Fault-Tolerant Architecture for On-board Dual-Core Synthetic-Aperture Radar Imaging

Helena Cruz[1,2], Rui Policarpo Duarte[1,2(✉)], and Horácio Neto[1,2]

[1] INESC-ID, Rua Alves Redol, 9, Lisbon, Portugal
[2] Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal
{helena.cruz,rui.duarte,horacio.neto}@tecnico.ulisboa.pt

Abstract. In this research work, an on-board dual-core embedded architecture was developed for SAR imaging systems, implementing a reduced-precision redundancy fault-tolerance mechanism. This architecture protects the execution of the BackProjection Algorithm, capable of generating acceptable SAR images in embedded systems subjected to errors from the space environment. The proposed solution was implemented on a Xilinx SoC device with a dual-core processor. The present work was able to produced images with less 0.65 dB on average, than the fault-free image, at the expense of a time overhead up to 33%, when in the presence of error rates similar to the ones measured in space environment. Notwithstanding, the BackProjection algorithm executed up to 1.58 times faster than its single-core version without any fault-tolerance mechanisms.

Keywords: Synthetic-Aperture Radar · BackProjection Algorithm · Approximate computing · FPGA · Dual-core · SoC

1 Introduction

There is an increasing need for satellites, drones and Unmanned Aerial Vehicles (UAVs) to have lightweight, small, autonomous, portable, battery-powered systems able to generate Synthetic-Aperture Radar (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 2D and 3D images of Earth which is usually mounted on moving platforms such as satellites, aircrafts and drones.

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

C. Hochberger et al. (Eds.): ARC 2019, LNCS 11444, pp. 3–16, 2019.
https://doi.org/10.1007/978-3-030-17227-5_1

SAR can operate 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, oil spills, deforestation, fires, natural disasters such as hurricanes, volcano eruptions and earthquakes.

Space is a harsh environment for electronic circuits and systems as it can cause temporary or permanent errors on them. Therefore, systems designed for spacecrafts or satellites must be reliable and tolerate space radiation. The main radiation sources 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, 15, 20]. These radiation sources are capable of deteriorating the electronic systems and provoking bit-flips, leading to failures in electronic systems [2, 11, 14, 16]. Fault tolerance mechanisms are used to increase the reliability of these systems at the expense of extra mechanisms, processing time and power.

BackProjection is an algorithm for SAR image generation that is capable of generating high quality images. BackProjection is considered the reference algorithm for image formation since it does not introduce any assumptions or approximations regarding the image. However, it is a very computationally intensive algorithm. Therefore, typical fault-tolerance mechanisms will introduce a huge penalty on its performance.

System-on-Chip (SoC) Field-Programmable Gate Arrays (FPGAs) were chosen as a target device because of their power efficiency, performance and reconfigurability, which are very important characteristics for space systems. Furthermore, the use of a SoC FPGA will enable future developments of dedicated hardware accelerators to improve the performance of the system.

2 Background

2.1 Synthetic-Aperture Radar

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, as illustrated in Fig. 1. 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

The BackProjection algorithm takes the following values as input: number of pulses, location of the platform for each pulse, the carrier wave number, the radial

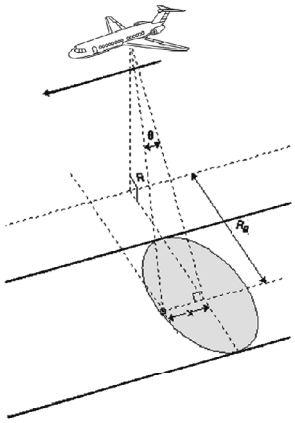

Fig. 1. Example of geometries involved in a SAR system.

distance between the plane and target, the range bin resolution, the real distance between two pixels and the measured heights. The BackProjection algorithm, from [1], performs the following steps for each pixel and each pulse:

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].

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

4. Scales the sampled value by a matched filter to form the pixel contribution. This value is calculated using Eq. 2, and dr is calculated using Eq. 3, as in [13].

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

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

5. Accumulates the contribution into the pixel. The final value of each pixel is given by Eq. 4 [13].

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

Table 1 summarizes the algorithm's variables and their meaning.

