

# On-Board Multi-Core Fault-Tolerant SAR Imaging Architecture

Helena Cruz\*, Rui Policarpo Duarte†, Horácio Neto‡

Instituto Superior Técnico, Lisbon, Portugal

Email: \*helena.cruz@tecnico.ulisboa.pt, †rui.duarte@tecnico.ulisboa.pt, ‡hcn@inesc-id.pt

**Abstract**—Nowadays, there is an increasing need for satellites, drones and UAVs to have lightweight, small, autonomous, portable, battery-powered systems able to generate SAR images on-board and broadcasting them to Earth, avoiding the time-consuming data processing at the receivers. SAR is a form of radar used to generate images of Earth, mounted on moving platforms, such as satellites, drones or airplanes and is used to monitor the surface of the Earth. Backprojection is an algorithm for SAR image generation capable of generating high quality images, however, it is one of the most computationally intensive algorithms. Space is a harsh environment, due to the radiation, which causes temporary or permanent errors on computing systems, thus, there is a need to mitigate its impact on the devices by implementing fault tolerance mechanisms to detect and correct errors. In this research work, an on-board multi-core embedded architecture was developed for SAR imaging systems, implementing two fault tolerance mechanisms: lockstep and reduced-precision redundancy. This architecture aims to protect the Backprojection algorithm from transient faults in the processing unit, using a software-only approach, generating acceptable SAR images in a space environment. The solution was implemented on a Xilinx SoC device with a dualcore processor. For error rates similar to the ones measured in a space environment, the present work produced images with less 0.65dB on average at the expense of a time overhead up to 33%. Notwithstanding, the Backprojection algorithm executed up to 1.58 times faster than its single-core version algorithm, without fault tolerance mechanisms.

## I. INTRODUCTION

Space is a harsh environment for electronic components used in circuits and systems. Therefore, systems designed for spacecrafts or satellites must be highly reliable and be able to tolerate the levels of radiation present in space. The main sources of radiation in space are high-energy cosmic ray protons and heavy ions, protons and heavy ions from solar flares, heavy ions trapped in the magnetosphere and protons and electrons trapped in the Van Allen belts [1]. These are capable of deteriorating the electronic systems and provoking bit-flips, leading to failures in electronic systems [2]. Fault tolerance mechanisms are used to increase the reliability of these systems.

Synthetic-Aperture Radar (SAR) is a form of radar which is usually mounted on moving platforms such as satellites, aircrafts and drones and is used to generate 2D and 3D images of Earth. SAR operates through clouds, smoke and rain and does not require a light source, making it a very attractive method to monitor the Earth, in particular, the melting of polar ice-caps, sea level rise, wind patterns, erosion, drought prediction, precipitation, landslide areas, oilspills, deforestation, fires, natural disasters such as hurricanes, volcano eruptions and earthquakes. There is a need for satellites, drones and

Unmanned Aerial Vehicles (UAVs) to have a lightweight, small, autonomous, portable, battery-powered system able to generate SAR images on-board and broadcasting them to Earth, avoiding processing data on the receivers. This paper describes an architecture for SAR imaging capable of tolerating faults resulting from radiation in a space environment. This architecture uses the Backprojection Algorithm to generate images and is implemented and tested on a System-on-Chip (SoC) device [3].

## II. BACKGROUND

This section introduces SAR, the Backprojection algorithm and fault tolerance mechanisms.

### A. Synthetic-Aperture Radar (SAR)

SAR is a form of radar used to generate 2D and 3D high resolution images of objects. Unlike other radars, SAR uses the relative motion between the radar and the target to obtain its high resolution. This motion is achieved by mounting the radar on moving platforms such as satellites, aircrafts or drones. The distance between the radar and the target in the time between the transmission and reception of pulses creates the synthetic antenna aperture. The larger the aperture, the higher the resolution of the image, regardless of the type of aperture used. To generate SAR images, it is necessary to use an image generation algorithm, such as the Backprojection Algorithm, described below.

### B. Backprojection Algorithm

Backprojection algorithm takes the following values as input: number of pulses, location of the platform for each pulse, the carrier wavenumber, the radial distance between the plane and target, the range bin resolution, the real distance between two pixels and the measured heights. Then, for each pixel and each pulse, the Backprojection algorithm performs the following steps [4]:

- 1) Computes the distance from the platform to the pixel.
- 2) Converts the distance to an associated position (range) in the data set (received echoes).
- 3) Samples at the computed range using linear interpolation, using Eq. 1 [5].

$$g_{x,y}(r_k) = g(n) + \frac{g(n+1) - g(n)}{r(n+1) - r(n)} \cdot (r_k - r(n)) \quad (1)$$

- $g(n)$  - Wave sample in the previous adjacent range bin.
- $g(n+1)$  - Wave sample in the following adjacent range bin.

