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

Helena Cruz
helena.cruz@tecnico.ulisboa.pt
Instituto Superior Técnico, Universidade de Lisboa

November 2018

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 processing data 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 for geology, agriculture, oceanography, glaciology, forestry and natural disasters. Backprojection is an algorithm for SAR image generation capable of generating high quality images, however, it is one of the most computationally intensive. 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 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, using a software-only approach, generating acceptable SAR images in a space environment. The solution was implemented on a Xilinx SoC device with a dual-core 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.

Keywords: Synthetic-Aperture Radar (SAR), Backprojection Algorithm, Multi-Core Fault Tolerance, Software Fault Tolerance

1. 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 [3]. These are capable of deteriorating the electronic systems and provoking bit-flips, leading to failures in electronic systems [15]. 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 [21].

2. Background

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

2.1. Synthetic-Aperture Radar (SAR)

Synthetic-Aperture Radar 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.

2.2. 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 [2]:

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 [13].

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

   - $g(n)$ - 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 θ_k.
4. Scales the sampled value by a matched filter to form the pixel contribution. This value is calculated using Eq. 2 [13]. dr is calculated using Eq. 3 [13].

$$e^{i\omega 2|\overrightarrow{r_k}|} = \cos(2 \cdot \omega \cdot dr) + i \sin(2 \cdot \omega \cdot dr) \quad (2)$$

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

   - 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 [13].

