Received 12 June 2024, accepted 25 June 2024, date of publication 1 July 2024, date of current version 9 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3421608

RESEARCH ARTICLE

A Parity-Based Dual Modular Redundancy Approach for the Reliability of Data Transmission in Nanosatellite's Onboard Processing

ALEX C. R. ALVES[1], LUIZ F. Q. SILVEIRA[2], (Member, IEEE), MÁRCIO E. KREUTZ[3], AND SAMAHERNI M. DIAS[4]

[1]Electrical and Computer Engineering Graduate Program, Federal University of Rio Grande do Norte (UFRN), Natal 59078-970, Brazil
[2]Department of Computer Engineering and Automation, UFRN, Natal 59078-970, Brazil
[3]Department of Computer Science and Applied Mathematics, UFRN, Natal 59078-970, Brazil
[4]Department of Electrical Engineering, UFRN, Natal 59078-970, Brazil

Corresponding author: Alex C. R. Alves (alex.alves@ufrn.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brazil (CAPES)-Finance Code 001.

ABSTRACT Nanosatellites' embedded systems must adapt to power, weight, size, and cost constraints. Thus, over the years, the use of Commercial-Off-The-Shelf (COTS) System-on-Chip (SoC) has become common. They have a lower development cost and better performance when compared to components specifically designed for space, although they are more susceptible to the radiation effects. A point of attention in these devices is the reliability of data transmitted between hardcore processors and applications in the reconfigurable logic area. This work proposes a Parity-based Dual Modular Redundancy (PDMR) approach for use in interconnect interfaces of COTS SoCs. The experiments were conducted through simulations with Python scripts and hardware implementations in a Xilinx Zynq-7000 SoC. The proposed technique was compared with the Triple Modular Redundancy (TMR) technique. The simulation results show that for specific rates, the proposed approach reaches values close to those of the TMR and implies a smaller number of bits transmitted even when data detected as erroneous are retransmitted. Meanwhile, hardware implementation results demonstrate a decrease in hardware resource utilization and power consumption compared to TMR implementation.

INDEX TERMS Nanosatellite, data transmission, fault tolerance, FPGA, DMR, parity.

I. INTRODUCTION

Recent microelectronic innovations allow for smaller space systems known as pico (0.1–1 kg) and nanosatellite (1–10 kg), which have significant power, weight, size, and cost constraints [1]. Nowadays, it is usual to adopt Commercial-Off-the-Shelf (COTS) electronic components in the design of onboard processing systems for nanosatellites. These devices offer a low cost to the project and are years ahead of parts specially designed for space usage in terms of performance [2], [3]. However, the harsh space environment can be a risk for space missions with COTS components since they are more susceptible to radiation effects, such as Single Event Upsets (SEUs) and Single Event Transients (SETs) [4].

SEUs are caused by the impact of heavy ions or protons on bistable elements, such as flip-flops or other memory cells, which can generate a change of state (bitflip) [5]. Meanwhile, SETs occur when charged particles hit the integrated circuit, causing momentary voltage excursions. Transients in logic gates can propagate and be captured by storage elements (flip-flops or latches), becoming an SEU and consequently affecting the component's behavior [5], [6].

Given the growing demand for processing capacity in nanosatellites, COTS Systems-on-Chip (SoCs) that integrate

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria De Munari.

processing cores and Field-Programmable Arrays (FPGAs) have been increasingly adopted in missions with small satellites [7]. COTS SoCs enable hybrid processing, in which an embedded hardcore processor can perform part of the processing, and the rest of the computations are performed by a hardware acceleration in the reconfigurable logic (FPGA). Since these devices are based on SRAM (Static Random Access Memory) FPGAs whose configuration memory is subject to Single Event Upsets (SEUs), several works have investigated the radiation effects and methods to mitigate them [8], [9], [10], [11].

In [12], the authors state that interconnection modules (communication interfaces) affected by SEUs can be a source of errors in architectures that use hardware acceleration. Benevenuti and Kastensmidt [13] mention that replacing interconnection modules with fault-tolerant versions may be necessary to increase the reliability in a system that includes a hardware accelerator module with fault tolerance. However, these works do not discuss methods to mitigate data transmission errors caused by faults in the interconnection modules. For instance, in [13], hardware redundancy techniques are only applied to the hardware acceleration module, not the communication architecture.

Different redundancy techniques can be applied to mitigate possible errors caused by radiation. The information redundancy involves adding extra bits to the data so that they can be used to detect and even correct bitflips. The correction capability depends on the coding scheme [14]. Those that allow correction without retransmitting the information are referred to as Forward Error Correction (FEC) codes.

Several FEC codes are described in the literature [15]. The complexity of implementing these codes may vary according to the objective of the application. In [16], the authors describe the complexity of classical error-correcting codes, such as Hamming, Reed-Muller, Golay, Reed-Solomon, Convolutional, and Turbo codes. Most of these codes involve sequential circuits in the encoder or decoder, which can be an issue in systems with time constraints. Other codes, such as Hamming and Reed-Muller, may involve implementing a matrix generator in the encoder, which can directly impact logical complexity. The authors state that due to the limited energy budget of an application, only codes of limited complexity may be considered. Therefore, selecting the code to be implemented is crucial for systems with power consumption constraints.

FEC codes have interesting correction capabilities, but the number of redundant bits can increase considerably depending on the code implemented. Therefore, its use in standard protocols may involve sending the message and checksum in different clock cycles or using extra bus lines to include redundant bits [17], [18]. Considering these characteristics and the complexity of FEC codes, their implementation can impact the development time and power consumption, which is at odds with what is desired for applications with nanosatellite platforms [1], [19], [20]. In this context, modern FEC codes may have limited application in embedded systems and on-chip communication, even though the use of Single Error Correction (SEC) and Single Error Correction Double Error Detection (SECDED) codes are verified, mainly for protecting memory components [17], [21]. Another important observation about FEC codes concerns that retransmission turns out to be more efficient than correction from an energy viewpoint [22], [23].

Hardware redundancy features a more straightforward implementation than FEC schemes and has been used in several applications to mitigate SETs and SEUs [17], [24]. One of the most common hardware redundancy techniques is Triple Modular Redundancy (TMR), which consists of three identical hardware components, all with the same input and performing the same tasks. The outputs of these components are compared by a voting circuit, which sides the system output to be the majority result [25], characterizing the TMR ability of fault masking. So, if one element has failed, generating an incorrect output, the voter establishes that the system output will be the same as the output of the two components that supposedly operate correctly. However, if two or three elements have failed, the voter agrees with the majority, and the system output is incorrect [25].

The classical voter logic for individual bits is defined as [25] and [26]:

$$v = x \cdot y + x \cdot z + y \cdot z, \tag{1}$$

where x, y, and z are the outputs of the redundant digital components.

Despite TMR's widespread usage in radiation effect mitigation, its disadvantages are related to the increase in more than 200% of the area and power overhead [9]. According to Siegle et al. [9], another drawback is related to the impact on the performance of a circuit, especially when there are many TMR partitions.

A possible methodology to overcome the area overhead issue of TMR is the approximate TMR (ATMR) [27]. It is based on replacing TMR modules with approximate logic circuits, which must perform a logic function that may differ but must be closely related to the function of the original circuit [27], [28]. Although using ATMR reduces the area occupied by redundant modules, according to Arifeen et al. [29], designing an ATMR is a challenging task due, among other characteristics, to the extensive search space and computational complexity. They also state that ATMR is more vulnerable to errors than TMR. Their work contains important references for the design and application of ATMR.

The difficulties inherent to the ATMR design [27], [29] distract from our objective of proposing a low-complexity solution for data transmission structures. As communication interfaces are generally composed of control and data signals, the use of ATMR may involve the reduction of circuits whose logical functions are related to the behavior of these signals. The reduced circuits must guarantee data transfer and the correct operation of control signals. The identification of such circuits adds extra complexity to design development. So, it may become unfeasible when considering the restrictions imposed by standard protocol specifications. For instance, the Advanced eXtensible Interface (AXI) protocol specification of the Advanced Microcontroller Bus Architecture (AMBA) standard [30] highlights that there must be no combinatorial paths between input and output paths on master and slave interfaces. In FPGAs, achieving simplification may be possible by applying ATMR to the Look-Up Tables (LUTs) that make up the interconnect interface [28]. However, this fine-grained implementation is beyond the scope of our work.

An alternative to TMR and the complexity of FEC codes is the Dual Modular Redundancy (DMR), which is based on duplicating logic elements. In this case, errors in the system can be detected by comparing the bits of each redundant component. Compared to the TMR, the DMR occupies a smaller area, but it only allows error detection. Consequently, one can not take the output of one of the elements as correct. Moreover, it is a straightforward protection approach compared to FEC codes.

Aiming to expand the DMR to have error mitigation capability, a parity-based DMR approach was developed to be applied to communication interfaces (interconnection) of COTS SoCs that integrate hardcore processors and reconfigurable logic (FPGA) on the same chip. The main objective is to increase the reliability of data transmission in these devices, which are increasingly used in space missions with nanosatellites. We also seek to contribute with an alternative to hardware redundancy techniques by providing a method with fault mitigation values close to TMR in the face of different error rates, with less hardware resource utilization and power consumption.