Table 1. Variables and their meaning

Variable	Meaning
$g(n)$	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
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
$f(x,y)$	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

Algorithm 1.1. BackProjection algorithm pseudocode.
Source: PERFECT Manual Suite [1].

```
1: for all pixels k do
2:     f_k ← 0
3:     for all pulses p do
4:         R ← ||a_k − v_p||
5:         b ← ⌊(R − R0)/ΔR⌋
6:         if b ∈ [0, N_b p − 2] then
7:             w ← ⌊(R − R0)/ΔR⌋ − b
8:             s ← (1 − w) · g(p, b) + w · g(p, b + 1)
9:             f_k ← f_k + e^{i·k_u·R}
10:        end if
11:    end for
12: end for
```

The pseudocode to compute the aforementioned steps is shown in Algorithm 1.1. k_u represents the wave number 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 BackProjection algorithm implementation used in this study was taken from the PERFECT Suite [1] and is written in C. This suite also contains three input image sets: small, medium and large, which produce images of sizes 512×512, 1024×1024 and 2048×2048 pixels, respectively.

2.3 SAR Image Quality Assessment

The metric used to evaluate the quality of a SAR image is the Signal-To-Noise Ratio (SNR). The SNR measures the difference between the desired signal and the background noise, see Eq. 5. The larger the SNR value, the greater the agreement between the pixel values. Values above 100 dB are considered reasonable [1].

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

- r_k - Reference value for *k-th* pixel.
- t_k - Test value for *k-th* pixel.
- N - Number of pixel to compare.

2.4 Fault-Tolerant SAR Image Generation

Precise fault-tolerant mechanisms consist of repetitions of the same operations in one or more units and evaluate which is the most voted result, regarding it as the correct one. The most common one is Triple Modular Redundancy (TMR) and consists of having three entities calculating the same value and have a voter entity compare the results. The most common output value is assumed to be the correct one. This mechanism is explained in [8,10]. In the aforementioned mechanism more than twice the power is consumed, and a latency overhead is always required.

Fault-tolerant versions of SAR image generation algorithms are presented in [8,10,19]. [10] proposes a fault tolerance mechanism for the 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. [19] also presents a mechanism for FFT algorithm based on a weighted checksum encoding scheme. [8] describes 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.

2.5 Approximate Computing Fault Tolerance

If small variations in the computation of image processing algorithms are introduced, they may not be perceptible at all. Therefore, such algorithms allow some deviations from the correct value while still having valid images. In this context, this paper proposes a novel fault-tolerance mechanism which relies on approximations of the computations when in the presence of errors.

Reduced-Precision Redundancy (RPR) is used to reduce the overhead introduced by TMR by using a full-precision computation and two reduced-precision

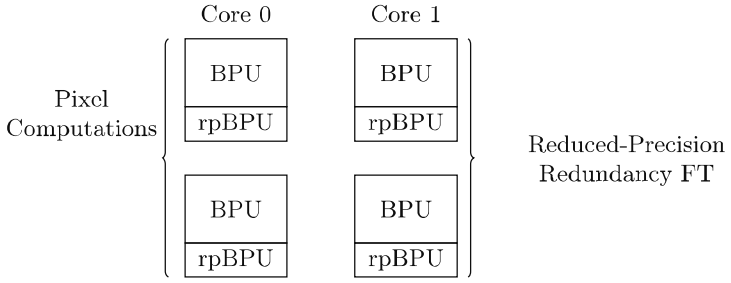

Fig. 2. Workload distribution of the developed fault tolerance mechanism between the CPU cores.

computations. RPR can be implemented in hardware, following an architecture similar to TMR, or software, following an architecture similar to temporal redundancy. The full-precision computation corresponds to the "original computation" and the other two computations to approximations. Computing the approximations reducing the overhead of the redundant computations, hence it is more efficient than calculating a full-precision values. However, the overhead of the voting process is kept constant. Examples of applications that use RPR are [12, 17].

In [4], the authors proposed a mechanism for Single Event Upset (SEU) mitigation, which relies only on the comparison of the full-precision result against only one approximation, obtained from a Look-Up Table (LUT). Due to the lack of precision, only the Most Significant Bits (MSbs) are compared. It they are equal, the full-precision result is passed to the output of the arithmetic units, otherwise, the approximate result is used. While it is not possible to determine which unit is the acting as the faulty one, the full-precision computation is always more prone to error than the reduced one.

3 Dual-Core Fault-Tolerant SAR Imaging Architecture

3.1 Proposed Architecture

In the BackProjection algorithm, the pixel computations are the most intensive set of computations.

The calculation of each pixel, or Backprojection Unit (BPU), is done in parallel, which means each core computes one pixel at a time. For this reason, it is protected by RPR, reducing the total overhead in the system. A scheme of the architecture of the fault tolerance mechanism is displayed in Fig. 2, where it is possible to observe which parts of the Backprojection (BP) algorithm are protected. The approximations are calculated after the full-precision computations. The approximation computation and the error detection are represented in Fig. 2 as Reduced-Precision Backprojection Units (rpBPUs).