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

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

Algorithm 1 Backprojection algorithm pseudocode.
Source: PERFECT Manual Suite [2].

```
for all pixels k do
    f_k ← 0
    for all pulses p do
        R ← ||a_k − v_p||
        b ← ⌊(R − R0)/ΔR⌋
        if b ∈ [0, N_b p − 2] then
            w ← ⌊(R − R0)/ΔR⌋ − b
            s ← (1 − w) · g(p, b) + w · g(p, b + 1)
            f_k ← f_k + e^{i·k_u·R}
        end if
    end for
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.

2.3. Quality Assessment

The quality of a SAR image can be evaluated Signal-To-Noise Ratio (SNR) [2]. 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 [2], 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.

2.4. 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 a assumed to be correct. This mechanism is used in [7, 9].

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 to be used in future comparisons. A lockstep approach implemented as described above is presented in [4]. Process-Level Redundancy (PLR) [14], mentioned in section above, 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. 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 redudant computations. Examples of applications that use RPR are [12, 16].

Fault-tolerant versions of SAR image generation algorithms are presented in [9, 18, 7]. [9] proposes a fault tolerance mechanism for FFT Algorithm based on range and azimuth compression by implementing Concurrent Error Detection (CED) and using weighted sum, and also implements scrubbing. [18] also presents a mechanism for FFT algorithm based on a weighted checksum encoding scheme. [7] describes a Fault-Management Unit which is resposible 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.

3. Multi-Core Fault-Tolerant Architecture

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

3.1. Algorithm Profiling

The Backprojection algorithm implementation used in this study is part of the PERFECT Suite [2] and is written in C. 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 `O3`[1]. To profile this algorithm, **gprof**[2] was used.

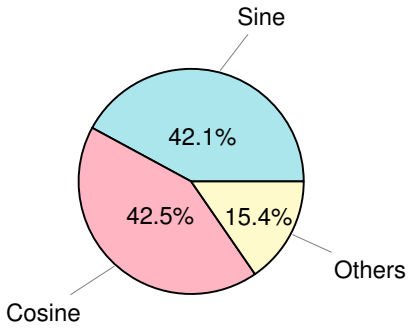

Figure 1: Backprojection algorithm profiling.

The trigonometric functions are responsible for over 80% 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.

3.2. 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

[1] `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`

[2] `https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html`

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, a mixed approach was developed:

- On the sections with less computations, a Lockstep mechanism was used, as seen in [4, 5, 14, 10]. This mechanism ensures the initialization is done properly and the raw SAR data is correct before beginning the image generation and that the output image is correctly saved.
- On the sections with more computations, a Reduced-Precision Redundancy (RPR) mechanism was developed. RPR is a special case of temporal redundancy where, instead of using full-precision computations for the redudant values, reduced-precision computations are used. RPR is used in several applications [12] 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 (BP) 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).

3.2.1 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 1, 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.

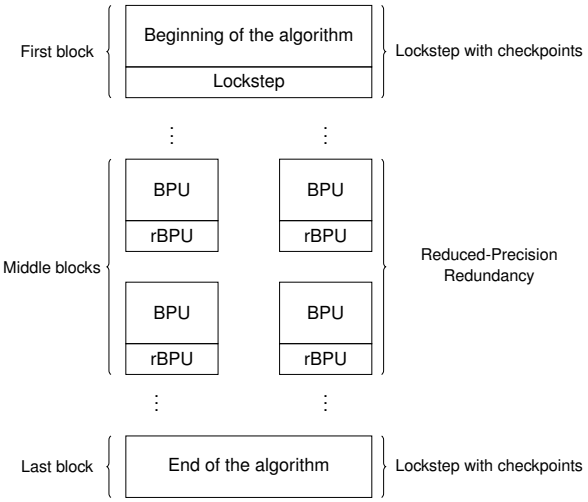

Figure 2: Architecture of the developed fault tolerance mechanism.

Table 1: 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	—	**240.4s**	**240.6s**	**241.5s**	**241.7s**
Core 1	—	239.9s	239.3s	241.0s	239.4s
Total	477.4s	480.3s	479.9s	482.6s	480.4s

From Table 1 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 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.

3.2.2 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 processor's registers are compared and, if they match, the execution is assumed correct. Afterwards, a checkpoint is created in memory containing these values. The checkpoints can be used later on to rollback from an incorrect execution to an error-free state. These checkpoints typically save the register values or variable values in memory.

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. Lastly, the lockstep is on once again to deal with the out-

put write, assuring the image is written correctly.

3.2.3 Reduced-Precision Redundancy

In RPR two values are calculated: Full-Precision (FP), or BPU, and Reduced-Precision (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.

Trigonometric Functions Optimization

The most optimizable section of the algorithm is the trigonometric functions. The optimization functions tested are described below and the results are presented in Table 2.

- COordinate Rotation DIgital Computer (CORDIC) algorithm [17];
- Taylor Series;
- Wilhem's Look-Up Table (LUT)[3];
- `libfixmath`[4];
- Ganssle optimizations [8].

One of the conclusions that can be drawn from the analysis of Table 2 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[5]. 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 tough 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 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.

Table 2: Comparison of optimization algorithms.

		Time	SNR
Original		477.4s	138.9dB
