.18}_{3.36}$	0.347	1.47	0.281
IN_H_P	1.78	0	7 / 34	17 / 11	$*0.06\ ^{0.31}_{0.00}$	**×60.7**	$0.39\ ^{0.81}_{0.16}$ / $1.91\ ^{2.69}_{1.29}$	$1.57\ ^{2.29}_{1.02}$	$1.57\ ^{2.29}_{1.02}$	11.3	0.403	0.390
IN_UH_A2	2.99	106	–	16 / 3	$3.54\ ^{4.36}_{2.81}$	×1	–	$0.64\ ^{1.00}_{0.37}$	$4.18\ ^{5.08}_{3.36}$	0.695	3.88	0.589
IN_H_A2	2.92	0	7 / 61	18 / 5	$*0.03\ ^{0.19}_{0.00}$	**×103**	$0.24\ ^{0.50}_{0.09}$ / $2.09\ ^{2.72}_{1.55}$	$0.79\ ^{1.19}_{0.49}$	$0.79\ ^{1.19}_{0.49}$	38.9	1.69	1.62
IN_UH_A5	2.68	81	–	21 / 3	$3.02\ ^{3.82}_{2.33}$	×1	–	$0.90\ ^{1.34}_{0.56}$	$3.92\ ^{4.83}_{3.11}$	1.09	3.69	0.843
IN_H_A5	2.88	0	5 / 52	22 / 9	$*0.04\ ^{0.22}_{0.00}$	**×77.68**	$0.17\ ^{0.41}_{0.06}$ / $1.81\ ^{2.40}_{1.31}$	$1.21\ ^{1.73}_{0.80}$	$1.21\ ^{1.73}_{0.80}$	72.3	2.33	2.26

* No errors observed, so for comparison purposes, this is calculated given one error (assuming the worst-case)

** Total is measured as SDC+Hang+Invalid Status

The test campaign was carried out on the external beamline of the compact 18/9 ion beam applications cyclotron located at Centro Nacional de Aceleradores (CNA) [26] in Sevilla, Spain, in March 2023. The *DUT* was irradiated without thinning in the open air. The beam energy at the *DUT* surface was 15.3 MeV, with an estimated spread in the order of 400 keV. The average proton flux was maintained in the range of $5.2 - 6.5 \cdot 10^8\ p^+/(cm^2 \cdot s)$ and the total fluence for each run was within $2.1 - 3.8 \cdot 10^{11}\ p^+/cm^2$. The homogeneous beam spot of $16mm$ in diameter covered the area of interest. The energy of the protons in the active area of the silicon is considered enough to produce single-event effects on the 28-nm technology device with no thinning.

We have observed the following error categories:

- Silent Data Corruption (SDC). The benchmark execution has finished with errors in the resulting output data. An error is considered when only one bit differs from the golden result.
- Hangs: The processor execution flow has been abruptly interrupted by an unexpected exception, probably caused by forbidden memory access. If not handled, this type of error would become a timeout error. The *DUT* becomes unresponsive.
- Communication error: The serial communication with the processor has become corrupted, making it impossible to identify further errors. The logs become unreadable.
- Communication stuck-at incongruous: The serial port enters an infinite loop and sends the same message forever.

Table I represents the results of the irradiation experiment. We show, for each benchmark, the total accumulated fluence (Φ) in the second column, next are all the observed events, beginning with the SDC events in the third column; the next has the "Detected / Detected&Recovered" (#Det/Det&Rec) that were measured using the hardened versions, and the fifth has system hangs and invalid status events (communication error, communication stacks, etc.). The sixth column begins with their corresponding cross-sections (σ) up to the ninth column, where the cumulative cross-section for each benchmark can be read. All the cross-sections are presented the same way: just to the right of the measured value, two numbers are provided; the top value is the upper limit of the confidence interval, while the bottom one is the lower limit of the confidence interval. These values were calculated using the inverse chi-squared ($inv - \chi^2$) distribution as described in [27], at a confidence level of 95% considering the fluence uncertainty of $\pm 10\%$. Versions (`P`, `A2` and `A5`) of each benchmark are arranged in separate rows in the table to better show the evidence of the improvement of the hardening technique. Finally, to make it possible to compare the different versions of the benchmarks taking into account not only the error rate but also the inherent time overhead, the well-known metric *MWTF* (Mean Work to Failure) was used.