The proposed approach, like TMR, allows fault masking. Moreover, error indication components enable the detection of potentially erroneous messages, which can facilitate retransmission actions. To achieve these characteristics, we focus on a solution with low implementation complexity in terms of logic elements.

Tests were carried out through software simulations and hardware implementations, in which the technique was applied to the AXI communication interfaces of the AMBA standard and compared with the TMR in the same scenarios.

The remainder of the paper is organized as follows: Section II provides an overview of related works; Section III presents the proposed approach, describing the formation rule and the logical expressions; Section IV describes the software simulations and their results; Section V provides hardware area and power occupancy, based on implementations for a Xilinx Zynq-7000 SoC; Section VI contains a comparison of some properties of the proposed approach, TMR and FEC codes; Section VII presents the final remarks.

II. RELATED WORKS

The reliability of communication modules was investigated in the work by De Sio et al. [12]. The authors present a reliability analysis of the AXI Interconnect IP module implemented in a Zynq-7000 SoC for connecting the processing system (ARM) and the reconfigurable logic (FPGA) integrated into the same chip. For this, faults were injected in the sections of the FPGA configuration memory occupied by the module. Furthermore, two hardware accelerators were developed and implemented in the FPGA along with the AXI Interconnect. During the testing routine, the processing system configured the accelerators and stimulated each with different inputs. The detected failures were observed and classified as follows: both accelerators stopped working; both had faulty calculations; only one accelerator continued to function correctly; the processing system accessed only one accelerator, but it returned erroneous results. Additionally, it was observed that the returned values were the same when both accelerators produced wrong calculations. Given the results, the authors conclude that the AXI Interconnect module can be a source of errors in architectures that use hardware acceleration. In [31], the authors further investigate the implications of radiation-induced SEUs on the AXI Interconnect module.

The redundancy of internal buses for low-speed peripherals has been explored by Lázaro et al. [32]. In their work, the authors propose a solution composed of an Intellectual Property (IP) Core called AXILiteRedundant and a redundant system. IP Core was used to triple the signals from a slave interface to master interfaces, which connect to external redundant peripherals. Also, the values from multiple masters to the slave were selected by a TMR voter. The IP Core could also indicate to the processor if all three masters had the same values, one of them had a different value (which could be corrected by the voter), or if they all had different values, signaling an error. The authors describe the modifications performed to the AXI standard to apply the redundancy mechanism and the elements that make up the redundant system. Finally, they conclude that the proposal protects the interconnection and peripheral IPs, requires few FPGA resources for implementation, and has negligible power consumption. Although this work stands out as an example of the application of TMR in communication interfaces, the lack of some descriptive elements related to the implementation and the specific nature of the IP made comparison with our approach difficult.

Bertozzi et al. [23] explore the detection capability and energy-reliability tradeoff of different coding schemes for on-chip communication links. The authors focus on schemes with Hamming codes, CRC codes, and a simple single-parity bit code. In this work, the Hamming-based correction codes allow the correction of single errors, while the CRC and the parity bit use retransmission as a recovery strategy. The authors present an interesting result for the subset of codes considered (CRC and Hamming), concluding that, from an energy viewpoint, retransmission turns out to be more efficient than correction. Despite the capability of CRC codes, their basic implementations only deal with error detection, and their complexity depends on the choice of the generator polynomial.

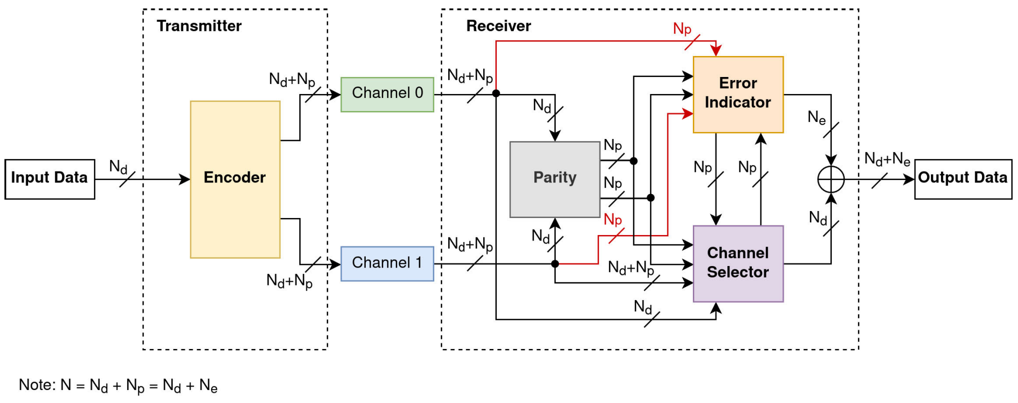

FIGURE 1. Data transmission with a parity-based DMR approach.

Mach et al. [17] present a review of existing protection approaches against transient faults of on-chip bus interconnects. The work focuses on embedded processors used in applications such as the automotive or space sectors, which have safety/mission-critical characteristics. The described protection approaches are information redundancy, temporal, and spatial (hardware). The authors state that implementing information redundancy is not straightforward, though it is viable in high-performance applications. They highlight parity and Error Detection and Correction (EDAC) codes, such as SECDED. In addition, regarding temporal redundancy, they point out that it requires accurate analysis of transient glitch durations and is suitable for low-performance applications. On the other hand, spatial redundancy (such as TMR) has a greater impact on area and power consumption but presents a straightforward implementation. The authors conclude that the selection of the protection approach depends on factors such as the processor use case, the operating environment, and the possibility of performing modifications to components external to the processor.

Regarding DMR with parity, in [33], a parity method (based on a cascade) is applied to cover all possible faults in the routing and LUT content using a totally self-checking circuit with a parity generator. The structure is duplicated so that if a fault is detected in an FPGA, the output of the one that was not affected is enabled. Clark et al. [34] apply DMR to detect errors at the Register File (RF) write-back stage of the pipeline, and SEU correction is achieved through parity that detects RF entry nibbles that are correct in one DMR copy but not the other. The correct contents are then copied from the DMR copy with correct parity to the other.

These works illustrate the application of DMR and parity in the architecture of memory components. During our state-of-the-art review, we could not find works related to the specific use of DMR or DMR with parity for data transmission. Our approach is combinational, and the original data is divided into groups with associated parity bits. This way, an error in different groups can be masked/corrected when considering logical expressions developed from the parity bits, as seen throughout the work.

III. PROPOSED APPROACH

In the parity-based DMR approach, we assume a data transmission structure where DMR is applied, such as the one presented in Fig. 1. We also assume that the transmission channels are susceptible to radiation effects. The input data is considered to have N_d bits. Initially, this data passes through an encoder (Encoder module in Fig. 1), which adds the parity bits (N_p-bit width) to the original message and forwards the codeword (input data + parity bits yielding an N-bit width word) to the transmitter that duplicates it and transmits each one over a digital channel (Ch0 and Ch1). At the receiver, the data from each channel is checked to determine which channel can deliver the correct message (set of bits).

Additionally, the parity bits are replaced by error indication bits (N_e-bit width), which have the same width as the parity bits ($N_e = N_p$). According to the need and implementation, the error indication bits (e_i) can serve as an auxiliary signal to take error mitigation actions, such as data retransmission or message discarding by a processing unit. In short, when presenting a logical value of 1, the error indication bits indicate an error in part of the received message, as described in this section.

Fig. 1, displays the modules of the proposed approach. Note that bus widths are indicated by N_d, N_p, and N_e. Also, the modules in the receiver will be referenced throughout this section.

A. WORD STRUCTURE AND PARITY GENERATION

As previously stated, for an input word of N_d bits (hereafter referred to as information bits), there is an output word with N bits, where N is a power of 2 greater than N_d. The number of parity bits the transmitter adds is given by $N_p = N - N_d$, where N_p must be a power of 2 less than or equal to N_d. Each parity bit is associated with a certain amount of information bits. This amount is referred to as M and can be calculated by

$$M = N/N_p - 1. \tag{2}$$

Thus, one parity bit plus M information bits constitute a group, and the total of groups equals the value of N_p. Fig. 2

illustrates an N-bit codeword. The index of the information bits t ranges from 0 to L, where $L = N_d - 1$.

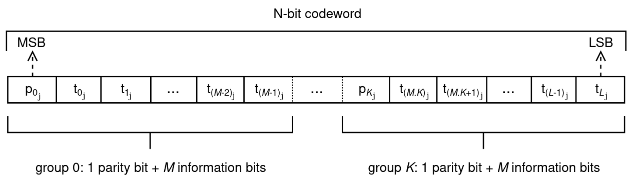

FIGURE 2. N-bit codeword with parity bits.

