electronics

Article

Research on EDAC Schemes for Memory in Space Applications

Mengfu Chen [1,†], Chenguang Guo [2,*,†], Lei Chen [3], Wenjie Li [3], Fan Zhang [3], Xiaoxiang Hu [2] and Jiancheng Xu [2]

1 School of Public Administration, Beijing University of Aeronautics and Astronautics, Beijing 100191, China; cmf746663@shou.com
2 School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710072, China; xxhu@nwpu.edu.cn (X.H.); xujchg@nwpu.edu.cn (J.X.)
3 Beijing Microelectronics Technology Institute, Beijing 100076, China; chenleinpu@vip.126.com (L.C.); liwenjie_lwj@163.com (W.L.); zhangfan@mxtronics.com (F.Z.)
* Correspondence: guochg@nwpu.edu.cn; Tel.: +86-186-9187-4806
† These authors contributed equally to this work.

Abstract: Memory used for storing the configuration bitstream of field programmable gate array in space applications often encounters single event upset problems, which may disrupt the integrity of data in memory and lead to unpredictable failures. For commercial memories used in low Earth orbit (LEO), single-bit errors and double-byte errors account for a large proportion. Meanwhile, error detection and correction (EDAC) schemes, e.g., triple modular redundancy, linear block codes, memory scrubbing, and the combination of these schemes, are very popular in LEO missions. To further these works, a novel EDAC scheme with cascaded "Bose–Chaudhuri–Hocquenghem and cyclic redundancy check" codes and a proper scrubbing method is presented in this paper. The performance of the proposed design is measured and compared with state-of-the-art EDAC schemes in terms of hardware overhead, time overhead and error correction and detection capabilities. It is concluded that the proposed EDAC scheme is better suited for memory in space applications.

Keywords: EDAC; memory; Bose–Chaudhuri–Hocquenghem; cyclic redundancy check; single event upset; field programmable gate array

Citation: Chen, M.; Guo, C.; Chen, L.; Li, W.; Zhang, F.; Hu, X.; Xu, J. Research on EDAC Schemes for Memory in Space Applications. *Electronics* **2021**, *10*, 533. https://doi.org/10.3390/electronics10050533

Academic Editor: Gaetano Palumbo

Received: 11 January 2021
Accepted: 20 February 2021
Published: 25 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

It is well known that data stored in memory chips suffer from single event upsets (SEUs) in space applications. Bit-flips are induced naturally by cosmic radiation, extreme temperature, electromagnetic radiation, etc. There is a famous experiment of SEU observations for commercial memories based on the Alsat-1 satellite in low Earth orbit (LEO). The Alsat-1 was launched on November 2002 and the experiment lasted over eight years. Table 1 shows a summary of 32 MBytes Ramdisk of SEU observations [1]. The probability of single-bit errors is the highest, which is 98.59%, followed by the probability of double-byte errors, which is 1.223% and is dominated by double-bit error in single-byte.

Error detection and correction (EDAC) technologies which can detect and correct errors in a certain degree are very popular in improving the reliability of memory devices. For the transmission of secure data between devices and its local memory, triple modular redundancy (TMR) is widely used for anti-SEU design, but its probabilistic error correction ability can cause cumulative error [2,3]. Another disadvantage of TMR is its large memory overhead (200%) [4]. Linear block codes are also well-known and widely used. However, for the error correction requirements in Table 1, these famous linear block codes, e.g., Hamming code, Parity code, Berger code, are not available because of the limitation of their error-correcting capabilities [5,6]. In addition, linear block codes with excessive error-correcting capabilities (e.g., LDPC, Reed–Solomon Code) are not necessary because they will increase design complexity and cause large hardware overhead [7,8]. For scrubbing methods which are also famous in space applications, it is too time-consuming to rewrite the entire memory [9]. Besides, some researchers have found a transistor-level EDAC method with small hardware overhead, which helps design the radiation-hardened memory cell library but is not suitable for error correction, in Table 1 [10–12].

Table 1. Single event upset (SEU) observations for Alsat-1.