Table 2. 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	Pixels in batch			
		4	8	16	32
Core 0	—	**240.4 s**	**240.6 s**	**241.5 s**	**241.7 s**
Core 1	—	239.9 s	239.3 s	241.0 s	239.4 s
Total	477.4 s	480.3 s	479.9 s	482.6 s	480.4 s

3.2 Algorithm Parallelization

In this algorithm, the pixel computations have no dependencies, therefore, they can be computed in parallel. The workload was divided between the cores statically since dynamic load-balancing introduces overhead in the system. The results of this test are presented in Table 2, where the execution time is presented in function of the number of pixels per batch. The tested number of pixels per batch was 4, 8, 16 and 32.

From Table 2 it is possible to conclude that the number of BPUs per batch does not have a significant influence on the total execution time since it is smaller or equal than 1%. 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 resulted in a similar execution time on both cores.

3.3 Modified Reduced-Precision Redundancy

This work uses a modified version of the RPR mechanism, which computes only one approximation (rpBPU) after computing the full precision result (BPU) to perform the comparison. The architecture of the modified RPR mechanism is presented in Fig. 3.

Both full-precision and reduced-precision values are compared by computing their difference. If the difference is greater than an acceptable threshold (T) from the reduced-precision value, it is assumed the value is incorrect and the reduced-precision value is used instead. If not, the full-precision result is assumed correct and is used. The reduced-precision value is copied to the output 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 reduced-precision values are calculated using the aforementioned optimizations.

3.4 Algorithm Profiling

To produce a reduced computation of the BPU it was necessary to profile the source code to determine which were the most time consuming operations.

Moreover, the operations that last longer are the ones that are more prone to be subjected to error. In future work, this profiling will also be important to determine which operations to port into a hardware accelerator.

For profiling, the software implementation of the BackProjection algorithm ran on the target device, Zynq FPGA from Xilinx, with the small image as input. It took approximately 8 min to generate this image, using the `o3`[1] optimization level. Other image sizes required processing times greater than 156 min. The implementation of the algorithm was profiled using **gprof**[2]. Table 3 shows the percentage of time dedicated to the most time consuming instructions in the BackProjection algorithm.

Table 3. BackProjection algorithm profiling.

Operation	Execution time (%)
Sine	42.05
Cosine	42.54
Others	15.41

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 within these functions. The rest of the algorithm, including input and output operations, is executed in under 16% of the time.

3.5 Trigonometric Functions Optimization

The optimizations for the trigonometric functions tested are described below and the results are presented in Table 4.

- COordinate Rotation DIgital Computer (CORDIC) algorithm [18];
- Taylor Series;
- Wilhem's LUT[3];

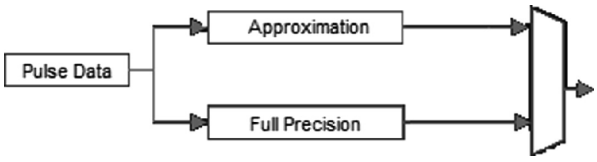

Fig. 3. The architecture of the modified RPR fault-tolerance mechanism.

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

Table 4. Comparison of the results produced by different optimization algorithms for the trigonometric functions.

Design	Variation	Time [s]	SNR [dB]
Baseline		477.4	138.9
CORDIC	10 iterations	238.8	60.5
	15 iterations	262.7	90.5
	20 iterations	286.3	120.2
	25 iterations	311.3	136.1
	30 iterations	335.1	136.3
Taylor Series	4 terms	186.0	71.8
	5 terms	192.3	103.8
	6 terms	201.5	133.6
	7 terms	210.4	135.3
Wilhem's Look-Up Table	n/a	123.2	69.1
Libfixmath	Taylor I	179.3	54.5
	Taylor II	158.8	33.6
	LUT	134.8	99.2
Ganssle	3 coefficients	163.5	66.3
	4 coefficients	167.3	105.2
	5 coefficients	170.7	118.3
	7 coefficients	176.5	134.8
		179.8	135.3

– `libfixmath`[4];
– Ganssle optimizations [9].

Observing the results on Table 4, the following conclusions can be drawn. 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, with all its tested versions being slower than any other version of another algorithm.

The results obtained from the Taylor Series algorithm were outperformed by the Ganssle methods, both in SNR and execution time.