- $r(n)$  - Corresponding range to the previous adjacent bin.
  - $r(n+1)$  - Corresponding range to the following adjacent bin.
  - $r_k$  - Range from pixel  $f(x, y)$  to aperture point  $\theta_k$ .
- 4) Scales the sampled value by a matched filter to form the pixel contribution. This value is calculated using Eq. 2 [5].  $dr$  is calculated using Eq. 3 [5].

$$e^{i\omega 2|\vec{r}_k|} = \cos(2 \cdot \omega \cdot dr) + i \sin(2 \cdot \omega \cdot dr) \quad (2)$$

$$dr = \sqrt{(x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2} - r_c \quad (3)$$

- $dr$  - Differential range from platform to each pixel versus center of swath.
  - $x_k, y_k, z_k$  - Radar platform location in Cartesian coordinates.
  - $x, y, z$  - Pixel location in Cartesian coordinates.
  - $r_c$  - Range to center of the swath from radar platform.
- 5) Accumulates the contribution into the pixel. The final value of each pixel is given by Eq. 4 [5].

$$f(x, y) = \sum_k g_{x,y}(r_k, \theta_k) \cdot e^{i \cdot \omega \cdot 2 \cdot |\vec{r}_k|} \quad (4)$$

- $f(x, y)$  - Value of each pixel  $(x, y)$ .
- $\theta_k$  - Aperture point.
- $\omega$  - Minimal angular velocity of wave.
- $g_{x,y}(r_k, \theta_k)$  - Wave reflection received at  $r_k$  at  $\theta_k$  (calculated using the linear interpolation in Eq. 1).

---

**Algorithm 1** Backprojection algorithm pseudocode.

**Source:** PERFECT Manual Suite [4].

---

```

1: for all pixels  $k$  do
2:    $f_k \leftarrow 0$ 
3:   for all pulses  $p$  do
4:      $R \leftarrow ||a_k - v_p||$ 
5:      $b \leftarrow \lfloor (R - R0) / \Delta R \rfloor$ 
6:     if  $b \in [0, N_k p - 2]$  then
7:        $w \leftarrow \lfloor (R - R0) / \Delta R \rfloor - b$ 
8:        $s \leftarrow (1 - w) \cdot g(p, b) + w \cdot g(p, b + 1)$ 
9:        $f_k \leftarrow f_k + e^{i \cdot k_u \cdot R} \cdot s$ 
10:    end if
11:  end for
12: end for

```

---

Algorithm 1 shows the pseudocode to compute the aforementioned steps. In the pseudocode,  $k_u$  represents the wavenumber and is given by  $\frac{2\pi f_c}{c}$ , where  $f_c$  is the carrier frequency of the waveform and  $c$  is the speed of light,  $a_k$  refers to the position of the pixel, and  $v_p$ , corresponds to the platform position. The complex exponential  $e^{i\omega}$  is equivalent to  $\cos(\omega) + i \sin(\omega)$  and, therefore, a cosine and sine computation is implied in the calculation of each pixel, represented in Eq. 4.

### C. Quality Assessment

The quality of a SAR image can be evaluated using Signal-To-Noise Ratio (SNR) [4]. SNR is used to obtain the relation between the desired signal and the background noise and is

expressed in decibels. SNR is calculated using Eq. 5. The larger the SNR value, the greater the agreement between the values. According to [4], values above 100dB are reasonable.

$$SNR_{dB} = 10 \log_{10} \left( \frac{\sum_{k=1}^N |r_k|^2}{\sum_{k=1}^N |r_k - t_k|^2} \right) \quad (5)$$

- $r_k$  - Reference value for  $k$ -th element.
- $t_k$  - Test value for  $k$ -th element.
- $N$  - Number of entries to compare.

### D. Fault Tolerance

Fault Tolerance (FT) is the ability of a system to be able to remain functional even in the presence of failures. Fault-tolerant systems are able to detect faults and to recover from them.

Triple Modular Redundancy (TMR) consists in having three entities calculating the same value and have a voter entity compare the results. The most common value is assumed to be correct [6], [7].

Lockstep consists in the concurrent execution of the same application on the different cores of a processor. The state of each core is periodically checked to ensure the execution is running without errors. If the states match, the execution is assumed to be correct and a checkpoint is taken to be used in case of a future fault. Otherwise, a previous state, resulting from a checkpoint, is restored. Checkpoints contain the values of the processor's registers or variables to be used in future comparisons. Oliveira et. al [8] presented a similar lockstep implementation. Process-Level Redundancy (PLR) [9] is a software-only fault tolerance mechanism that supports TMR and lockstep with checkpoints.

Reduced-Precision Redundancy (RPR) is used to reduce the overhead introduced by TMR by using a Full-Precision (FP) computation and two Reduced-Precision (RP) computations. RPR can be implemented in hardware, following an architecture similar to TMR, or software, following an architecture similar to temporal redundancy (i.e., fault tolerance mechanism where an operation is executed several times and the correct result is determined by majority voting). The FP computation corresponds to the "original computation" and the other two compute an approximation. Computing the approximations is more efficient than calculating a full-precision value, decreasing the overhead of the redundant computations. Examples of applications that use RPR are [10], [11].

There are several fault-tolerant versions of SAR image generation algorithms [7], [12], [6]. Jacobs et. al [7] propose a fault tolerance mechanism for Fast-Fourier Transformer (FFT) Algorithm based on range and azimuth compression by implementing Concurrent Error Detection (CED) and using weighted sum, and also implements scrubbing. Wang et. al [12] also present a mechanism for FFT algorithm based on a weighted checksum encoding scheme. Fang et. al [6] describe a Fault-Management Unit which is responsible for the following functions: a scrub controller to periodically reload the FPGA configurations data, a fault detection circuit to periodically test the hardware, a switching circuit responsible for removing a faulty processor and replace it by an alternative processor, and a majority voter circuit, which is responsible for comparing the

results of a TMR mechanism used during the SAR algorithm execution. The work presented in this section is directed to protect frequency-domain algorithms [7], [12], such as the FFT, or SAR systems in general [6]. This work aims to protect the Backprojection algorithm from faults in the processing unit caused by radiation, protecting one of the best SAR image generation algorithms considering the image quality.