System Monitored	Parameters
Memory Size	32 M-Bytes
Observation period	2622 days, 29 November 2002–14 August 2010
Bits monitored	268,435,456
Total number of errors	265,649
No. of Single-bit errors	261,905 (98.59%)
No. of Double-byte errors	3249 (1.223%)
No. of Severe errors	247 (0.093%)
No. of Multiple-bit errors	233 (0.087%)
No. of Hardware errors	15 (0.005%)

Figure 1 shows the block diagram of an SRAM-based field programmable gate array (FPGA) in space applications. Once SEU occurs, it will lead to data flipping of the memory chip, which will affect the logical state of the FPGA, or even threaten the safety of the FPGA, resulting in irreparable failures [13]. An in-orbit controller acts as a bridge for the FPGA and its configuration memories. Programmable read-only memory (PROM) which is immune to SEU can be programmed only once. The bitstream stored in the PROM is usually used for the initialization of the FPGA, while the bitstream stored in the memory array is usually used for the functional reconfiguration of the FPGA. In addition, the bitstream stored in the memory array has the function of in-orbit maintenance and update through universal asynchronous receiver/transmitter (UART).

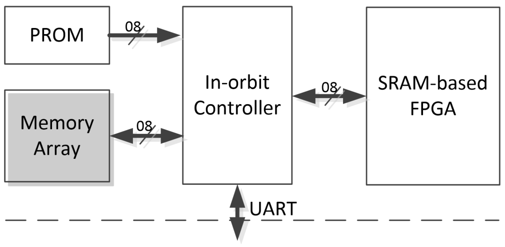

Figure 1. Block diagram of an SRAM-based field programmable gate array (FPGA) in space applications.

In order to improve the reliability of data in the memory array, the in-orbit controller is usually designed to detect and correct the configuration bitstream that may be affected by SEU. The corresponding steps are as follows:

(1) Data encoding: The configuration data are received through the UART, and then redundant encoding is performed by EDAC method.
(2) Periodic readback checking: Check the data stored in the memory array periodically through the FPGA readback function.
(3) Periodic scrubbing: According to the scrub interval set in advance, the memory array can be rewritten unconditionally by continuous scrubbing.

This paper focuses on hardware implemented EDAC schemes for memory in space applications. Section 2 presents the state-of-the-art EDAC schemes for memory in space applications. Then, a novel proposed EDAC scheme is given in Section 3. Section 4 analyzes experimental results. Finally, conclusions are drawn in Section 5.

2. Present State-of-the-Art EDAC Schemes

EDAC is committed to adding redundant bits to data in a specific way. When the data is corrupted, the redundant bits can help to identify and eliminate the corruption in a certain range. There are many EDAC schemes that meet the design purpose of this paper [14–16]. At present, the EDAC schemes that have been applied to space applications include TMR, linear block codes, memory scrubbing, and the combination of these schemes.

2.1. TMR

TMR is widely used for anti-SEU design. The implementation consists of three identical memories storing the same data and a voter [2,3], as shown in Figure 2. The voter outputs the majority of the data from the three memories, so that when SEU occurs in one of the memories, the output remains valid. Since the probability of error events occurring simultaneously in two memories is minimal, the error correction ability of the circuit can be improved by the TMR.

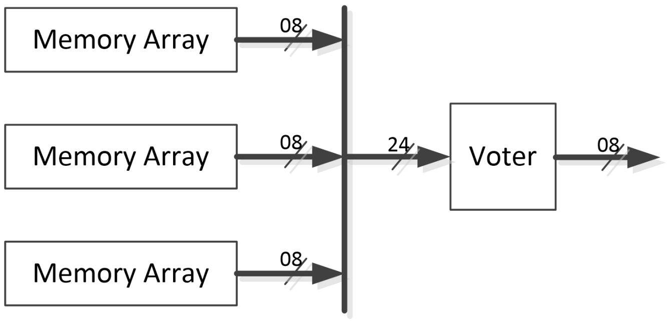

Figure 2. Block diagram of triple modular redundancy (TMR).