The Wilhem's Look-Up Table method was the fastest overall and outperformed some variations of the other algorithms. It is a good alternative in systems with very limited memory since the LUT table occupies 66 bytes only, however, if memory does not represent an issue, the `libfixmath` library is a better alternative.

[4] https://github.com/PetteriAimonen/libfixmath.
[5] https://people.sc.fsu.edu/~jburkardt/c_src/cordic/cordic.html.

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 optimization.

The Ganssle optimizations are a good alternative to replace the trigonometric functions in the BackProjection algorithm. The first variation, the one that uses 3 coefficients to calculate the final result, is outperformed by both the 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 computing them in prone to error environments. 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 s, making the former a better alternative. The 5-coefficient variation provides more precision with an execution time increase of less than 36 s. The 7-coefficient (implemented with single precision) function provides a precision very similar to the original, with a difference of only less than 4 dB in the SNR, and an increase of less than 43 s.

To sum up, 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.

4 Implementation Results

The research design was implemented on a Pynq-Z2 board from TUL. This board contains a Zynq XC7Z020 device from Xilinx, an external 512 MB 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 FPGA. The PS main components are a dual-core ARM Cortex-A9 processor and a memory controller.

4.1 Precision Optimization Evaluation

Algorithm 1.1 was implemented using three precision reduction optimizations: 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 5. 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

Table 5. Comparison between the execution times depending on the optimization.

Design	Baseline (single core)	Baseline (dual core)	Libfixmath	Ganssle 5-coef.	Ganssle 7-coef.
Exec. time [s]	477.4	240.4	301.5	317.3	319.7

Table 6. Results of RPR with Agressive Fault-Injection.

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

`libfixmath` LUT method, the 5-coefficient and 7-coefficient Ganssle algorithms introduce an overhead of 25%, 32% and 33%, respectively.

4.2 Solution Evaluation

To test the developed architecture, a set of tests were performed. The fault injection was implemented in software and at compile-time by introducing bit-flips according to a specific distribution. Measurements performed in the L2 space were reported in[6] and on average there is one SEU per day. However, other locations in space induce more bit-flips.

Regarding the Reduced-Precision Redundancy mechanism, the objective was 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 RPR With Aggressive Fault Injection.** The average occurrences of bit-flips in space is 1 per day. To evaluate the mechanism on a more aggressive 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 6.
- **Test RPR With 1440, 2880 and 8640 Bit-Flips per Day.** Considering the average of bit-flips, a worse-case scenario was tested: an average of 1440 bit-flips per day, or one every 60, 30 and 10 s, respectively. The bit-flip affects a random bit in a random variable. The results of this test are presented in Table 7.

Each of the RPR tests was executed three times for each of the optimizations implemented: `libfixmath`, 5-coefficient and 7-coefficient Ganssle algorithms.

[6] http://herschel.esac.esa.int/Docs/Herschel/html/ch04s02.html.

Table 7. Results of RPR with 1440, 2880, and 8640 bit-flips per day.

	Optimization			
	`libfixmath`	5-coefficient Ganssle	7-coefficient Ganssle	
1440	#1	138.9 dB	138.8 dB	19.9 dB
	#2	138.6 dB	138.5 dB	134.8 dB
	#3	138.8 dB	138.8 dB	138.8 dB
2880	#1	97.8 dB	67.9 dB	109.9 dB
	#2	8.3 dB	129.1 dB	34.4 dB
	#3	90.3 dB	101.1 dB	83.3 dB

5 Discussion

The overall results for the executions with injection of 1440 bit-flips 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 4.1 dB and an average of 0.65 dB, when in the presence of errors. 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.

Most of the results for very aggressive error rates were not considered acceptable, since the SNR values are inferior to 100 dB. 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 SNR values obtained for 2880 bit-flips are inferior when compared to the results of 1440 bit-flips, 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.

The rate of 8640 bit-flips represents a fault injection of 10 bit-flips per second. At this rate the proposed 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.

6 Conclusions and Future Work

This work explored the research and development of a fault-tolerant architecture for SAR imaging systems capable of generating SAR images using the Backprojection Algorithm in a space environment.

The modified RPR mechanism proposed avoids the use of more costly mechanisms, such as TMR, while taking advantage of the dual-core processor on the Zynq device to improve performance. The main drawback of this mechanism, is the inability to detect or correct control errors.