Regarding the number of observed *Hang/Invalid* events, those related to the hardened versions always suppose an increment with their un-hardened counterparts with the only exception of the Hang number from the first row (`FW_UH_P` vs `FW_H_P`). This behavior was also expected because the hardened code is always instrumented to implement both the protection (DWCF) and the corresponding AC technique. This implies a more failure-prone code because we have more code to be executed, with newly added critical points that have to

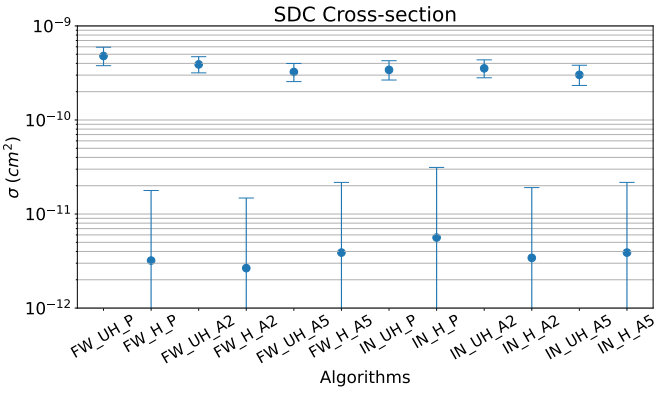

Fig. 7. Representation of SDC Cross-section for all tested benchmarks. Note that the missing lower values extends down to zero

TABLE II
AC TECHNIQUE WITH ACCURACY AWARE LAYER EXPERIMENTAL RESULTS

Bench	$\Phi \cdot 10^{11}(p^+/cm^2)$	#events			$\sigma \cdot 10^{-10}(cm^2)$				$MWTF \cdot 10^{12}$		
		#SDC	#Crit / Tol	#Hang / Invalid Status	SDC	Crit / Tol	Hang + Invalid Status	Total**	SDC	Hang + Invalid Status	Total**
FW_UH_A2-T	3.94	128	47 / 81	15 / 2	$3.25\ ^{3.94}_{2.62}$	$1.19\ ^{1.60}_{0.86}$ / $2.06\ ^{2.60}_{1.59}$	$0.43\ ^{0.69}_{0.25}$	$3.68\ ^{4.43}_{3.00}$	2.77	20.9	2.45
FW_UH_A5-T	3.90	158	60 / 98	12 / 4	$4.05\ ^{4.84}_{3.32}$	$1.54\ ^{2.01}_{1.14}$ / $2.51\ ^{3.12}_{1.98}$	$0.41\ ^{0.67}_{0.23}$	$4.46\ ^{5.30}_{3.68}$	3.16	31.2	2.87

** Total is measured as SDC+Hang+Invalid Status

do with the control flow of the program.

It is worth mentioning that no SDC events were observed in the hardened versions of the benchmarks. With this scenario, the corresponding cross-sections were calculated assuming a worst-case of 1 SDC event per experiment [28], thus avoiding a null cross-section.

As expected, the protected versions of each benchmark show an improvement in the SDC cross-section. This improvement is measured to be of at least one order of magnitude in each case. For example on the first 4.79 vs 0.03 ($10^{-10}cm^2$) (see Table I and Figure 7) in `FW_UH_P` and `FW_H_P`, respectively. The improvements range from $\times 60$ (`IN_UH_P` vs `IN_H_P`) up to $\times 150$ (`FW_UH_P` vs `FW_H_P`). Focusing on the AC techniques (A2 and A5), the SDC cross-sections increase also as expected, this is, we observe larger SDC cross-sections in the A2 versions than in the A5 ones. The reason behind is because A5 versions suppose less computing than the A2 ones.