### III. MULTI-CORE FAULT-TOLERANT ARCHITECTURE

The details of the developed architecture are described in the next sections, including the lockstep and RPR mechanisms.

#### A. Algorithm Profiling

The Backprojection algorithm implementation used in this study is part of the PERFECT Suite [4] and is written in C. The pseudocode was presented in Algorithm 1 and is based on the equations presented in the previous section. This suite also contains three input image sets: small, medium and large, which generate an image of sizes 512x512, 1024x1024 and 2048x2048 pixels, respectively. For profiling, the chosen input set was the small one, which took less than 8 minutes. These times were obtained using the optimization level  $\text{O3}^1$ . To profile this algorithm, **gprof**<sup>2</sup> was used.

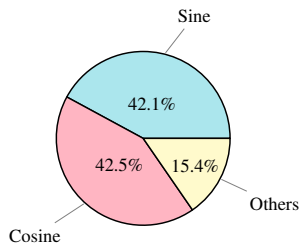


Fig. 1. Backprojection algorithm profiling.

The trigonometric functions are responsible for over 84% of the execution time of the algorithm, which means that the potential for the reduced-precision redundancy mechanism lies in these functions. The rest of the algorithm, including input and output operations, is executed in under 16% of the time.

#### B. Proposed Architecture

The Backprojection algorithm, as analysed in the previous section, can be divided into three blocks: a first and last block which represent the least intensive sections of the algorithm and a middle block where the image generation is performed. The middle block, referred to as image generation, corresponds to the intensive sections. To reduce the overhead introduced by the fault tolerance mechanism, when compared to temporal redundancy or lockstep only, a mixed approach was developed:

- On the sections with less computations, a Lockstep mechanism was used, as seen in [8], [13], [9], [14]. This mechanism ensures the initialization is done properly and the raw SAR data is correct before beginning the image generation.
- On the sections with more computations, a RPR mechanism was developed. RPR is a special case of temporal

redundancy where, instead of using full-precision computations for the redundant values, reduced-precision computations are used. RPR is used in several applications [10] where a small fraction of precision is sacrificed for performance. This allows the generation of an image with an acceptable quality in the presence of errors without compromising the overall performance. This mechanism allows a reduction in the total overhead when comparing to other mechanisms, such as lockstep.

The calculation of each pixel, or Backprojection Unit (BPU), can be done in parallel, which means each core can calculate one pixel at a time without being necessary for error detection, contrary to lockstep. The middle block, for this reason, is protected by RPR, reducing the total overhead in the system. A scheme of the architecture of the developed fault tolerance mechanism is displayed in Figure 2, where it is possible to observe each of the phases of the Backprojection algorithm and which mechanism is responsible for its protection. In the middle blocks, approximations are calculated after the complete computations. The approximations are represented in Figure 2 as Reduced-Precision Backprojection Units (rBPUs).

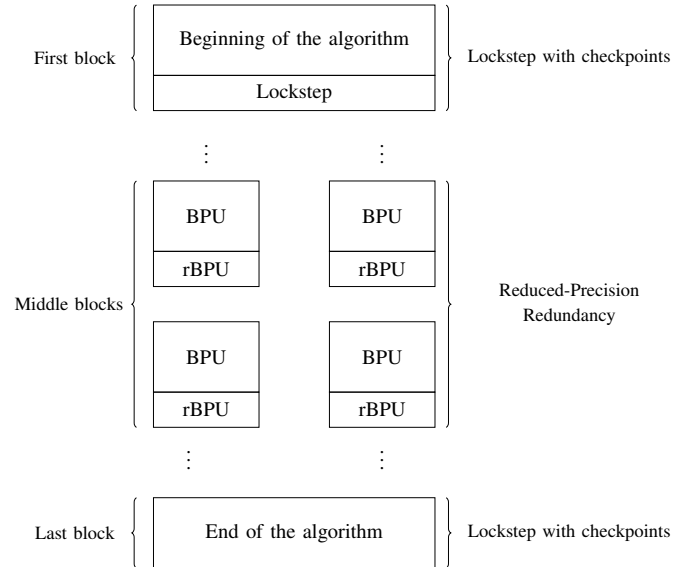


Fig. 2. Architecture of the developed fault tolerance mechanism.

1) *Algorithm Parallelization:* In this algorithm, the pixel calculations have no dependencies, therefore, they can be computed in parallel. The workload can be divided between the cores statically or dynamically. A static load-balancing mechanism was tested, since dynamic load-balancing introduces overhead in systems. The results of this test are presented in Table I, where the execution time is presented in function of the number of pixels per batch. The tested number of BPU per batch was 4, 8, 16 and 32.

From Table I it is possible to conclude that the number of BPUs per batch does not have a significant influence on the total execution time. It is also possible to observe that the workload is relatively balanced, since there are not any accentuated differences in the execution times of each core. This leads to conclude that dynamic load-balancing is not

<sup>1</sup><https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

<sup>2</sup>[https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html\\_mono/gprof.html](https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html)

TABLE I

DUAL-CORE EXECUTION TIMES IN FUNCTION OF THE NUMBER OF PIXELS PER BATCH. THE LONGER EXECUTION PER BATCH NUMBER IS DISPLAYED IN BOLD IN THE TABLE.