The final architecture consists of a dual-core implementation of the Backprojection Algorithm, protected by the modified Reduced-Precision Redundancy mechanism. Depending on the optimization used, the overhead of the fault tolerance mechanism ranges from 25% to 33% when compared to the dual-core version of the Backprojection Algorithm.

In spite of the limitations of a software implementation the modified RPR mechanism, the algorithm was tested under pessimistic conditions, different from the average use scenario. Furthermore, the developed architecture with an approach of RPR was demonstrated to be a good alternative for intensive space applications. Future work involves exploring optimization techniques such as the ones described in [5–7].

References

1. Barker, K., et al.: 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/
2. Baumann, R.C.: Radiation-induced soft errors in advanced semiconductor technologies. IEEE Trans. Device Mater. Reliab. **5**(3), 305–316 (2005). https://doi.org/10.1109/TDMR.2005.853449
3. Claeys, C., Simoen, E.: Radiation Effects in Advanced Semiconductor Materials and Devices. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04974-7
4. Duarte, R.P., Bouganis, C.: Zero-latency datapath error correction framework for over-clocking DSP applications on FPGAs. In: 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), pp. 1–7, Deceember 2014. https://doi.org/10.1109/ReConFig.2014.7032566
5. Duarte, R.P., Bouganis, C.S.: High-level linear projection circuit design optimization framework for FPGAs under over-clocking. In: 2012 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 723–726. IEEE (2012)
6. Duarte, R.P., Bouganis, C.-S.: A unified framework for over-clocking linear projections on FPGAs under PVT variation. In: Goehringer, D., Santambrogio, M.D., Cardoso, J.M.P., Bertels, K. (eds.) ARC 2014. LNCS, vol. 8405, pp. 49–60. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05960-0_5
7. Duarte, R.P., Bouganis, C.S.: ARC 2014 over-clocking KLT designs on FPGAs under process, voltage, and temperature variation. ACM Trans. Reconfigurable Technol. Syst. **9**(1), 7:1–7:17 (2015). https://doi.org/10.1145/2818380

8. Fang, W.C., Le, C., Taft, S.: On-board fault-tolerant SAR processor for spaceborne imaging radar systems. In: 2005 IEEE International Symposium on Circuits and Systems, vol. 1, pp. 420–423, May 2005. https://doi.org/10.1109/ISCAS.2005.1464614
9. Ganssle, J.: The Firmware Handbook. Academic Press Inc., Orlando (2004)
10. Jacobs, A., Cieslewski, G., Reardon, C., George, A.: Multiparadigm computing for space-based synthetic aperture radar (2008)
11. Maki, A.: Space radiation effect on satellites. Joho Tsushin Kenkyu Kiko Kiho **55**(1–4), 43–48 (2009)
12. Pratt, B., Fuller, M., Wirthlin, M.: Reduced-precision redundancy on FPGAs (2011). https://doi.org/10.1155/2011/897189
13. Pritsker, D.: Efficient global back-projection on an FPGA. In: 2015 IEEE Radar Conference (RadarCon), pp. 0204–0209, May 2015. https://doi.org/10.1109/RADAR.2015.7130996
14. Sinclair, D., Dyer, J.: Radiation effects and cots parts in smallsats (2013)
15. Sørensen, J., Santin, G.: The radiation environment and effects for future ESA cosmic vision missions. In: 2009 European Conference on Radiation and Its Effects on Components and Systems, pp. 356–363, September 2009. https://doi.org/10.1109/RADECS.2009.5994676
16. Tambara, L.A.: Analyzing the impact of radiation-induced failures in all programmable system-on-chip devices (2017)
17. Ullah, A., Reviriego, P., Pontarelli, S., Maestro, J.A.: Majority voting-based reduced precision redundancy adders. IEEE Trans. Device Mater. Reliab. **PP**(99), 1 (2017). https://doi.org/10.1109/TDMR.2017.2781186
18. Volder, J.: The cordic computing technique. In: Papers Presented at the 3–5 March 1959, Western Joint Computer Conference, IRE-AIEE-ACM 1959 (Western), pp. 257–261. ACM, New York (1959). https://doi.org/10.1145/1457838.1457886
19. Wang, S.J., Jha, N.K.: Algorithm-based fault tolerance for FFT networks. IEEE Trans. Comput. **43**(7), 849–854 (1994). https://doi.org/10.1109/12.293265
20. Ya'acob, N., Zainudin, A., Magdugal, R., Naim, N.F.: Mitigation of space radiation effects on satellites at low earth orbit (LEO). In: 2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), pp. 56–61, November 2016. https://doi.org/10.1109/ICCSCE.2016.7893545