For the proposed approach, an even parity code was chosen, and it is obtained by

$$p_i = t_{(Mi)} \oplus t_{(Mi+1)} \oplus \cdots \oplus t_{(Mi+M-2)} \oplus t_{(Mi+M-1)}, \tag{3}$$

where index i ranges from 0 to K, for $K = N_p - 1$. This index defines to which parity and group the set of information bits belongs. After passing through the channels, the parity bits are indicated by $\boldsymbol{p}_{i_j}$, where the subscript j determines the respective channel, assuming 0 or 1. It is worth noting that parity is also generated at the receiver (Parity module in Fig. 1), similarly to (3) and referenced by r_{i_j}, since the parity bits and information bits may be corrupted after passing through the channels.

B. CHANNEL SELECTOR, ERROR FLAG, COMPARATOR, AND OUTPUT

The codeword formation rule allows the original data to be divided into groups. Based on a logic implementation in the receiver, the parity bits can be used to generate the receiver's output data, which may be formed by information bits from groups of different channels. For example, suppose the original data was divided into two groups with one parity bit each, as shown in Fig. 3. When passing through the receiver, the first slice (group without parity bit) of the receiver's output data can be formed by information bits from Channel0 (Ch0). In contrast, the second slice can come from Channel1 (Ch1), as long as the transmitted parity bits and the ones calculated at the receiver indicate the absence of errors in these groups.

The selection of the channel (Channel Selector module in Fig. 1) each slice will be obtained from depends on the proposed logical expression:

$$s_i = (\boldsymbol{p}_{i_0} \oplus r_{i_0}) + (\boldsymbol{p}_{i_1} \cdot \overline{r_{i_0}} \cdot r_{i_1}) + (\overline{\boldsymbol{p}_{i_1}} \cdot r_{i_0} \cdot \overline{r_{i_1}}). \tag{4}$$

In (4), $\boldsymbol{p}_{i_0}$ corresponds to the parity bit of the i-th group received by the receiver from the Ch0, $\boldsymbol{p}_{i_1}$ stands for the i-th parity bit received from the Ch1, r_{i_0} is the i-th parity bit of the Ch0 computed at the receiver, and r_{i_1} is the i-th parity bit of the Ch1 computed at the receiver. Observe that $\boldsymbol{p}_{i_0}$ and $\boldsymbol{p}_{i_1}$ are represented by bold letters to indicate that they may be corrupted bits after passing through the channels. Also, s_i is the channel select bit associated with the i-th group. The logical value 0 for s_i means that the information bits of group i, coming from Ch0, must be part of the output message, while a logical value 1 means that those coming from Ch1 must be considered. The truth table for (4) can be observed in Table 1. It should be noted that Ch0 is used as a default for the output when $\boldsymbol{p}_{i_0}, \boldsymbol{p}_{i_1}, r_{i_0}$, and r_{i_1} have the same logical value (referred to as *SLV cases*). Meanwhile, Ch1 is the default channel when two parity bits have the same value and the other two have the same opposite value (e.g., $\boldsymbol{p}_{i_0} = 0$, $\boldsymbol{p}_{i_1} = 0$, $r_{i_0} = 1$, $r_{i_1} = 1$). When all parity bits are equal, it is assumed that the information bits of the channels are equal, and any of the channels can be considered for the receiver's output data. In the other case, although two parity bits are equal and the other two are equal with inverse value, it is not inferred which channel can deliver the correct bits.

TABLE 1. The truth table for the channel selector and error flag.

$\boldsymbol{p}_{i_0}$	$\boldsymbol{p}_{i_1}$	r_{i_0}	r_{i_1}	s_i	f_i
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	1

Based on the parity bits ($\boldsymbol{p}_{i_0}, \boldsymbol{p}_{i_1}, r_{i_0}$, and r_{i_1}), a logic was elaborated to add error detection capacity to the proposed approach through error indication bits (Error Indicator module in Fig. 1). During the development of logical expressions, two scenarios were considered. In the first one (*S1*), the error indication bits are obtained considering only the parity bits. Thus, the error flag was designed to simplify the logical implementation and evaluate the behavior of the solution without the need to compare information bits from both channels. For *S1*, the error flag (f_i) and the error indication bit (e_i) are obtained from

$$f_i = (\boldsymbol{p}_{i_0} \oplus r_{i_0}) \odot (\boldsymbol{p}_{i_1} \oplus r_{i_1}), \tag{5}$$

$$e_i = s_i \cdot f_i, \tag{6}$$

where the $\odot$ operator in (5) represents a logical *XNOR* operation.

The last column of Table 1 contains the binary values for (5). Although the error flag is set for *SLV cases* (first and last rows of Table 1), the error indication bit (e_i) is based on an *AND* operation with f_i, which means that these cases are disregarded for *S1* (the operation results in the binary value 0). The fact that f_i is set for *SLV cases* also allows its logical expression to be written in terms of *XOR* and *XNOR* operations, which reduces the number of logic gates and gate inputs. If f_i were 0 for these cases, its expression would be composed of six minterms (table rows in which f_i is set), each with four logical variables. Another advantage of expressing

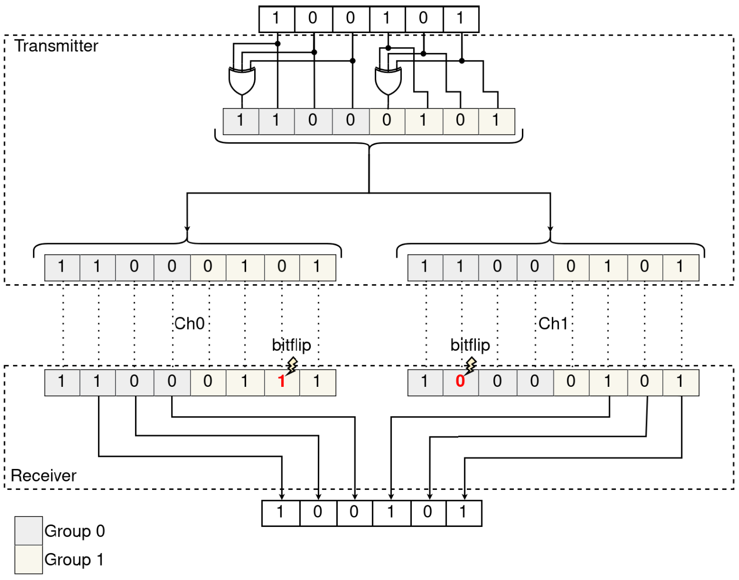

FIGURE 3. PDMR example with 6 information bits and 2 parity bits.

f_i as (5) is that this logical expression can be used for both scenarios (*S1* and *S2*). Since, for *S1*, we disregard *SLV cases* as a source of possible errors, it is assumed that these cases imply equal and correct information bits on both channels. The impact of this assumption is verified through simulation results.

In the second scenario (*S2*), the *SLV cases* are taken into account because, although all parity bits have the same logical value, it is assumed that information bits from the two channels associated with these parity bits may have different values, which indicates an error in one of the channels. In this scenario, the channel information bits must be compared using comparator logic, which determines whether they are the same on both channels. The comparator is part of the Error Indicator module represented in Fig. 1. Therefore, the width of the buses in red lines becomes $N_d + N_p$ for (S2) since the information bits must be compared. Furthermore, the comparison is performed through a bitwise operation between the information bits of the channels, which may be corrupted. The comparison bit is found by

$$c_i = \boldsymbol{t}_{(Mi)_0} \oplus \boldsymbol{t}_{(Mi)_1} + \boldsymbol{t}_{(Mi+1)_0} \oplus \boldsymbol{t}_{(Mi+1)_1} + \cdots + \boldsymbol{t}_{(Mi+M-2)_0} \oplus \boldsymbol{t}_{(Mi+M-2)_1} + \boldsymbol{t}_{(Mi+M-1)_0} \oplus \boldsymbol{t}_{(Mi+M-1)_1}, \tag{7}$$

where subscripts 0 and 1 indicate bits from Ch0 and Ch1, respectively, and the bold letters represent the information bits after passing through the channels. It is verified that the error flag and comparison bits are computed for each group that makes up the codeword. Finally, for *S2*, the error indication bit is obtained from the logical expression

$$e'_i = f_i \cdot (s_i + c_i). \tag{8}$$

It is worth noting that the prime symbol (′) is used to differentiate (8) from (6).

Given the previous expressions and descriptions, the output word from the receiver assumes the format shown in Fig. 4. A logical value of 1 in any error indication bits means an error in one or more information bits in the slice associated with that indication bit. That is how an error data can be detected.

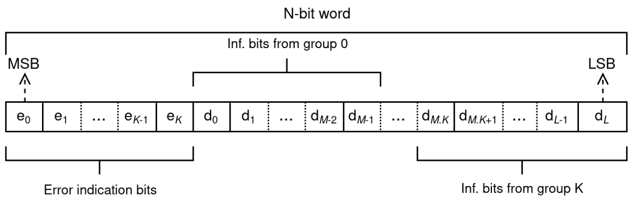

FIGURE 4. Receiver's output word format.

The codeword formation rule presented in this section was designed to maintain compatibility with the data bus of already consolidated on-chip communication protocols, which generally have a width defined by a power of 2 (e.g., 8, 16, 32, 64). However, another formation rule could be considered by doing, for example, $M' = N'_d/N'_p$ and $N' = N'_d + N'_p$, where N'_d and N'_p are powers of 2 with $N'_p \leq N'_d$.