	Original	Batches of:			
		4	8	16	32
Core 0	—	<b>240.4s</b>	<b>240.6s</b>	<b>241.5s</b>	<b>241.7s</b>
Core 1	—	239.9s	239.3s	241.0s	239.4s
<b>Total</b>	477.4s	480.3s	479.9s	482.6s	480.4s

necessary and that the batch number is also indifferent. The final chosen number of units per batch was 4, since it was the fastest of all executions in total.

2) *Lockstep*: The Lockstep solution proposed consists in a mechanism where two identical SAR applications are executed simultaneously on a dual-core processor. Both cores execute the applications until reaching the verification point, where the application's variables are compared and, if they match, the execution is assumed correct. Afterwards, a checkpoint is created in memory containing these values. The lockstep mechanism is used during the input reading to avoid the generation of images from incorrect data. Afterwards, the image generation starts and the lockstep is no longer active.

3) *Reduced-Precision Redundancy*: In RPR two values are calculated: FP, or BPU, and RP, or rBPU. Both FP and RP values are compared and, if the FP value deviates more than the acceptable threshold from the RP value, it is assumed the value is incorrect and the RP value is used instead. If not, the FP value is assumed correct and is used. The RP value is used when an error is detected because it is calculated in a shorter amount of time and thus it is less likely to have been affected by a fault. The RP values are calculated using optimizations.

a) *Trigonometric Functions Optimization*: The most optimizable section of the algorithm are the trigonometric functions. The optimization functions tested are described below and the results are presented in Table II.

- COordinate Rotation DIgital Computer (CORDIC) algorithm [15];
- Taylor Series;
- Wilhem's Look-Up Table (LUT)<sup>3</sup>;
- `libfixmath`<sup>4</sup>;
- Ganssle optimizations [16].

One of the conclusions that can be drawn from the analysis of Table II is that all optimizations are indeed faster than the original version, which was expected. However, most of these optimizations lead to a large precision loss.

The implementation of the CORDIC algorithm used to test was developed by John Burkardt<sup>5</sup>. CORDIC is the algorithm with the worst performance, as can be observed, with all its tested versions (from 5 to 30 iterations) being slower than any other version of another algorithm. Even though the precision obtained from the 15 iterations and up was relatively good in comparison with the other algorithms, there were better alternatives. This was expected, since CORDIC is an algorithm

<sup>3</sup><https://www.atwillys.de/content/cc/sine-lookup-for-embedded-in-c/>

<sup>4</sup><https://github.com/PetteriAimonen/libfixmath>

<sup>5</sup>[https://people.sc.fsu.edu/~jburkardt/c\\_src/cordic/cordic.html](https://people.sc.fsu.edu/~jburkardt/c_src/cordic/cordic.html)

TABLE II

COMPARISON OF OPTIMIZATION ALGORITHMS.

		Time	SNR
<b>Original</b>		477.4s	138.9dB
<b>CORDIC</b>	5 iterations	214.7s	30.7dB
	10 iterations	238.8s	60.5dB
	15 iterations	262.7s	90.5dB
	20 iterations	286.3s	120.2dB
	25 iterations	311.3s	136.1dB
	30 iterations	335.1s	136.3dB
<b>Taylor Series</b>	3 terms	176.3s	43.6dB
	4 terms	186.0s	71.8dB
	5 terms	192.3s	103.8dB
	6 terms	201.5s	133.6dB
	7 terms	210.4s	135.3dB
<b>Wilhem's Look-Up Table</b>		123.2s	69.1dB
<code>libfixmath</code>	Taylor I	179.3s	54.5dB
	Taylor II	158.8s	33.6dB
	LUT	134.8s	99.2dB
<b>Ganssle</b>	3 coefficients	163.7s	66.3dB
	4 coefficients	167.3s	105.2dB
	5 coefficients	170.7s	118.3dB
	7 coefficients	176.7s	134.8dB
	7 coefficients (doubles)	179.8s	135.3dB

with a great performance when no hardware multiplier is available. This was not the case in the tested environment and therefore CORDIC was not the best alternative.

The results obtained show that all Taylor Series experiments required less computational time for the same or better SNR than CORDIC. Exception for the 25 and 30 CORDIC iterations which provided approximately 1dB more, but at the expense of more than 2 minutes.

Comparing the results of the Ganssle optimizations with the Taylor Series, for the same image the SNR did not differ significantly, less than 18%. It means that sometimes the processing time for one is greater than the other, and vice-versa.

The Wilhem's Look-Up Table method was the fastest of the tested and outperformed CORDIC with 5 iterations, Taylor Series with 3 terms, Taylor I and Taylor II from `libfixmath` and 3-coefficient Ganssle. It is a good alternative in systems with very limited memory since the LUT table occupies 66 bytes only. The LUT variation of `libfixmath` is more precise and the execution time difference is not significant (11 seconds), but requires a larger LUT (200kB). The memory of the Zybo board does not represent an issue, which means the `libfixmath` is a better alternative given the precision achieved.

Besides the LUT variation, `libfixmath` provides two functions based on Taylor Series. These two variations are outperformed by the Ganssle optimizations and even the author's Taylor Series implementation, with worse performance and less precision. `libfixmath` LUT variation is one of the best options for the Backprojection algorithm optimization.