However, there is still a certain probability of error decoding, resulting in cumulative error after subsequent propagation. Furthermore, as can be seen from Figure 2, memory overhead is a major drawback of the TMR. It wastes twice the required memory.

2.2. Linear Block Code Schemes

The EDAC circuit for memory array including an encoder and a decoder is shown in Figure 3. The encoder generates redundant data bits based on input data, while the decoder uses redundant data bits to detect and eliminate errors in a certain range. For linear block code (n, k), the number of codeword data bits, encoded data bits and redundant data bits are n, k and c ($c = n - k$), respectively. Therefore, the encoder has k-bit input and n-bit output, while the decoder has n-bit input and k-bit output. However, for the data bus of the memory array, a DDR $\times 8$ chip with a bit width of eight is one of the most common options.

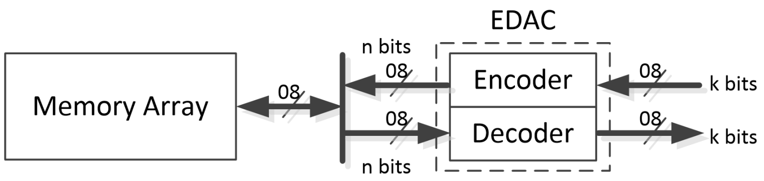

Figure 3. Block diagram of memory with error detection and correction (EDAC) circuit.

Typically, there are four parameters used to measure the performance of linear block codes, i.e., error correction capacity (t_c), error detection capacity (t_d), code rate and bit overhead. They are expressed by

$$t_c = \frac{d-1}{2} \tag{1}$$

$$t_d = d - 1 \tag{2}$$

$$Code\ Rate = \frac{k}{n} \tag{3}$$

$$Bit\ Overhead = \frac{c}{k} \tag{4}$$

where d is the minimum Hamming distance. In addition, time and space complexity needs to be considered to measure the efficiency of coding schemes.

According to the data in Table 1, it is necessary to select the coding scheme with proper Hamming distance so that t_c is equal to two or slightly higher. Therefore, some of the schemes, such as Hamming code, Parity code, Berger code, etc., do not meet this requirement [5,6]. In addition, Reed–Solomon code which is good at correcting non-binary codes is not considered in this paper [17]. Table 2 shows linear block code schemes with proper error correction capabilities, where $p \times q$ means the size of memory block.

Table 2. Linear block code schemes with proper t_c.

Code	**Scheme** (n,k,d)	t_d	t_c	**Code Rate**	**Bit Overhead**	**Complexity**	
						Time	**Space**
Hadamard	(16,5,8)	7	3	31.25%	220%	$O(nlog_2n)$	$O(nlog_2n)$
Binary Golay	(23,12,7)	6	3	52.17%	91.66%	$O(pq)$	$O(2^q)$
Ternary Golay	(11,6,5)	4	2	54.54%	83%	$O(pq)$	$O(2^q)$
BCH	(15,7,5)	-	2	46.67%	114.29%	$O(q^2)$	$O(pn)$
	(31,21,5)	-	2	67.74%	47.62%		
	(63,51,5)	-	2	80.95%	23.53%		
	(127,113,5)	-	2	88.98%	12.39%		
	(255,239,5)	-	2	93.73%	6.69%		
4D Parity	(561,560)	-	2	84.21%	18.56%	$O(pn)$	$O(pn)$
	(2193,2048)	-	2	87.67%	13.28%		

As can be seen from Table 2, Hadamard code has the lowest time and space complexity, but the ratio of code rate to bit overhead (C2B) is also the lowest, which means that the coding overhead is too high. Compared to Hadamard code, Golay codes have better coding overhead, but they are still not good enough. As the length of the Bose–Chaudhuri–Hocquenghem (BCH) code increases, the size of the corresponding memory block increases and the C2B ratio becomes better, but the time complexity increases exponentially [18]. 4D Parity codes have great C2B ratio and great time and space complexity. A big problem of 4D Parity codes is that it is applicable to the whole memory block, and its error correction and detection capability needs to be applied to the whole memory block, which means that the error correction and error detection time will be sacrificed in a certain degree [19,20].