IV. SOFTWARE SIMULATION

A. SIMULATION ENVIRONMENT

The proposed approach was verified by comparing the techniques of the TMR, PDMR (parity-based DMR without comparator, *S1*), and PDMR-C (parity-based DMR with comparator, *S2*). Tests were carried out via software using Python programming language. For this, some scripts were created with different functionalities, such as:

- *Data generation*: It generates the data (words) for transmission. As input arguments, the script receives the amount of data to be generated and the bit width (N_d) of each data. As output, the script creates a *CSV* file, where each line represents a single data with N_d information bits, and the number of rows corresponds to the amount of data (T_D);

- *Fault generation*: The script was designed to generate fault vectors by receiving the amount of data and the bit width (in this case, defined by N) of each data as input arguments. In addition, the script gets as input the error rate to be used for the fault vectors. This error rate is applied bit by bit and given in terms of $errors/(bit.day)$, representing a typical measurement for SEUs [35], [36]. Therefore, the random event of a bitflip occurring is simulated, for simplicity, through a uniform distribution based on the error rate. A random number is generated, and a bitflip occurs if its value is less than the rate value. As output, the script generates a *CSV* file with three columns, each containing an N-bit fault vector. Each of the three columns must feed a channel for the TMR test, while only two must be used for the PDMR and PDMR-C tests. The number of rows also corresponds to the amount of data (T_D);
- *Simulation control and data analysis*: It must read the files generated by the two scripts with the described functionalities. Among the inputs received by the script are the bus data width (N), the number of information bits (N_d), and an argument that indicates the group width, which follows the diagram shown in Fig. 2. Also, the script generates a *CSV* output file, which contains the originally transmitted data, the data after passing through each channel, and the data received by the receiver after applying the TMR, PDMR, and PDMR-C tolerance methods. This file also contains the counting values of bit errors for data analysis.

In Fig. 5, a simulation diagram is presented. Initially, the *CSV* files generated by the data and fault generation scripts are read, and each row is analyzed. The transmitter takes care of doubling or tripling the channels and inserting the parity for the PDMR and PDMR-C cases. In the next step, faults are injected into the channels through a function that takes the channel data as an argument and performs an *XOR* operation with one of the columns (according to the channel) of the file that contains fault vectors. In tests with DMR, only two channels are used, so only columns 0 and 1 of the file are considered, whereas, for TMR, columns 0, 1, and 2 are used. At the receiver, these data are treated to mitigate errors so that, for the TMR, the classical voter logic (1) is used, while for the PDMR and PDMR-C, the logics described in Section III are used.

B. RESULTS AND DISCUSSION

During the simulations, two basic configurations were implemented:

- *C1*: $N_d = 24$ and $N_p = 8$ corresponds to 24 information bits and eight parity bits, referred to as PDMR 24 × 8;
- *C2*: $N_d = 24$ and $N_p = 8$ corresponds to 24 information bits and eight parity bits with comparator (defined by (7)), referred to as PDMR-C 24 × 8.

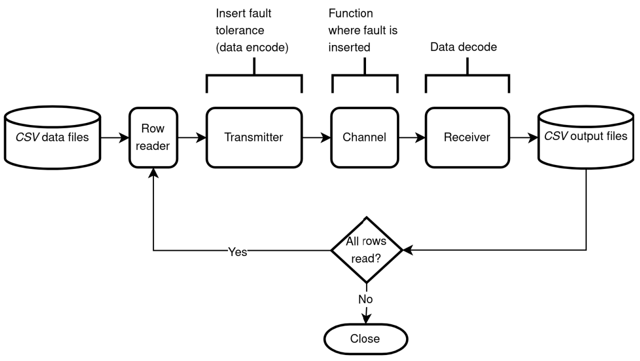

FIGURE 5. Simulation diagram.

Note that both configurations yield a 32-bit bus for DMR tests. In the case of TMR, as no extra bits are added to the information bits, there are only 24 information bits. Therefore, for DMR cases, each channel's 32 bits (parity + information) are subject to bitflip, while for TMR, only the information bits (24) of each channel are considered susceptible. Additionally, the amount of data (T_D) was taken to be 10^6 messages per day.

Table 2 contains the simulation results for *C1* and *C2*. These results were obtained considering the average of 100 simulation runs. In other words, a hundred fault vector files, with a total of T_D lines, were generated for each error rate, and each file was applied to the data set. In this table, P_D represents the percentage of detected errors, while P_U and P_E represent the percentages of undetected and total errors, respectively. Note that the value of P_E is related to the total number of messages transmitted T_D. At the same time, P_D and P_U are percentages associated with the number of erroneous messages ($E = T_D \times P_E$).

TABLE 2. Simulation results.

E.R. (errors/(bit.day))	Method	(%) P_D	P_U	P_E
2×10^{-2}	TMR-24	0	100	2.81
	C1	85.10	14.90	7.14
	C2	96.74	3.26	7.93
10^{-2}	TMR-24	0	100	0.71
	C1	86.40	13.60	1.90
	C2	98.33	1.67	2.13
5×10^{-3}	TMR-24	0	100	0.18
	C1	87.18	12.82	0.49
	C2	99.14	0.86	0.55
2×10^{-3}	TMR-24	0	100	0.03
	C1	87.66	12.34	0.08
	C2	99.66	0.34	0.09
10^{-3}	TMR-24	0	100	0.01
	C1	88.18	11.82	0.02
	C2	99.83	0.17	0.02

Considering the results of Table 2, one can notice that values tend to get closer as rates decrease. Moreover, the TMR classical logic does not have error detection capability, and the voting scheme is responsible for mitigating the errors. The results for *C2* suggest that adding a comparator to the logical design significantly impacts the percentages of detected errors, while the percentages P_E for *C1* and *C2* are kept close. So, it can be concluded that despite the comparator increasing the logical elements for hardware synthesis, its capacity of decreasing P_U and increasing P_D is advantageous depending on the application and might be explored in future works.

In order to evaluate the percentage results, taking into account the possibility of retransmission of messages detected with error, the PDMR *C1* and *C2* implementations were taken for a more detailed analysis. The percentages presented in Table 2 were used for retransmission analysis to obtain the total number of messages with errors accumulated in each transmission. For this, the equation $A_E = T \times P_E + U_T$ was used, where A_E is the total accumulated errors and represents the total number of erroneous messages in the current transmission plus the number of undetected incorrect messages accumulated throughout transmissions; T is the number of messages transmitted; P_E is the percentage of errors and U_T corresponds to the total number of erroneous messages not detected throughout the transmissions. In the first transmission, T equals to the total number of messages transmitted (e.g., $T = T_D = 10^6$) and $U_T = 0$. After the first transmission, $T = D'$, where $D' = E' \times P_D$ is the number of erroneous messages detected, obtained from the total number of messages with errors in the previous transmission ($E' = T' \times P_E$, where T' is the number of messages transmitted in the previous transmission). In this case, $U_T = E' \times P_U + U'_T$, where U'_T is the value of U_T obtained in the previous transmission. The results of this evaluation are in Table 3, where the columns contain the $\%A_E = A_E * 100/T_D$ values.

TABLE 3. Retransmission evalution.

Method	Transmission	Error rate (errors/(bit.day))				
		2×10^{-2}	10^{-2}	5×10^{-3}	2×10^{-3}	10^{-3}
TMR-24	1st	2.81	0.71	0.18	0.03	0.01
C1	1st	7.14	1.90	0,49	0,08	0.02
	2nd	1.50	0.29	0.06	0.01	0
C2	1st	7,93	2.13	0.55	0.09	0.02
	2nd	0.87	0.08	0.01	0	0

By analyzing Table 3, it is clear that for both *C1* and *C2*, one retransmission is enough so that the accumulated error percentages are lower than a single transmission of the TMR-24. A retransmission for the TMR is not considered once it has no detection capability, so one should have to retransmit all data. When analyzing numerically, a single transmission of TMR-24 implies 72×10^6 bits transmitted per day (3 channels $\times$ 24 bits per message per channel $\times$ 10^6 messages per day). Meanwhile, the PDMR 24 $\times$ 8 with rate 2×10^{-2} yields approximately 67.886×10^6 bits transmitted per day (2 channels $\times$ 32 bits per message per channel $\times(10^6$ messages per day $+T_R)$, where $T_R = P_E \times P_D \times 10^6$ messages per day is the total of retransmitted messages). Following the same reasoning, for PDMR-C 24 $\times$ 8 with a 2×10^{-2} rate, we have approximately 68.910×10^6 bits transmitted per day, considering the two channels and the retransmission. The difference between the values is even more significant as the rates decrease since the PDMR implementations require fewer bits to be retransmitted for lower rates. These results demonstrate that TMR implies a greater occupancy of digital transmission channels than the proposed approach.

V. HARDWARE IMPLEMENTATION

After software simulations, TMR-24, *C1*, and *C2* configurations were implemented using hardware description language (HDL) and incorporated into the AMBA AXI communication interfaces. Thus, we seek to compare the techniques regarding area and power consumption.