The Ganssle optimizations represent another alternative for the Backprojection algorithm optimization. The variation with only 3 coefficients is outperformed by both LUT methods. Nevertheless, the other variations provide higher precision without a significant increase in the execution time. There are two functions that vary only in the type of variables they use:

single-precision or double-precision. Double-precision is more subject to errors since it requires more bitwise calculations and the gain in precision is not significant to the point of being worth it. The 4-coefficient variation does not provide much more precision when compared to the `libfixmath` LUT function and the execution time increases by more than 30 seconds, making the former a better alternative. The 5-coefficient variation provides more precision with an execution time increase of less than 36 seconds. The 7-coefficient (implemented with single-precision) function provides a precision very similar to the original, with a difference of only 4dB in the SNR, and an increase of more than 42 seconds.

Concluding, the functions that represent a better option for the Backprojection algorithm optimization are the `libfixmath` LUT and the Ganssle variations of 5 and 7 coefficients. These three functions are used in the implementation of the RPR mechanism.

*b) Word-Length Optimization:* In addition to the trigonometric optimization, the impact of the floating-point precision was also tested. Reducing the precision from double-precision to single-precision variables in the Backprojection Algorithm resulted in a 88% quality loss. The original image had a SNR of 138.9dB and the single-precision variation had 15.7dB. Besides reducing the precision of the variables, it would also be an option to use fixed-point notation instead of floating-point notation. This cannot be done with all variables because the algorithm deals with large values, easily leading to overflow errors.

#### IV. IMPLEMENTATION

The research work developed was tested on a Zybo Zynq-7000 board [3] from Digilent. This board contains a Zynq device from Xilinx, an external 512MB DDR3 memory, and I/O peripherals. The Zynq device contains a Programmable Logic (PL) and a Processing System (PS). The PL corresponds to a Xilinx 7-series Field-Programmable Gate Array (FPGA). The PS main components are a dual-core ARM Cortex-A9 processor (with CPU0 and CPU1) and a memory controller. The Zynq device contains an On-Chip Memory (OCM) with 256kB of RAM and 128kB of ROM.

##### A. Lockstep

To prepare for the execution of the applications on each core, it is necessary to divide the memory between the cores, avoiding overlapping memory positions. Table III presents this division, showing which addresses belong to which application. The applications are in the Double Data Rate (DDR) memory and each of them has the same amount of memory, 100MB, which allows 312MB of free memory to be used for input and output data. This information is put in the `lscript.ld` file, generated by the Xilinx SDK when a project is created. In Table III it is possible to observe that both applications have access to the same OCM memory, since it will be shared between the cores. This leads to the need for synchronization.

Since the developed application is bare-metal, therefore without an Operating System (OS), there is no file system available to deal with input files. To be able to copy the input information to the board memory it is necessary to use the Xilinx Software Command-Line Tool (XSCT) [17]. This can

TABLE III  
ADDRESS RANGE FOR EACH APPLICATION.

	Processor 0 Application	Processor 1 Application
<b>DDR Base Address</b>	0x00100000	0x06400000
<b>DDR Size</b>	0x06400000	0x06400000
<b>OCM 1 Base Address</b>	0x00000000	0x00000000
<b>OCM 1 Size</b>	0x00030000 (192KB)	0x00030000
<b>OCM 2 Base Address</b>	0xFFFF0000	0xFFFF0000
<b>OCM 2 Size</b>	0x0000FE00 (63.5KB)	0x0000FE00

be done using the command `mwr`, which allows the copy of data from a file to the board memory. After the input data is copied into memory, the setup is complete and the applications can be executed. The architecture of the developed lockstep mechanism is described in Fig. 3.

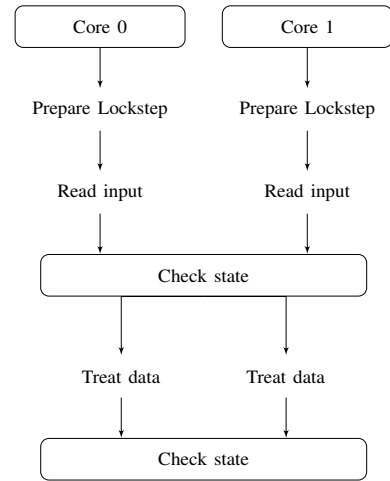


Fig. 3. Lockstep architecture.

The first thing applications do when executing is to prepare for the lockstep. To ensure both processors start at the same time, there is a synchronization point at the beginning of each application.

The main issue resulting from concurrent execution [9] is cache coherence. In order to prevent this, L2 cache, which is shared between cores, is disabled during lockstep. In a concurrent application such as this one it is common to disable both caches. Even when two resources change values in different memory positions it is possible to generate errors - this happens because caches work in blocks of words instead of a single word. In this case, each core has a memory space of 100MB and each application occupies less than 400kB, making it impossible for one core to change a memory position that may belong to a block changed by the other core. Therefore, it is possible to only deactivate L2 cache instead of both cache levels. This also reduces the overhead introduced by the fault tolerance mechanism.

After disabling the L2 cache, it is necessary to prepare the memory for checkpointing. Checkpoints need to be saved in a reserved section of memory which is going to be accessed by both cores. This concurrent access also leads to coherency

issues, therefore, it is mandatory to disable the cache in this section. The memory chosen to store the checkpoints is the OCM, which has a smaller access time when compared to the DDR memory, reducing the overhead introduced by disabling the cache.

The applications on both cores begin by reading the input data from memory. The raw SAR data, read from the input file, contains the following variables:  $f_c$ , corresponding to the carrier frequency,  $r_0$ , corresponding to the range bin,  $dr$ , corresponding to the range bin resolution, an array containing the position of the platform during the data collection and an array of pulses. All of these values except the pulses and platform positions are saved in checkpoints. It is not possible to save the this data because of their size, which can be over 262MB for the large input set, making it impossible to save in memory two copies (one for each application). Instead, it is saved the memory address, ensuring each application is reading their input from the correct address. This work aims to protect against faults in the processing unit, assuming the memory is protected against radiation.