2.3. TMR Based EDAC

To further improve the reliability of data in memory array, the combination of TMR and linear block code is very effective [4,21,22], as shown in Figures 4 and 5.

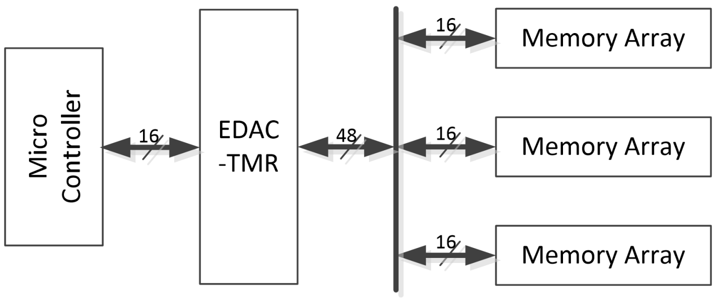

Figure 4. Block diagram of EDAC-TMR scheme with triple memories.

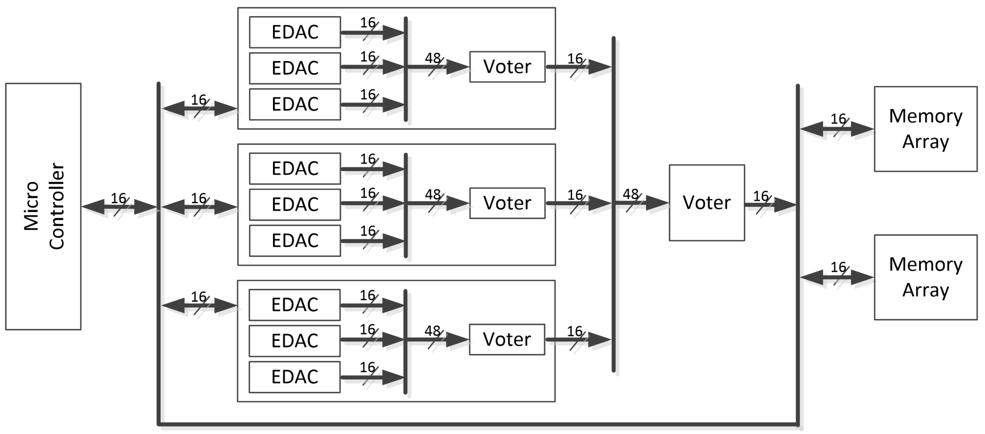

Figure 5. Block diagram of EDAC-TMR scheme with triple EDACs.

Both circuits inherit the error correction and detection function of the linear block code and the redundancy of TMR technology. However, the circuit shown in Figure 4 requires an additional two times the required memory, while the circuit shown in Figure 5 requires an additional eight times the required EDAC module and one time the required memory.

2.4. Memory Scrubbing

There are two kinds of scrubbing operation, one is scrubbing based on the UART in a fixed interval, the other is scrubbing based on the EDAC circuit [23,24]. For the former, the memory is rewritten unconditionally and the scrubbing interval is set according to the SEU rate. For the latter, the SEU is scrubbed out sequentially by decoding and rewriting operations. Since the EDAC circuit has inherent error correction and detection capabilities, potential errors can be automatically corrected and overridden. Memory scrubbing is also an effective EDAC solution, but the problem is that rewriting all data in memory is extremely time-consuming.

3. A New Proposed EDAC Scheme

For the purpose of reducing hardware and time overhead while meeting the anti-SEU requirements shown in Table 1, this section presents a new EDAC scheme by designing a cascaded code scheme and an improved scrubbing method.

3.1. Cascaded Code Scheme

Taking into account the time and space complexity and the C2B ratio in Table 2, BCH and 4D parity codes are suitable for data protection of memory in space applications. However, 4D parity codes have greater time overhead than BCH in response to errors, which can adversely affect memory scrubbing. Accordingly, this paper prefers BCH codes with similar C2B ratios to 4D parity codes.