Figure 8 represents the diverse MWTF SDC and Hang scores for each benchmark. X-axis coordinates have been ordered so that each corresponding score improves all previous ones. By doing that we obtain a series of monotonically increasing scores that give relevant information (ranking) about the most convenient benchmark to be used. For example, focusing on the MWTF SDC ordering, we observe that, as expected, a first group of non-hardened versions (`UH_P` $<$ `UH_A2` $<$ `UH_A5`) show a similar score which is then improved, by 2 orders of magnitude, by the remaining 3 MWTF for hardened versions, ordered as `H_P` $<$ `H_A5` $<$ `H_A2` for `FW` benchmark and `H_P` $<$ `H_A2` $<$ `H_A5` for `IN` one. It is also remarkable that precise versions (`H_P`) elicit worse scores in each group (including Hang scores). These arrangements meet the progressive increment in the complexity and duration of each version.

Gathering the relevant information from Table I and Figure 8, we see that the single application of the AC techniques implies an improvement in protection as a side effect. In fact, the MWTF and SDC cross-sections become larger in this case. This improvement is more evident in the SDC cross-section. If the hybrid AC + DWCF is applied, we observe an improvement in both metrics. Logically, not all algorithms allow us to vary the accuracy using approximate computing, so these techniques are limited to the cases where it is possible.

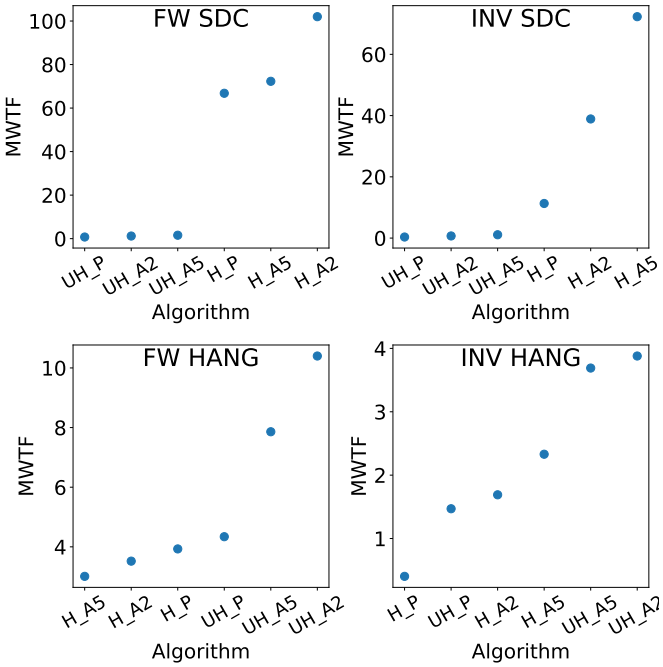

Fig. 8. Monotonically increasing representation of SDC and Hang MWTF (10^{12}) scores for each version of the algorithm.

B. Accuracy-aware layer for AC technique

An additional accuracy-aware layer has been developed for the following experimental phase to introduce an enhanced differentiation among errors, categorizing them as either "tolerable" or "critical." This layer uses a tolerance parameter to evaluate the degree of deviation in the results, thereby distinguishing errors that require recalculation from those that can be considered acceptable. The results gathered in this experiment are presented in Table II.

In the previous case, an error is considered when just one bit in the experimental result a' differs from the golden result a. However, when using AC there is an intrinsic inaccuracy regarding the full precision values that is considered tolerable. In this approach, erroneous results that fall inside this tolerable accuracy interval will not need any intervention at all. To take this effect into account we examine the errors to determine if, even being affected by a fault, they keep the necessary accuracy.

The layer fundamentally alters the error detection methodology, replacing the usual precise comparison ($a - a' = 0$) with an absolute value subtraction, where the outcome must fall below a defined tolerance threshold ($T > 0 : |a - a'| < T$). For this experiment, T has been determined as twice the standard deviation of the differences between the precise result and the approximated one ($T = stdev_{|precise-approx|} \cdot 2$),

ensuring it encompasses at least 95% of the values. This subtle classification of errors into tolerable and critical allows for reducing the number of error corrections, thereby enhancing efficiency. As long as the condition $(a - T) < a' < (a + T)$ is met, an error is considered tolerable, obviating the need for a third computation of the output. By conducting this validation on the output, it is ensured that errors do not propagate, eliminating the risk of causing undesired situations or potentially impacting performance.