Besides saving the input values, five synchronization variables are also saved in the OCM, used to synchronize both cores. The first step is to ensure each core has created, saved its checkpoint and is ready to compare its state. For this, `p0_saved_flag` and `p1_saved_flag` variables are used. Once each core has saved their state, the value of these flags is switched to 1. Each core waits ten seconds for the other core to save its state, afterwards assumes the other core stopped responding and is not functional and takes its place, executing the rest of the program alone. Besides the timeout, there is also a number of possible tries for each core to try to correct its errors. When comparing the results, if a core detects a discrepancy between its own values and the other core's, the core rereads these values in an attempt to correct the error. The maximum number of tries is 10, afterwards the execution is considered to be incorrect and not recoverable.

After checking the integrity of the input data, the next step, presented in Fig. 3, is the data treatment. This consists in a pre-processing of the input data before starting the algorithm. This pre-processing is followed by another synchronization point and the states of the applications are compared once more. There is also a timeout of ten seconds and a maximum of tries of 10. With the data input concluded, the next step is to execute the Backprojection algorithm. This step, however, is not protected by Lockstep but by RPR.

Before the end of the execution it is necessary to ensure that all data is written in the DDR memory, which can be done by forcing a flush in cache. `Xil_DCacheFlushRange()` is the Xilinx function [18], used to trigger flushes. Afterwards, the image can be written into a file using XSCT. To do this, command `mrd`, is used.

### B. Reduced-Precision Redundancy

The Backprojection algorithm implementation used to test the FT mechanism [4] consists in three nested loops where the two outer ones correspond to the  $x$  and  $y$  coordinates of the pixel and the inner loop corresponds to the number of pulses. The pixel value is given by the sum of pulse contributions, as seen previously in Algorithm 1. RPR is implemented by

adding all the pulse contributions, subject to the effect of faults, and checking for errors against the RP value. Therefore, for each pixel, the final value is equal to the sum of the protected pulse calculations. After calculating both the FP and the RP values, the values are voted on. The FP value is considered correct if deviates less from a certain threshold from the RP value. The threshold equals the maximum error between the approximation and the original value. Concluding, the RPR mechanism avoids the use of more costly mechanisms, such as TMR or lockstep, while taking advantage of the dual-core processor of the Zynq device. This mechanism, similarly to what was mentioned above about the implemented lockstep, targets data errors and is not able to detect nor correct control flow errors.

## V. RESULTS

The results from the evaluation of the mechanism are presented in the next sections.

### A. Dual-Core Evaluation

The three precision reduction optimizations that were implemented and tested are the `libfixmath` LUT and the 5 and 7-coefficient Ganssle trigonometric functions. The execution times of the complete architecture for each of these optimizations is presented in Table IV. As can be observed, the architecture implemented using the `libfixmath` is 1.58 times faster than the serial original version of the algorithm. Regarding the 5-coefficient Ganssle algorithm, the execution was 1.50 times faster than the original and the 7-coefficient Ganssle algorithm was 1.49 times faster than the original version. When compared to the dual-core version of the Backprojection algorithm, the final architecture using the `libfixmath` LUT method, the 5-coefficient and 7-coefficient Ganssle algorithms introduce an overhead of 25.4%, 32% and 33%, respectively.

TABLE IV  
COMPARISON BETWEEN THE EXECUTION TIMES DEPENDING ON THE OPTIMIZATION.

	Optimization				
	Original	Original (dual-core)	<code>libfixmath</code>	5-coefficient Ganssle	7-coefficient Ganssle
Execution Time	477.4s	240.4s	301.5s	317.3s	319.7s

### B. Solution Evaluation

To test the developed architecture, a set of tests was implemented. The fault injection was implemented in software and at compile-time. Regarding the lockstep mechanism in particular, the objective is to observe the impact of the mechanism on the performance of the system, since in the presence of faults the execution is repeated. To test this mechanism, the following tests were implemented:

- **Test 1: Lockstep Deterministic Test** The lockstep-protected section of the applications was affected by a defined number of faults, causing a bit-flip in a pre-defined bit of the same pre-defined variable. Six versions of this test were implemented: a fault occurred 1, 10, 100, 1000, 10000 and 100000 times. The results of this test are presented in Table V.

- **Test 2: Lockstep Dynamic Test** The lockstep- protected section of the applications was affected by a defined number of faults, causing a bit-flip in a random bit in a random variable. All lockstep variables could be affected by a bit-flip at a random moment during the execution. Six versions of this test were implemented: a fault occurred 1, 10, 100, 1000, 10000 and 100000 times. The results of this test are presented in Table VI.

Regarding the RPR mechanism, the objective is to observe the final quality of the generated images, using the SNR, in the presence of faults. To test this mechanism, the following tests were implemented. To inject faults, a fault injection function was called after every statement and a bit-flip could or not affect the last modified variable. The frequency of the bit-flips depends on the test.