The AMBA standard allows the interconnection of blocks in systems-on-chip and includes specifications for AXI interfaces, which support high-frequency and high-performance communication [12]. According to [37], the fourth version of the AXI protocol, AXI4, has three interfaces: AXI4, AXI4-Lite, and AXI4-Stream. The AXI4 interface can be used for high-performance memory mapping requirements. AXI4-Lite is suitable for low throughput memory mapping communication. Finally, AXI4-Stream is useful for high-speed streaming data.

The AXI4 specification defines five independent channels: Read Address, Read Data, Write Address, Write Data, and Write Response [30]. It must be noted that each independent channel comprises a set of information signals and signals that control the transfer flow (handshake mechanism) [30].

The protection of control and handshake signals can be performed via parity bits [17], simplifying the logic implementation but requiring extra bus lines. Another option, since we consider two redundant communication interfaces in PDMR, is to allow transmission only when both interfaces' control signals have equal logical values required to perform the transfers. This way, a transient event on the control line of one of the channels does not compromise data transmission. As the objectives of the hardware implementation were to validate the proposed approach and compare it with TMR, for simplicity, we compared the techniques by applying them only to the data bus of the Read Data channel, which was considered the data transmission path to the CPU (Central Processing Unit).

A. HARDWARE PLATFORM AND SYSTEM DESIGNS

The designs were implemented and tested on the Zedboard development board, composed of the Xilinx Zynq-7000 XC7Z020 SoC. This chip integrates an ARM processing system (CPU) and reconfigurable logic (FPGA). Furthermore, *Vivado 2022.1* software was used for the FPGA synthesis and implementation process. Although the tests have been performed on a Zynq platform, as long as the expressions in section III are implemented in hardware, there is no restriction on using the proposed approach on other platforms.

During the development of the hardware designs, the *Vivado* tool for creating and packaging custom IP Cores was used to develop the blocks for the TMR and PDMR configurations [38]. This tool builds IPs with AXI4 interfaces

(Fig. 6) and generates Verilog codes for these interfaces. Thus, codes can be modified according to the application. In this paper, only small changes were performed, maintaining the generated interfaces' operating characteristics.

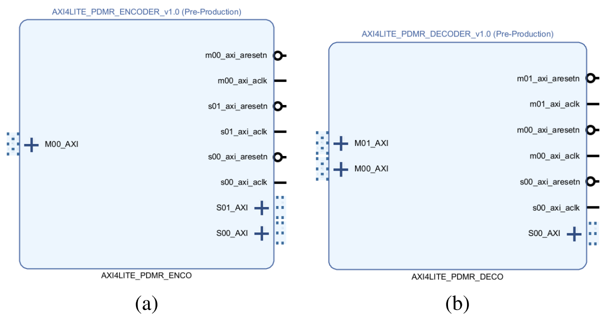

FIGURE 6. Generated IPs. (a) Encoder. (b) Decoder.

In the designs for this work, two IPs called encoder and decoder with AXI4-Lite interfaces were generated for each configuration. The encoder incorporates the function previously presented for the transmitter, while the decoder works as the receiver. In the case of PDMR and PDMR-C, the encoders are the same, and they add the parity bits given by the logic in (3) to the data read through the master interface. The resulting codeword is then duplicated and made available to the slave interfaces. On the other hand, the TMR encoder only triplicates the data coming from the master interface.

The PDMR decoder IP receives data through the master interfaces, decides the output word based on the expressions in Section III-B, and keeps the result in a register that the CPU can read through the slave interface. The decoder for PDMR-C presents the same behavior but incorporates the comparator and the logic in (8) instead of (6). The TMR decoder presents similar behavior, but the output word is obtained using voters based on (1).

Based on the developed encoders and decoders, different systems were designed to compare the occupied area and power consumption for TMR and PDMR hardware implementations. In Fig. 7, the block design for the most basic system is presented, in which one can observe a communication path between the processing system and peripherals (represented as *axi_data_reader*). In this case, with the aid of *AXI Interconnect*, 24-bit data is transferred from the CPU to the *axi_data_source* block, which in turn sends it to the *axi_data_reader* block. These *AXI GPIO* blocks are part of Xilinx's IP Repository and were used to make test development more straightforward and faster. Subsequently, the reading is carried out using the decoder. Since AXI4-Lite allows memory mapping communication, the CPU can access the addresses of the *S_AXI* interfaces of the *axi_data_source* block and *S00_AXI* of the decoder block. Code was developed in the C programming language and incorporated into the Zynq processor for this test. The encoder and decoder for PDMR were marked in red to indicate that they are the developed blocks. Meanwhile, the other blocks, in blue, represent the components that emulate the data transmission structure in the test, such as the CPU and peripherals.

In order to verify the functioning of the system in the face of data faults in the communication channels, the block design in Fig. 8 was constructed. In this figure, some blocks were marked in red to indicate that they are the developed PDMR blocks, while the others, in blue, were used to emulate the fault injection and the data transmission structure, such as the CPU and peripherals. The *fault_injector_ch0* and *fault_injector_ch1* blocks were used to insert faults in the data bus of the Read Data channels between the encoder and decoder blocks. The CPU sent the 32-bit fault vectors to be inserted into the channels to the blocks *fault_vec_data_ch0* (green line) and *fault_vec_data_ch1* (blue line). Furthermore, the 24-bit data to be read through the decoder was also passed to the *axi_gpio_data_source* block, which transmitted it to the *axi_gpio_data_reader* block (red line). As described for the system in Fig. 7, the routine for transferring data and faults and reading data via decoder was carried out using an embedded C program in the Zynq processor.

An example of the data flow in the system in Fig. 8 is illustrated in Fig. 9. Initially, the data $0 \times 8a1b76$, represented in hexadecimal, was transferred by the Zynq processing system to *axi_gpio_data_source*, which passed it to *axi _gpio_data_reader*. The fault vectors 0×00100000 and 0×00104004 were stored in the blocks *fault_vec_data_ch0* and *fault_vec_data_ch1* through the CPU. Then, the information was read from *axi_gpio_data_reader*, and parity bits were added in the encoder block following the codeword formation rule described in Section III. When passing through the *fault_injector_ch0* and *fault_injector_ch1* blocks, the duplicate codewords were corrupted using a *XOR* bitwise operation with the respective fault vectors. Corrupted data from *fault_injector_ch0* and *fault_injector_ch1* blocks were treated in the decoder to mitigate faults. It can be observed that, at the end of the process, the value read by the CPU was $0 \times 208a9b6$ and that the two most significant hexadecimal digits, underlined in Fig. 9, correspond to the error indication bits $0b00100000$. These bits indicate an error in one of the information bits in group 2, and the hexadecimal 0×9, also underlined, highlights the error in the message.

The last block design implemented is shown in Fig. 10. In this system, we seek to add fault tolerance to AXI Interconnect since, as described in [12] and [31], it can be a source of errors in architectures that use hardware acceleration. The behavior of this system is similar to that described in Fig. 7. However, we consider that faults can occur in the paths between the master and slave interfaces of the AXI Interconnect and the encoder, as well as between the master and slave interfaces of the AXI Interconnect and the decoder.

Systems such as those in Fig. 7 and Fig. 10 were also developed for TMR tests. The characteristics and modifications of the communication interfaces generated by *Vivado* were kept the same for comparison purposes. The

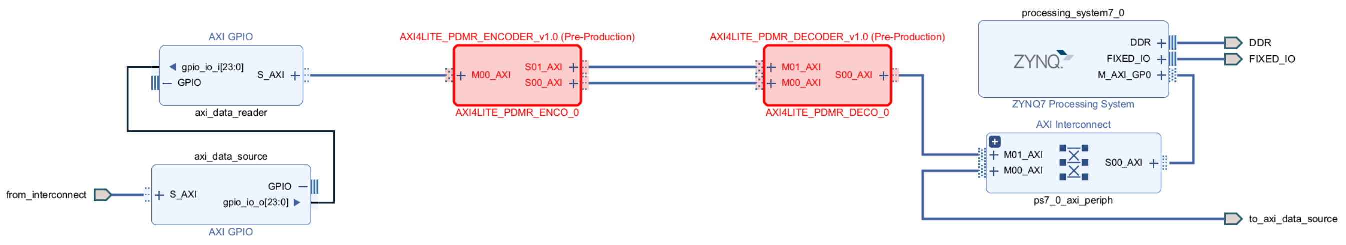

FIGURE 7. System design for emulating data transmission with PDMR implementations (blocks marked in red).