Figure 6 illustrates the effect of increasing code length on C2B ratio of BCH and 4D parity codes. Greater code length means greater space complexity and greater hardware overhead. Therefore, the middle BCH(63, 51, 5) which has similar C2B ratio to 4D parity codes is more appropriate for single-bit and double-bit error correction. For other error cases with less error probability in Table 1, this paper uses cyclic redundancy check 32 (CRC32) code which has high error detection ability for error checking.

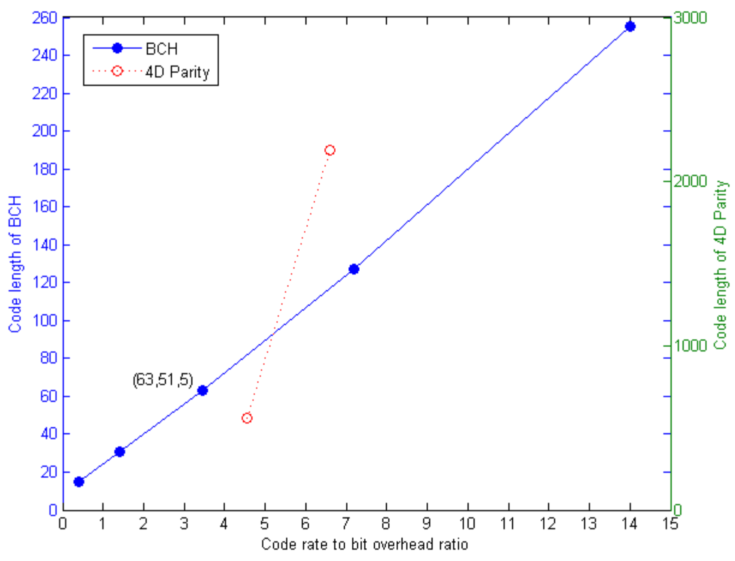

Figure 6. Code length to code rate to bit overhead (C2B) ratio.

Figure 7 shows the block diagram of the proposed EDAC system. Because the data bit width of the memory array in this paper is 8 bits, in byte, the calculation parallelism of the encoders (i.e., CRC_EN and BCH_EN) and decoders (i.e., CRC_DE and BCH_DE) designed below is eight bits.

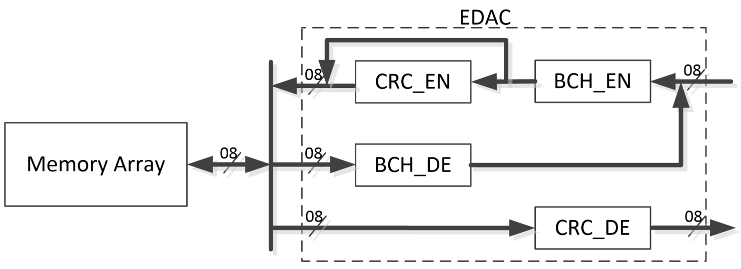

Figure 7. Block diagram of the proposed EDAC system.

3.2. BCH

For BCH(63, 51, 5) over GF(2) and GF(2^6), the data polynomial $m(x)$ and the generator polynomial $g(x)$ are defined as

$$m(x) = m_0 + m_1x + \cdots + m_{50}x^{50} \tag{5}$$

$$g(x) = x^{12} + x^{10} + x^8 + x^5 + x^4 + x^3 + 1 \tag{6}$$

3.2.1. Encoding

The codeword polynomial $c(x)$ is given by

$$c(x) = p(x) + x^{12}m(x) \tag{7}$$

where the remainder is

$$p(x) = x^{12}m(x) \bmod g(x) \tag{8}$$

Let

$$\begin{aligned} M_i =& m_{8i+7}x^7 + m_{8i+6}x^6 + \cdots + m_{8i+1}x + m_{8i}, \\ i =& 0, 1, \cdots, 6. \end{aligned} \tag{9}$$

where m_{55},m_{54},…,and m_{51} are zeros. Then

$$\begin{aligned} p(x) &= x^{12}m(x) \bmod g(x) \\ &= (((((((M_6x^8 + M_5)x^8 + M_4)x^8 + M_3)x^8 + M_2)x^8 \\ &\quad + M_1)x^8 + M_0)x^8 + 0)x^4 \bmod g(x) \end{aligned} \tag{10}$$