In the case the range of the inputs is known, we can precalculate the inaccuracy of the approximate computing and then obtain the threshold. However, in other cases, we will need to use a statistical or worst-case approach to derive the tolerance threshold. Another mechanism of defining T value could be based on the real specifications of the task or the capabilities of the actual robot for this matter.

The accuracy-aware layer is functional for both hardened and unhardened algorithmic versions. In this experiment, the layer was tested on implementations denoted as `FW_UH_A2-T` and `FW_UH_A5-T`, where the suffix '`-T`' indicates the incorporation of the accuracy-aware layer.

The results are presented in Table II. Notably, the number of SDC events is comparable to the previous findings in Table I. Remarkably, only approximately 37% of SDC events in both cases are deemed as "critical" while the remainder are considered tolerable. This observation presents an opportunity to substantially reduce computational and temporal costs associated with software redundancy by selectively executing the third iteration in DWCF only for SDCs lying outside the defined tolerance interval.

Note that in this experiment the T value was defined and constant, nevertheless, this parameter can be configurable to be variable and adjustable to different levels of confidence to match the task needs

VI. Conclusions

Approximate computing is used to shrink the time overheads to a minimum while the accuracy of the approximated program output remains valid and under control. In this paper, we have studied and assessed a variation of an approximate computing-based mitigation technique (using DWCF instead of traditional TMR) for radiation-induced soft errors in COTS microprocessors. Several unprotected and protected versions at different levels of approximation were studied and subjected to a radiation campaign under proton.

Radiation experimental results show that the technique can detect and correct single-event effects keeping the accuracy under control and without compromising performance. The main result of this work is that the computation can be significantly protected by applying AC with or without a hardening technique. This is, AC improves the protection as a side effect, as the two most relevant metrics (SDC cross-section and MWFT) reveal.

Regarding the last experiment, with the inclusion of the accuracy-aware layer, a considerable reduction in errors that require correction can be observed. Also, it should be taken into account that reducing the size of the accuracy interval, will result in more values falling outside the interval, which will increase the number of critical events. T should be adjusted to provide the best trade-off between the number of tolerable errors and the necessary accuracy.

References

[1] G. S. Rodrigues, A. Barros de Oliveira, F. L. Kastensmidt, V. Pouget, and A. Bosio, "Assessing the reliability of successive approximate computing algorithms under fault injection," *Journal of Electronic Testing*, vol. 35, no. 3, pp. 367–381, Jun. 2019.

[2] A. Aponte-Moreno, A. Moncada, F. Restrepo-Calle, and C. Pedraza, "A review of approximate computing techniques towards fault mitigation in HW/SW systems," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*. IEEE, Mar. 2018, pp. 48–53.

[3] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[4] A. Aponte-Moreno, F. Restrepo-Calle, and C. Pedraza, "A low-cost fault tolerance method for arm and risc-v microprocessor-based systems using temporal redundancy and approximate computing through simplified iterations," *Journal of Integrated Circuits and Systems*, vol. 16, no. 3, p. 1–14, Apr. 2022.

[5] I. A. C. Gomes and F. G. L. Kastensmidt, "Reducing TMR overhead by combining approximate circuit, transistor topology and input permutation approaches," in *2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Sep. 2013, pp. 88–93.

[6] T. Arifeen, A. S. Hassan, H. Moradian, and J. A. Lee, "Probing Approximate TMR in Error Resilient Applications for Better Design Tradeoffs," in *Proceedings - 19th Euromicro Conference on Digital System Design, DSD 2016*. IEEE, Aug. 2016, pp. 637–640.

[7] A. J. Sanchez-Clemente, L. Entrena, and M. Garcia-Valderas, "Partial TMR in FPGAs Using Approximate Logic Circuits," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2233–2240, Aug. 2016.

[8] A. Sánchez, L. Entrena, and F. Kastensmidt, "Approximate TMR for selective error mitigation in FPGAs based on testability analysis," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, Aug. 2018, pp. 112–119.