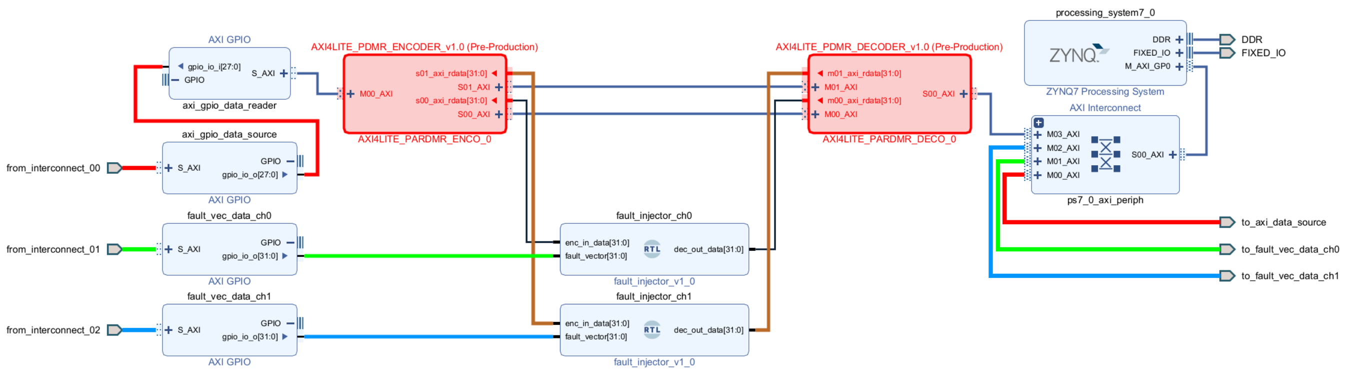

FIGURE 8. Block design for fault injection emulation using the developed PDMR blocks (marked in red).

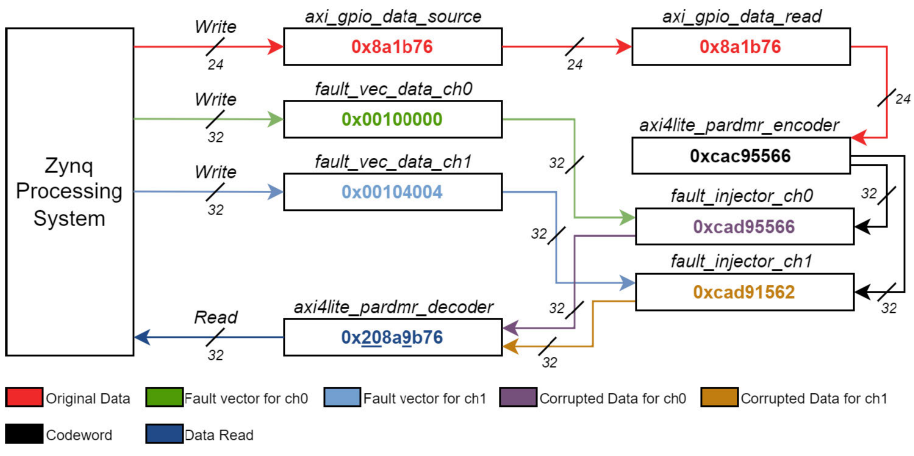

FIGURE 9. Data flow example for fault injection test.

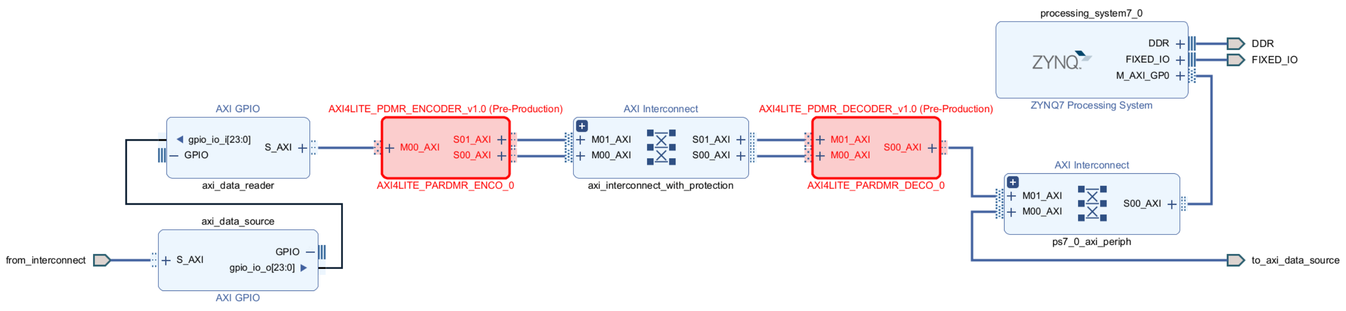

FIGURE 10. The emulation test environment includes redundancy in the interconnect with PDMR blocks (in red).

differences concern adding an extra master interface to the TMR decoder IP and another slave interface to its encoder IP. These IPs replaced the PDMR blocks (marked in red in the figures), and the *axi_interconnect_with_protection* (Fig. 10) was configured with three master interfaces and three slave interfaces.

B. AREA AND POWER CONSUMPTION RESULTS

Table 4 contains the resource utilization and power estimation results obtained from the Vivado post-implementation. This table displays the values of Slice LUTs, Slice Registers, and Power in milliwatts (mW). The columns % correspond to the reduction relationship between the values of the PDMR configurations relative to the TMR, obtained by

$$V_{\%} = 100 \times (1 - V_{PDMR}/V_{TMR}), \tag{9}$$

where $V_{\%}$ represents the percentage value, V_{PDMR} is the value for the *C1* or *C2* configuration, and V_{TMR} is the value for TMR. Note that the values in the % columns indicate the percentage of the *C1* configuration in relation to the TMR and the *C2* configuration in relation to the TMR. Additionally, negative values indicate an increase in the utilization of resources for *C1* or *C2* relative to the TMR implementation.

TABLE 4. Resource utilization and power estimation.

Module	Method	Resource				Power	
		Slice LUTs	%	Slice Registers	%	mW	%
Encoder	TMR-24	202	–	443	–	2	–
	C1 and *C2*	174	13.86	383	13.54	2	0
Decoder	TMR-24	162	–	349	–	2	–
	C1	161	0.62	296	15.19	1	50
	C2	181	-11.73	296	15.19	2	0
Total [a]	TMR-24	364	–	792	–	4	–
	C1	335	7.97	679	14.27	3	25
	C2	355	2.47	679	14.27	4	0

[a]Total equals the sum of Encoder and Decoder values.

Table 4 shows that the encoder used for *C1* and *C2* presents a reduction of 13.86% in Slice LUTs and 13.54% in Slice Registers compared to TMR. Meanwhile, no reduction was observed regarding power consumption. Considering the values for the decoder, there was a decrease of 0.62% for *C1* and an increase of 11.73% for *C2* in terms of Slice LUTs. This increase in *C2* can be explained by the insertion of the comparator logic. On the other hand, in both configurations, there was a reduction of 15.19% in Slices Registers compared to the TMR. The Power values show that, in *C1*, there is a consumption 50% lower than that for the TMR decoder and that the consumption remained the same for *C2*.

Given the total values as the sum of the encoder and decoder numbers, we see 7.97% less Slice LUTs for the *C1* configuration and 2.47% less for *C2* when compared to the total for the TMR. In the case of Slice Registers, both configurations present a decrease of 14.27% in relation to the TMR implementation. There was no difference in power consumption between *C2* and the TMR, while *C1* had an estimated power consumption reduction of 25%.

The resource utilization and power consumption results for the test presented in Fig. 10 can be found in Table 5. In this system implementation, the *AXI Interconnect* (named *axi_interconnect_with_protection*) was considered part of the data transmission path. This way, its Slice LUTs, Slice Registers, and Power values were included in the total sum. The values in Table 5 show that the percentages of Slice Registers for the encoder and decoder and their sum remained the same as in Table 4. However, there was an increase of 9.26% in Slice Registers for the Interconnect for both *C1* and *C2* configurations. Taking the total as the sum of the encoder, decoder, and Interconnect values, there was a decrease of 11.44% in Slice Registers. Concerning Slice LUTs, we see percentages close to those in Table 4, and we get a more significant reduction when considering the total values. Also, estimated power consumption was reduced for both *C1* and *C2* configurations, reaching a total value of 20%.

TABLE 5. Resource utilization and power estimation with interconnect.

Module	Method	Resource				Power	
		Slice LUTs	%	Slice Registers	%	mW	%
Encoder	TMR-24	214	–	443	–	2	–
	C1 and *C2*	174	18.69	383	13.54	2	0
Decoder	TMR-24	161	–	349	–	2	–
	C1	161	0	296	15.19	1	50
	C2	182	-13.04	296	15.19	1	50
Interconnect	TMR-24	165	–	108	–	1	–
	C1 and *C2*	131	20.61	118	-9.26	1	0
Encoder + Decoder	TMR-24	375	–	792	–	4	–
	C1	335	10.67	679	14.27	3	25
	C2	356	5.07	679	14.27	3	25
Total [a]	TMR-24	540	–	900	–	5	–
	C1	466	13.7	797	11.44	4	20
	C2	487	9.81	797	11.44	4	20

[a]Total equals the sum of Encoder, Decoder, and Interconnect values.