CORDIC	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
Taylor Series	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
Wilhem's Look-Up Table		123.2s	69.1dB
`libfixmath`	Taylor I	179.3s	54.5dB
	Taylor II	158.8s	33.6dB
	LUT	134.8s	99.2dB
Ganssle	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

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 Zybp 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 out-

[3] `https://www.atwillys.de/content/cc/sine-lookup-for-embedded-in-c/`

[4] `https://github.com/PetteriAimonen/libfixmath`

[5] `https://people.sc.fsu.edu/~jburkardt/c_src/cordic/cordic.html`

performed 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.

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 BP 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.

4. Implementation

The research work developed was tested on a Zybo Zynq-7000 board [21] 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 an On-Chip Memory (OCM) with 256kB of RAM and 128kB of ROM.

4.1. 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 3 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 3 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.

Table 3: Address range for each application.

	Processor 0 Application	Processor 1 Application
DDR Address Range	0x00100000 to 0x063FFFFF	0x06400000 to 0x0C7FFFFF
DDR Size	0x06400000	0x06400000
OCM 1 Address Range	0x00000000 to 0x0002FFFF	0x00000000 to 0x0002FFFF
OCM 1 Size	0x00030000	0x00030000
OCM 2 Address Range	0xFFFF0000 to 0xFFFFFDFF	0xFFFF0000 to 0xFFFFFDFF
OCM 2 Size	0x0000FE00	0x0000FE00

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) [20]. This can 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 Figure 3.

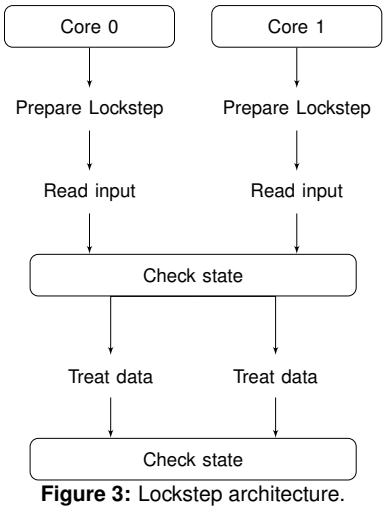

Figure 3: Lockstep architecture.

The first thing applications do when executing is 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 [14] 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. All of these values except the raw SAR data and the platform positions are saved in checkpoints. It is not possible to save the raw SAR data and the platform positions 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, making sure each application is reading their input from the correct address.

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 Figure 3, is the data treatment. This consists in a pre-processing of the input data before starting the algorithm. This preprocessing 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.

According to the lockstep description presented in the previous section, the output is written in memory whilst being protected by lockstep. In this implementation, however, the value of each pixel is written after its calculation, which means it is done during the RPR process. Before the end of the execution it is necessary to assure 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 [19] used to trigger flushes. Afterwards, the image can be written into a file using XSCT. To do this, command `mrd`, is used.

As aforementioned, in this implementation of lockstep only stores and compares selected variables, used to check the application correctness instead of registers. The register approach was implemented and tested, however, since this is a software approach, it was not successful. First of all, lockstep requires the applications to be identical and thus be able to compare the register values. Hardware implementations of lockstep use a hardware checker unit to detect errors and correct them, without differences in the software of each application. In a software implementation, the lockstep functions differ slightly between each application, leading to small but existent differences in the Assembly code. These generate an incorrect error detection. There are registers that cannot be protected by a lockstep mechanism since their values differ even in a fault-free execution. These registers are the R0, R1, R11 (frame pointer), R12 (used for intra-procedure call), Stack Pointer (SP), Link Register (LR) and Program Counter (PC) [11]. It is also important to mention that ARM deprecates the use of specific registers (PC, SP and LR) for any other purpose other than what they are specified for [1], making it impossible to detect control flow errors in these variables and correcting them. For this reason, the works mentioned above [11, 4] and the present work do not protect against control flow errors.

4.2. Reduced-Precision Redundancy

The Backprojection algorithm implementation used to test the FT mechanism [2] 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 contribuitions, 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.

5. Results

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

5.1. 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 4. 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 4: Comparison between the execution times depending on the optimization.

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

5.2. 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 5.
- **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 6.

Regarding the Reduced-Precision Redundancy mechanism, the objective is to observe the final quality of the generated images, using the SNR, in the presence of faults. To test the 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 [6], the average occurences 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 7.