[9] B. Deveautour, M. Traiola, A. Virazel, and P. Girard, "Qamr: An approximation-based fully reliable tmr alternative for area overhead reduction," in *2020 IEEE European Test Symposium (ETS)*, vol. 2020-May. Institute of Electrical and Electronics Engineers Inc., May 2020, pp. 78–83.

[10] B. Deveautour, M. Traiola, A. Virazel, and P. Girard., "Reducing Overprovision of Triple Modular Reduncancy Owing to Approximate Computing," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, Jun. 2021, pp. 142–148.

[11] M. Traiola, J. Echavarria, A. Bosio, J. Teich, and I. O'Connor, "Design Space Exploration of Approximation-Based Quadruple Modular Redundancy Circuits," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, vol. 2021-November. IEEE, Nov. 2021, pp. 759–767.

[12] H. Cho, L. Leem, and S. Mitra, "ERSA: Error Resilient System Architecture for Probabilistic Applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 546–558, Apr. 2012.

[13] F. Baharvand and S. G. Miremadi, "Lexact: Low energy n-modular redundancy using approximate computing for real-time multicore processors," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 431–441, Apr. 2020.

[14] G. S. Rodrigues, A. Barros de Oliveira, A. Bosio, F. L. Kastensmidt, and E. Pignaton de Freitas, "ARFT: An Approximative Redundant Technique for Fault Tolerance," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, Nov. 2018, pp. 108–113.

[15] F. F. dos Santos, M. Brandalero, M. B. Sullivan, P. M. Basso, M. Hubner, L. Carro, and P. Rech, "Reduced Precision DWC: An Efficient Hardening Strategy for Mixed-Precision Architectures," *IEEE Transactions on Computers*, vol. 71, no. 3, pp. 573–586, Mar. 2022.

[16] A. Aponte-Moreno, F. Restrepo-Calle, and C. A. Pedraza, "FTxAC: Leveraging the Approximate Computing Paradigm in the Design of Fault-Tolerant Embedded Systems to Reduce Overheads," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 797–810, Apr. 2021.

[17] L. M. Luza, D. Söderström, G. Tsiligiannis, H. Puchner, C. Cazzaniga, E. Sanchez, A. Bosio, and L. Dilillo, "Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, Oct. 2020, pp. 49–54.

[18] L. M. Luza, A. Ruospo, D. Söderström, C. Cazzaniga, M. Kastriotou, E. Sanchez, A. Bosio, and L. Dilillo, "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1867–1882, Oct. 2022.

[19] G. S. Rodrigues, F. L. Kastensmidt, V. Pouget, and A. Bosio, "Approximate tmr based on successive approximation to protect against multiple bit upset in microprocessors," in *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, vol. 1, Sep. 2018, pp. 129–133.

[20] M. M. Goncalves, I. P. Lamb, P. Rech, R. M. Brum, and J. R. Azambuja, "Improving selective fault tolerance in gpu register files by relaxing application accuracy," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1573–1580, Jul. 2020.

[21] H. M. Quinn, Z. K. Baker, T. D. Fairbanks, J. L. Tripp, and M. G. Duran Il, "Robust duplication with comparison methods in microcontrollers," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, p. 338–345, Jan. 2016.

[22] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, Apr. 2017.

[23] Z. Wen, Y. Wang, J. Luo, A. Kuijper, N. Di, and M. Jin, "Robust, fast and accurate vision-based localization of a cooperative target used for space robotic arm," *Acta Astronautica*, vol. 136, pp. 101–114, Jul. 2017.

[24] C. Tofallis, "A better measure of relative prediction accuracy for model selection and model estimation," *Journal of the Operational Research Society*, vol. 66, no. 8, pp. 1352–1362, aug 2015.

[25] Xilix, UG585, "Zynq-7000 all programmable SoC: Technical reference manual," 2016.

[26] Centro Nacional de Aceleradores, "CNA," http://www.cna.us.es, Last visited: March 31st, 2023, Seville, Spain.

[27] ESA/ESCC, "Single event effects test method and guidelines. escc basic specification no. 25100," ESA/ESCC European Space components Coordination, Oct 2014.

[28] H. Quinn, "Challenges in testing complex systems," *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, Apr. 2014.