Facing the results presented in this subsection, we highlight that the AXI interfaces generated by Vivado are composed of several signals from the AXI specification whose behaviors are implemented by the tool, mainly with registers. In other words, the basic interfaces, without applying a tolerance method, have more registers than LUTs, which may explain the more significant impact of the proposed approach on Slice Registers. Meanwhile, the combinational logics of PDMR and PDMR-C are more complex than the classical voter (1), justifying a greater utilization of LUTs, even though the TMR decoder block has an additional master interface. However, reaching lower total values for *C1* and *C2* configurations was still possible. Considering power consumption and lower resource utilization for the decoder, the *C1* configuration would be more appropriate, although it represents a reduction in detection capability. Furthermore, from Table 5, we conclude that including the proposed method in conjunction with more elaborate structures, such as the Interconnect, may yield a more significant reduction in the use of resources and power consumption.

It is worth noting that the proposed technique was compared with TMR on basic interfaces to check the occupied area and power consumption at a more fundamental application level. However, redundancy techniques generally involve replicating hardware acceleration components with more robust tasks, which occupy a considerable part of the FPGA's reconfigurable logic. For example, a hardware accelerator for image compression can be tripled with TMR, and the voting circuit can vote on the data resulting from the operations. In the case of PDMR, there would be a need

TABLE 6. Characteristics of different methods, adapted from [17].

Property/Characteristic	Method		
	PDMR	TMR	FEC
Concurrent glitches in multiple wires can be undetected	Partially	Partially	Yes
Impacts bus clock frequency	Partially	Partially	Partially
Increases area for the interconnection	Yes	Yes	Partially
Requires changes to the manager (master)	Partially	Partially	Yes
Requires changes to the subordinates (slaves)	Partially	No	Yes
Implementation is straightfoward	Yes	Yes	No
Some transactions may need several transfers	Yes	No	Yes

to duplicate the application, which would already represent a reduction in the occupied area, and the addition of the combinational logic presented for the encoder (codeword formation), which can be embedded in the communication interface of each redundant element.

Hardware implementations may depend on the technology and communication protocols chosen for the designs. From this perspective, as stated by Lázaro et al. [32], other approaches are specific to other interface definitions, which makes comparisons difficult. The TMR-based approach described by [32] differs from this paper, mainly in the following aspects: the authors performed modifications to the five AXI channels to implement the redundancy mechanism; a single module was developed, which would be equivalent to our decoder implementations. Furthermore, the authors focus on the minimization of FPGA resources. At the same time, our hardware implementations were developed to verify the viability of the proposed approach with respect to TMR, considering minimal changes to the structure of the interfaces generated by *Vivado*. Comparison of area occupancy becomes difficult due to the specific nature and modifications of the IP, as well as differences in the focus of the approaches. Therefore, a fair comparison would involve a more detailed change of our modules' code to minimize resource utilization and the integration of the technique to the other AXI independent channels.

VI. FURTHER COMPARISONS

Despite comparison with FEC codes not being among the objectives of our work, since we consider the proposed approach closer to methods that involve fault-masking or the possibility of retransmission, we leverage the analysis carried out by Mach et al. [17] to summarize the properties of PDMR, TMR, and FECs. This summary is presented in Table 6. Notably, the authors performed the analysis based on information redundancy like parity and SECDED. As FECs are part of this type of redundancy, we adapted the table to designate the characteristics of FECs.

The first property in Table 6 is related to transient events in multiple bus wires. In the case of FEC schemes, such as SEC and SECDED, as the position in which an error occurred is identified through the syndrome, multiple concurrent glitches may end up undetected by the decoder logic, for example, when transient faults occur in several wires protected by the same parity bit. For TMR, the effect is partial because, once the channel is replicated, the correction does not happen if the glitches hit the same bus line on two different channels. So, if a channel suffers multiple glitches, the majority voter guarantees the correction based on the bits from the other channels. The logic of PDMR has a similar behavior to that of TMR in the sense of channel replication. However, dividing the codeword into groups also means that the occurrence of glitches in different groups can be masked or detected.

All techniques can impact the clock frequency, mainly due to the encoders and decoders added to the channels for PDMR and FEC. In the case of TMR, the impact comes from the voting circuit. Moreover, TMR and PDMR have a greater effect on increasing the area for interconnection (communication interfaces) when compared to simple FEC codes. However, PDMR has a reduced impact in relation to TMR, which can be even smaller if we consider the replication of hardware acceleration components, as discussed in Section V.

At this point, the manager is considered to be the component responsible for initiating a data transfer, such as the processor. Meanwhile, subordinates are the components that receive or send information to the manager, such as peripherals. Regarding the execution of changes in the manager (master) and subordinates (slaves), the information redundancy (FEC codes) requires that they be carried out because it is necessary to implement the encoder logic in the manager and the decoder in the subordinates. Depending on the width of the input data, FEC codes may require extra bus lines so that data and redundant bits can be transferred. These lines must be included in the design of the communication interfaces. Conversely, TMR requires a partial change in the manager if the majority voter is used alongside it to vote data from replicated subordinates. In this case, there is no need to modify the subordinates. In PDMR, changes to the manager and subordinates must be executed to include the coding and decoding logic. However, the inclusion of extra bus lines is discarded depending on the input data width.

The PDMR encoder and decoder mainly involve the use of *XOR* operations to calculate the parity and comparator bit (Section III), if used, and *MUX* to select the channel, which is given by (4). On the other hand, although the SEC and SECDED encoder is relatively simple (it uses *XOR* logical operations), its decoder can be composed of *XOR* operations and flip-flops to store results [16], which makes its implementation more complex. Considering the differences in complexity, we classify FEC codes as not straightforward and PDMR and TMR as straightforward. Based on this, we also classify the change in the manager and subordinates as *Partially* for PDMR.

Finally, PDMR, like FEC codes, may require multiple transfers for a transaction. For example, when the processor (system) can only send a limited number of bits so that, for a single transaction, several transfers are necessary for data and redundant bits to be sent. Meanwhile, the transaction can occur with a single transfer for TMR, as no extra bits are added to the data.

It is important to note that TMR presents better characteristics in the comparison of Table 6. However, its high area and power overhead can be a problem in applications with design constraints, such as onboard systems in nanosatellites. As demonstrated, PDMR has lower power consumption and a smaller area, which can be even smaller if the detection/comparison logic is disregarded, which can be interesting in applications where the error rate is very small. Detectability is also an attractive property of PDMR over TMR. This property allows possible errors to be indicated in the groups of the transmitted data. So, retransmission actions can be carried out by hardware or software implementations. Moreover, it should be noted that dividing the data into groups allows implementations in which only groups with an error indication are retransmitted, the impact of which will be explored in future work. Regarding FEC codes, PDMR has an advantage concerning ease of implementation and the possibility of masking faults on multiple wires, while other characteristics may be application-dependent.

VII. CONCLUSION

This paper presents a parity-based DMR approach for application to COTS SoCs to be used for onboard processing in nanosatellites. The objective of the approach is to increase the reliability of data transmission between the hardcore processing unit and the application located in the reconfigurable logic area (FPGA).

Initially, the experiments on the proposed approach were carried out through simulations developed in Python scripts. The results show that the percentage of total errors for the parity-based DMR approach configurations (PDMR and PDMR-C) get close to TMR as the SEU rates decrease. In addition, when considering the possibility of detecting errors in the proposed approaches, reducing the number of accumulated errors through retransmissions is possible while still keeping the total number of transmitted bits lower than TMR.

After the software simulations, hardware implementations were carried out with the aid of the Xilinx Zynq-7000 SoC to compare with the TMR method in terms of hardware resource utilization and power consumption. The tests were based on applying the proposed method on the Read Data channel data bus of simple AXI-Lite interfaces. In this context, the proposed approach showed a reduction in the utilization of Slice LUTs and Slice Registers when considering the total values of the implementations. Furthermore, it was possible to observe a decrease in power consumption.

The decrease in hardware resource utilization values may have a more significant impact on more complex communication structures. Therefore, in future work, we will analyze the integration of the technique in AXI and AXI-Stream interfaces and redundant hardware acceleration applications that incorporate communication interfaces. We also plan to explore the optimization of the described hardware designs and implementation of the retransmission structure in software and hardware. Additionally, we will seek to analyze the correction and detection capability of the proposed method, considering other configurations of parity bits, information bits, and codeword formation rules that make the technique more transparent for communication between processor and hardware acceleration.

REFERENCES

[1] K. Woellert, P. Ehrenfreund, A. J. Ricco, and H. Hertzfeld, "Cubesats: Cost-effective science and technology platforms for emerging and developing nations," *Adv. Space Res.*, vol. 47, no. 4, pp. 663–684, Feb. 2011.

[2] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *J. Aerosp. Inf. Syst.*, vol. 15, no. 4, pp. 178–192, Apr. 2018.

[3] M. Pignol, "COTS-based applications in space avionics," in *Proc. Design, Autom. Test Eur. Conf. Exhibition (DATE)*, Mar. 2010, pp. 1213–1219.

[4] K. A. LaBel, M. M. Gates, A. K. Moran, P. W. Marshall, J. Barth, E. G. Stassinopoulos, C. M. Seidleck, and C. J. Dale, "Commercial microelectronics technologies for applications in the satellite radiation environment," in *IEEE Aerosp. Appl. Conf. Proc.*, Sep. 1996, pp. 375–390.