- **Test 3: RPR Normal Distribution Test** According to [19], the average occurrences of bit-flips in space is 1 per day. To evaluate mechanism on a scenario with worse conditions, this fault injection follows a normal distribution with a mean value of 40 and a standard deviation of 5. The results of this test are presented in Table VII.
- **Test 4: RPR 1440 bit-flips per day Test** Considering the average of bit-flips according to [19], a worse-case scenario was tested: an average of 1440 bit-flips per day, or one every 60 seconds. The bit-flip affects a random bit in a random variable. The results of this test are presented in Table VIII.
- **Test 5: RPR 2880 bit-flips per day Test** Similarly to the test above, was also tested a scenario where the average of bit-flips per day is 2880, or one every 30 seconds. The bit-flip affects a random bit in a random variable. The results of this test are presented in Table VIII.
- **Test 6: RPR 8640 bit-flips per day Test** A variation of the tests above was tested: a scenario with an average of 8640 bit-flips per day, or one every 10 seconds. The bit-flip affects a random bit in a random variable. The results of this test are presented in Table VIII.

Each of the RPR tests was executed three times for each of the optimizations implemented: `libfixmath`, 5-coefficient and 7-coefficient Ganssle algorithms. The execution times are not presented since they do not reflect the behaviour or performance of the mechanism but the impact of the fault injection, which inserted an overhead of approximately 6 to 7 minutes, with an average of 11 minutes per test.

TABLE V  
RESULTS OF TEST 1: LOCKSTEP DETERMINISTIC TEST.

Test 1: Number of injected faults						
Number of faults	1	10	100	1000	10000	100000
Execution Time	299.6s	299.6s	300.5s	301.0s	301.1s	308.9s
SNR	138.9dB	138.9dB	138.9dB	138.9dB	138.9dB	138.9dB

### C. Energy Consumption

After analysing the impact of the optimizations in the architecture regarding the total execution time, it is also important to analyse the impact regarding the total energy consumption of the system. Table IX presents the power and

TABLE VI  
RESULTS OF TEST 2: LOCKSTEP DYNAMIC TEST.

Test 2: Number of injected faults						
	1	10	100	1000	10000	100000
Execution Time	300.1s	300.7s	301.0s	302.0s	302.4s	309.2s
SNR	138.9dB	138.9dB	138.9dB	138.9dB	138.9dB	138.9dB

TABLE VII  
RESULTS OF TEST 3: RPR NORMAL DISTRIBUTION TEST.

Optimization			
	<code>libfixmath</code>	5-coefficient Ganssle	7-coefficient Ganssle
#1	55.4dB	37.9dB	-62.3dB
#2	63.4dB	79.8dB	103.3dB
#3	-inf	82.1dB	94.7dB

energy (work) consumed by each of the implemented versions. The power was measured by a PL programmable power supply from TTI, measuring the power supplied to the whole system. Comparing the baseline single-core implementation against its dual-core version, it is possible to observe that the extra energy required for the second CPU core is around 150mW but allowed a reduction of almost half of the processing time. Depending on the approximation used, the mechanism uses from 30% to 40% more energy when compared to the baseline. Moreover, computing approximations makes use of simpler CPU instructions, therefore consuming less instant power. It can be observed that the processing time is what impacts the most in terms of energy consumption.

## VI. DISCUSSION

Two tests were designed to test the lockstep mechanism: a test where the fault injection targeted a pre-defined bit in a pre-defined variable and a test where the affected bits and variables were random.

As can be observed in Table V, the SNR values of the images generated by Test 1 were equal to the original, as expected, since the lockstep mechanism repeated the loading from memory in case of error. Nonetheless, the execution time increased due to the repeated executions. Similarly to Test 1, the SNR values of Test 2 were equal to the original, which was expected and the execution time was superior to the error-free execution.

Regarding the tests used to evaluate the RPR mechanism, they differed in the bit-flip rate per day. According to [19], the average of bit-flips per day is one. To evaluate the reliability of the solution, the tests had a bit-flip rate of one every 10 seconds, every 30 seconds and every minute.

Most of the results obtained by Test 3 are not considered acceptable, since the SNR values are inferior to 100dB, according to [4]. Two iterations, the third of `libfixmath` and the first of the 7-coefficient Ganssle algorithm were either minus infinite or a negative value, which generate a blank image.

The overall results the executions of Test 4 were close to the original SNR value of the image, except the first execution of the 7-coefficient Ganssle algorithm. The other iterations deviated from the original value a maximum of 4.1dB and an

TABLE VIII  
RPR BIT-FLIP TEST RESULTS

Test	Number of bit-flips per day	Iteration	Optimization		
			libfixmath	5-coefficient Ganssle	7-coefficient Ganssle
4	1440	1	138.9dB	138.8dB	19.9dB
		2	138.6dB	138.5dB	134.8dB
		3	138.8dB	138.8dB	138.8dB
5	2880	1	97.8dB	67.9dB	109.9dB
		2	8.3dB	129.1dB	34.4dB
		3	90.3dB	101.1dB	83.3dB
6	8640	1	-inf	-inf	-inf
		2	-0.4dB	-inf	nan
		3	-inf	nan	-inf

TABLE IX  
ENERGY AND POWER CONSUMPTION OF THE FAULT-TOLERANT ARCHITECTURE DEPENDING ON THE OPTIMIZATION.

	Original	Baseline (dual-core)	Optimization		
			libfixmath	5-coefficient Ganssle	7-coefficient Ganssle
<b>Power</b>	1.695W	1.841W	1.834W	1.829W	1.826W
<b>Energy</b>	808.13J	414.02J	541.93J	581.45J	587.25J

average of 0.65dB. The low SNR value of the first iteration of the 7-coefficient Ganssle algorithm is justified by the fault injection in random variables. Certain variables are more critical than others, for example, the final result of the approximation has a greater impact on the final image quality.