- **Test 4: RPR 1440 bit-flips per day Test** Considering the average of bit-flips according to [6], 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 8.
- **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 9.
- **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 10.

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 perfomance 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 5: 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

Table 6: 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 7: Results of Test 3: RPR Normal Distribution Test.

	Optimization		
	`libfixmath`	5-coefficient Ganssle	7-coefficient Ganssle
#1	55.4	37.9	-62.3
#2	63.4	79.8	103.3
#3	-inf	82.1	94.7

6. 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 5, 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 Reduced-Precision Redundancy mechanism, they differed in the bit-flip rate per day. According to [6], the average of bit-flips per day is one. To evaluate the reliabilty 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 [2]. 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.

Table 8: Results of Test 4: RPR 1440 bit-flips per day Test.

	Optimization		
	`libfixmath`	5-coefficient Ganssle	7-coefficient Ganssle
#1	138.9dB	138.8dB	19.9dB
#2	138.6dB	138.5dB	134.8dB
#3	138.8dB	138.8dB	138.8dB

Table 9: Results of Test 5: RPR 2880 bit-flips per day Test.

	Optimization		
	`libfixmath`	5-coefficient Ganssle	7-coefficient Ganssle
#1	97.8dB	67.9dB	109.9dB
#2	8.3dB	129.1dB	34.4dB
#3	90.3dB	101.1dB	83.3dB

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 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 9, 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, and can be observed in Table 10, 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 a certain rate of faults.

7. Conclusions

This work explored the development of an architecture for SAR imaging systems. This architecture consists in 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 implemen-

Table 10: Results of Test 6: RPR 8640 bit-flips per day Test.

	Optimization		
	libfixmath	5-coefficient Ganssle	7-coefficient Ganssle
#1	-inf	-inf	-inf
#2	-0.4	-inf	nan
#3	-inf	nan	-inf

tation of the Backprojection Algorithm, protected by two fault tolerance mechanisms: lockstep and Reduced-Precision Redundancy.

Despite the limitations of the implemented lockstep mechanism, its ability to detect and correct errors was proved. Even with the time overhead, the lockstep was successfully able to recover from errors. 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.

8. Future Work

The work presented in this thesis leads to several enhancement possibilities, described below. One possibility is to implement dynamic load-balancing during the parallel execution of the Backprojection Algorithm. In this work, the algorithm optimizations were solely based on the trigonometric functions. Future work could include optimizations on other sections of the algorithm. The objective of the proposed architecture is to be able to generate images in space and broadcasting them to Earth. As an extended version of this work, a real-time Backprojection Algorithm could be implemented, allowing the on-board generation of images to be faster and opening new possibilities, such as video-SAR.

References

[1] ARM. *ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition*, 2014.

[2] 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/.

[3] E. S. Cor Claeys. *Radiation Effects in Advanced Semiconductor Materials and Devices*. Springer-Verlag Berlin Heidelberg, 2002.

[4] 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*. Springer International Publishing, Cham, 2017.

[5] 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)*, pages 1–6, June 2017.

[6] ESA. *Herschel Observers' Manual*. ESA, Mar. 2014.

[7] 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*, pages 420–423 Vol. 1, May 2005.

[8] J. Ganssle. *The Firmware Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[9] A. Jacobs, G. Cieslewski, C. Reardon, and A. George. Multiparadigm computing for space-based synthetic aperture radar, 2008.

[10] 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)*, pages 921–926, March 2013.

[11] d. B. d. Oliveira. Applying dual core lockstep in embedded processors to mitigate radiation induced soft errors. Master's thesis, Universidade Federal do Rio Grande do Sul, 2017.

[12] B. Pratt, M. Fuller, and M. Wirthlin. Reduced-precision redundancy on fpgas, 2011.

[13] D. Pritsker. Efficient global back-projection on an fpga. In *2015 IEEE Radar Conference (RadarCon)*, pages 0204–0209, May 2015.

[14] 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*, 6(2):135–148, 2009.

[15] L. A. Tambara. Analyzing the impact of radiation-induced failures in all programmable system-on-chip devices. 2017.

[16] 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.

[17] J. Volder. The cordic computing technique. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), pages 257–261, New York, NY, USA, 1959. ACM.

[18] S.-J. Wang and N. K. Jha. Algorithm-based fault tolerance for fft networks. *IEEE Transactions on Computers*, 43(7):849–854, 1994.

[19] Xilinx. Os and libraries document collection, 2014.

[20] Xilinx. Xilinx software command-line tool (xsct) reference guide, 2016.

[21] Xilinx. Zybo fpga board reference manual, 2017.