[5] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth, "Harsh environments: Space radiation," *Johns Hopkins APL Tech. Dig.*, vol. 28, no. 1, p. 17, Jul. 2008.

[6] C. De Sio, S. Azimi, L. Sterpone, and B. Du, "Analyzing radiation-induced transient errors on SRAM-based FPGAs by propagation of broadening effect," *IEEE Access*, vol. 7, pp. 140182–140189, 2019.

[7] A. D. George and C. M. Wilson, "Onboard processing with hybrid and reconfigurable computing on small satellites," *Proc. IEEE*, vol. 106, no. 3, pp. 458–470, Mar. 2018.

[8] F. L. Kastensmidt and R. Reis, "Fault tolerance in programmable circuits," in *Radiation Effects on Embedded Systems*. Cham, Switzerland: Springer, 2007, pp. 161–181.

[9] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," *ACM Comput. Surveys*, vol. 47, no. 2, pp. 1–34, Jan. 2015.

[10] S. Kasap, E. Weber Wächter, X. Zhai, S. Ehsan, and K. Mcdonald-Maier, "Survey of soft error mitigation techniques applied to LEON3 soft processors on SRAM-based FPGAs," *IEEE Access*, vol. 8, pp. 28646–28658, 2020.

[11] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond," *Proc. IEEE*, vol. 103, no. 3, pp. 379–389, Mar. 2015.

[12] C. De Sio, S. Azimi, and L. Sterpone, "On the evaluation of SEU effects on AXI interconnect within AP-SoCs," in *Architecture of Computing Systems—ARCS*. Cham, Switzerland: Springer, 2020, pp. 215–227.

[13] F. Benevenuti and F. L. Kastensmidt, "Reliability evaluation on interfacing with AXI and AXI-S on Xilinx Zynq-7000 AP-SoC," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, Mar. 2018, pp. 1–6.

[14] P. Faraj, J. Leibrich, and W. Rosenkranz, "Coding gain of basic FEC block-codes in the presence of ASE noise," in *Proc. 5th Int. Conf. Transparent Opt. Netw.*, 2003, pp. 80–83.

[15] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[16] C. Desset and A. Fort, "Selection of channel coding for low-power wireless systems," in *Proc. 57th IEEE Semiannual Veh. Technol. Conf.*, Oct. 2003, pp. 1920–1924.

[17] J. Mach, L. Kohutka, and P. Cicák, "On-chip bus protection against soft errors," *Electronics*, vol. 12, no. 22, p. 4706, Nov. 2023.

[18] ARM. (2011). *Cortex$_{TM}$-R5 Technical Reference Manua*. Accessed: Jun. 2, 2024. [Online]. Available: https://developer.arm.com/documentation/ddi0460/latest/
[19] W. Shiroma, L. Martin, J. Akagi, J. Akagi, B. Wolfe, B. Fewell, and A. Ohta, "CubeSats: A bright future for nanosatellites," *Open Eng.*, vol. 1, no. 1, pp. 9–15, Jan. 2011.
[20] T. Villela, C. A. Costa, A. M. Brandão, F. T. Bueno, and R. Leonardi, "Towards the thousandth CubeSat: A statistical overview," *Int. J. Aerosp. Eng.*, vol. 2019, pp. 1–13, Jan. 2019.
[21] J. Gracia-Morán, L. J. Saiz-Adalid, D. Gil-Tomás, and P. J. Gil-Vicente, "Improving error correction codes for multiple-cell upsets in space applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 10, pp. 2132–2142, Oct. 2018.
[22] V. Raghunathan, M. B. Srivastava, and R. K. Gupta, "A survey of techniques for energy efficient on-chip communication," in *Proc. Design Autom. Conf.*, Apr. 2003, pp. 900–905.
[23] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: The energy-reliability tradeoff," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 818–831, Jun. 2005.
[24] K. S. Morgan, D. L. Mcmurtrey, B. H. Pratt, and M. J. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2065–2072, Dec. 2007.
[25] M. L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance,Analysis,and Design*. Hoboken, NJ, USA: Wiley, 2002.
[26] P. Balasubramanian, K. Prasad, and N. E. Mastorakis, "A fault tolerance improved majority voter for TMR system architectures," *WSEAS Trans. Circuits Syst.*, vol. 15, pp. 108–122, Oct. 2016.
[27] I. A. Gomes, M. G. Martins, A. I. Reis, and F. L. Kastensmidt, "Exploring the use of approximate TMR to mask transient faults in logic with low area overhead," in *Proc. Microelectron. Rel.*, 2015, vol. 55, no. 9, pp. 2072–2076.
[28] A. J. Sánchez-Clemente, L. Entrena, and M. García-Valderas, "Partial TMR in FPGAs using approximate logic circuits," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 4, pp. 2233–2240, Aug. 2016.
[29] T. Arifeen, A. S. Hassan, and J.-A. Lee, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139851–139867, 2020.
[30] ARM. (2013). *Amba AXI and Ace Protocol Specification AXI3, AXI4, and AXI4-lite ACE and ACE-Lite*. Accessed: Mar. 19, 2024. [Online]. Available: https://developer.arm.com/documentation/ihi0022/e
[31] C. De Sio, S. Azimi, and L. Sterpone, "On the analysis of radiation-induced failures in the AXI interconnect module," *Microelectron. Rel.*, vol. 114, Nov. 2020, Art. no. 113733.
[32] J. Lázaro, A. Astarloa, A. Zuloaga, J. Á. Araujo, and J. Jiménez, "AXI lite redundant on-chip bus interconnect for high reliability systems," *IEEE Trans. Rel.*, vol. 73, no. 1, pp. 1–6, Oct. 2024.
[33] J. Borecký, M. Kohlík, P. Vít, and H. Kubátová, "Enhanced duplication method with TMR-like masking abilities," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2016, pp. 690–693.
[34] L. T. Clark, D. W. Patterson, N. D. Hindman, K. E. Holbert, S. Maurya, and S. M. Guertin, "A dual mode redundant approach for microprocessor soft error hardness," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 6, pp. 3018–3025, Dec. 2011.
[35] G. Furano and A. Menicucci, "Roadmap for on-board processing and data handling systems in space," in *Dependable Multicore Architectures Nanoscale*. Cham, Switzerland: Springer, 2018, pp. 253–281.
[36] C. Argyrides, H. R. Zarandi, and D. K. Pradhan, "Matrix codes: Multiple bit upsets tolerant method for SRAM memories," in *Proc. 22nd IEEE Int. Symp. Defect Fault-Tolerance VLSI Syst. (DFT)*, Sep. 2007, pp. 340–348.
[37] Xilinx. (2017). *Vivado Design Suite: AXI Reference Guide*. Accessed: Mar. 19, 2024. [Online]. Available: https://docs.amd.com/v/u/en-U.S./ug1037-vivado-axi-reference-guide
[38] AMD-Xilinx Inc. (2023). *Vivado Design Suite User Guide: Creating and Packaging Custom IP*. Accessed: Mar. 20, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug1118-vivado-creating-packaging-custom-ip/Creating-and-Packaging-Custom-IP

ALEX C. R. ALVES received the B.S. degree in electrical engineering from the Federal University of Campina Grande (UFCG), Brazil, in 2016, and the M.S. degree in mechatronics engineering from the Federal University of Rio Grande do Norte (UFRN), Brazil, in 2019, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. Since 2022, he has been a Professor with the Federal Institute of Education, Science, and Technology of Ceará (IFCE). His research interests include embedded systems, onboard processing, and FPGAs.

LUIZ F. Q. SILVEIRA (Member, IEEE) received the B.S. degree in electrical engineering from the Federal University of Paraíba (UFPB), Brazil, in 2000, and the M.Sc. and Ph.D. degrees in electrical engineering from the Federal University of Campina Grande (UFCG), Brazil, in 2002 and 2006, respectively. He was a Visiting Professor with the University of Toronto (UofT), Canada, from September 2021 to August 2022. He is currently an Associate Professor with the Department of Computing Engineering and Automation, Federal University of Rio Grande do Norte (UFRN), Brazil. His research interests include wireless communications, information theoretic learning, channel coding, and energy saving in data processing.

MÁRCIO E. KREUTZ received the B.S. degree in computer science and the M.Sc. and Ph.D. degrees in computer science and microelectronics from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1994, 1997, and 2005, respectively. His thesis was developed on the topic of networks-on-chip architectural optimization. He is currently an Associate Professor with the Federal University of Rio Grande do Norte (UFRN), Natal, Brazil. His research interests include embedded architectures modeling and specification, embedded software mapping, and communication/processing architectures optimization.

SAMAHERNI M. DIAS received the M.Sc. and Ph.D. degrees in electrical engineering from the Federal University of Rio Grande do Norte (UFRN), Brazil, in 2007 and 2010, respectively. He is currently a Professor with the Department of Electrical Engineering, UFRN. His research interests include variable structure control and embedded systems.