As can be observed from Table VIII, the overall SNR values obtained from Test 5 are inferior when compared to the results of Test 4, which was expected since the rate of bit-flips doubled. The 5-coefficient Ganssle algorithm provided the best results of this test: two out of three SNR values are considered acceptable and the other has a SNR almost half of the original value. The results obtained using the 7-coefficient Ganssle algorithm generate one acceptable image. For this test, the optimization which provided the best results was the 5-coefficient Ganssle algorithm.

Test 6 had the higher rate of fault injection, 10 bit-flips per second, as can be observed in Table VIII, the mechanism was not successful at detecting and correcting faults. The values in the results table are nan,  $-\infty$  or negative values, which generate a blank image. A SNR equal to nan happens when a bit-flip affects a floating-point variable and the resulting value is not considered a valid floating-point representation. Regarding the SNR of  $-\infty$ , the calculation of this metric involves a logarithm operation, which equals  $-\infty$  in C when calculating the logarithm of 0. The mechanism became ineffective due to the elevated rate of bit-flips, leading to the conclusion the mechanism is only able to tolerate faults up to a rate similar to test 5.

## VII. CONCLUSIONS

This work explored the development of an architecture for SAR imaging systems. This architecture consists of an on-board fault-tolerant system capable of generating SAR images using the Backprojection Algorithm in a space environment. The final architecture consists on a dual-core implementation of the Backprojection Algorithm, protected by two fault tolerance

mechanisms: lockstep and RPR. In spite of the limitations of the RPR mechanism, the algorithm was tested under pessimistic conditions, different from the average. Furthermore, the developed architecture with a mixed approach of lockstep and RPR was demonstrated to be a good alternative for intensive space applications.

## ACKNOWLEDGMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 and project SARRROCA, "Synthetic Aperture Radar Robust Reconfigurable Optimized Computing Architecture" with referencens PTDC/EEI-HAC/31819/2017, funded by FCT/MCTES, and POCI - Programa Operacional Competitividade e Internacionalização e PORLisboa – Programa Operacional Regional de Lisboa.

## REFERENCES

- [1] E. S. Cor Claeys, *Radiation Effects in Advanced Semiconductor Materials and Devices*. Springer-Verlag Berlin Heidelberg, 2002.
- [2] L. A. Tambara, "Analyzing the impact of radiation-induced failures in All Programmable System-on-Chip devices," 2017.
- [3] Xilinx, "Zybo FPGA board reference manual," 2017.
- [4] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo, *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*, Pacific Northwest National Laboratory and Georgia Tech Research Institute, December 2013, <http://hpc.pnnl.gov/projects/PERFECT/>.
- [5] D. Pritsker, "Efficient global back-projection on an FPGA," in *2015 IEEE Radar Conference (RadarCon)*, May 2015, pp. 0204–0209.
- [6] W.-C. Fang, C. Le, and S. Taft, "On-board fault-tolerant SAR processor for spaceborne imaging radar systems," in *2005 IEEE International Symposium on Circuits and Systems*, May 2005, pp. 420–423 Vol. 1.
- [7] A. Jacobs, G. Cieslewski, C. Reardon, and A. George, "Multiparadigm computing for space-based Synthetic Aperture Radar," pp. 146–152, 2008.
- [8] Á. B. de Oliveira, L. A. Tambara, and F. L. Kastensmidt, *Exploring Performance Overhead Versus Soft Error Detection in Lockstep Dual-Core ARM Cortex-A9 Processor Embedded into Xilinx Zynq APSoC*. Cham: Springer International Publishing, 2017. [Online]. Available: [https://doi.org/10.1007/978-3-319-56258-2\\_17](https://doi.org/10.1007/978-3-319-56258-2_17)
- [9] A. Shye, J. Blomstedt, T. Moseley, V. J. Reddi, and D. A. Connors, "PLR: A software approach to transient fault tolerance for multicore architectures," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 2, pp. 135–148, 2009.
- [10] B. Pratt, M. Fuller, and M. Wirthlin, "Reduced-precision redundancy on FPGAs," *International Journal of Reconfigurable Computing*, vol. 2011, p. 12, 2011. [Online]. Available: <http://dx.doi.org/10.1155/2011/897189>
- [11] A. Ullah, P. Reviriego, S. Pontarelli, and J. A. Maestro, "Majority voting-based reduced precision redundancy adders," *IEEE Transactions on Device and Materials Reliability*, 2017.
- [12] S.-J. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 849–854, 1994.
- [13] M. Didehban, S. R. D. Lokam, and A. Shrivastava, "Incheck: An in-application recovery scheme for soft errors," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.
- [14] H. Mushtaq, Z. Al-Ars, and K. Bertels, "Efficient software-based fault tolerance approach on multicore platforms," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 921–926.
- [15] J. Volder, "The cordic computing technique," in *Papers Presented at the March 3-5, 1959, Western Joint Computer Conference*, ser. IRE-AIEE-ACM '59 (Western). New York, NY, USA: ACM, 1959, pp. 257–261. [Online]. Available: <http://doi.acm.org/10.1145/1457838.1457886>
- [16] J. Ganssle, *The Firmware Handbook*. Orlando, FL, USA: Academic Press, Inc., 2004.
- [17] Xilinx, "Xilinx Software Command-Line Tool (XSCT) reference guide," 2016.
- [18] Xilinx, "OS and Libraries document collection," 2014.
- [19] ESA, *Herschel Observers' Manual*, ESA, Mar. 2014.