Since the encoded data bits cannot be divisible by eight, the formula for the first eight iterations of the encoder is

$$\begin{aligned} p_i(x)_1 &= (p_{i-1}(x)x^8 + M_i) \bmod g(x) \\ &= p_{i-1}(x)x^8 \bmod g(x) + M_i \end{aligned} \tag{11}$$

and the formula for the last iteration of the encoder is

$$p_i(x)_2 = p_{i-1}(x)x^4 \bmod g(x) \tag{12}$$

Accordingly, the schematic of the 8-bit parallel BCH(63, 51, 5) encoder is shown in Figure 8.

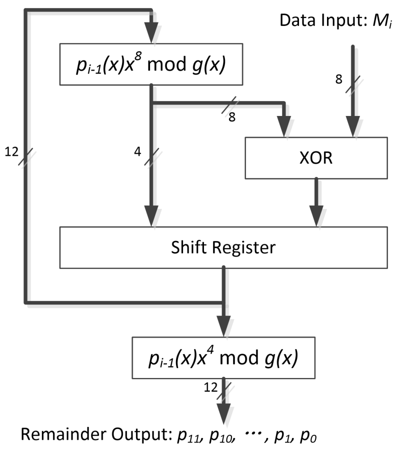
Data Input: M_i
$p_{i-1}(x)x^8$ mod $g(x)$
8
8
12
4
XOR
Shift Register
$p_{i-1}(x)x^4$ mod $g(x)$
12
Remainder Output: $p_{11}, p_{10}, \cdots, p_1, p_0$

Figure 8. Schematic of the 8-bit parallel BCH(63, 51, 5) encoder.

Then the codeword vector is given by

$$\begin{aligned} (c_{62}, c_{61}, \cdots, c_1, c_0) = &(m_{50}, m_{49}, \cdots, m_1, m_0, \\ &p_{11}, p_{10}, \cdots, p_1, p_0) \end{aligned} \tag{13}$$

3.2.2. Decoding

The decoding process of BCH includes three steps:

(1) Computing the syndrome polynomial $s(x)$ based on the input $r(x)$ to the decoder.
(2) Calculating the error location polynomial $\Lambda(x)$ by the key equation.
(3) Calculating the roots $(e(x))$ of $\Lambda(x)$ by Chien search algorithm, and correcting errors based on the roots:

$$\hat{c}(x) = r(x) - e(x) \tag{14}$$

The flow of BCH decoding is shown in Figure 9.

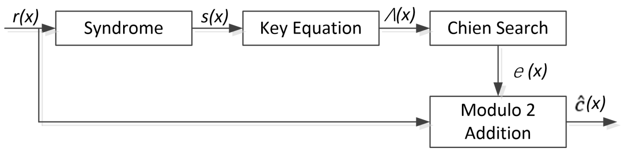

Figure 9. Flow of Bose–Chaudhuri–Hocquenghem (BCH) decoding.

(1) Syndrome

For BCH codes that can correct two bits errors, the number of syndrome values s_i should be four. s_i is defined as

$$s_i = \sum_{j=0}^{n-1} r_j \left(\alpha^i\right)^j = r_0\left(\alpha^i\right)^0 + r_1\left(\alpha^i\right)^1 + \cdots + r_{n-1}\left(\alpha^i\right)^{n-1} \tag{15}$$

where α is the primitive element of GF(2^6).

For 8-bit parallel computing, the above formula can be derived as

$$\begin{aligned} s_i = &\left(0\alpha^7 + r_{62}\alpha^6 + \cdots + r_{56}\right)\alpha^{56} + \left(r_{55}\alpha^7 + r_{54}\alpha^6 + \cdots \right. \\ &\left. + r_{48}\right)\alpha^{48} + \cdots + \left(r_7\alpha^7 + r_6\alpha^6 + \cdots + r_0\right) \end{aligned} \tag{16}$$

The corresponding calculation circuit is shown in Figure 10.

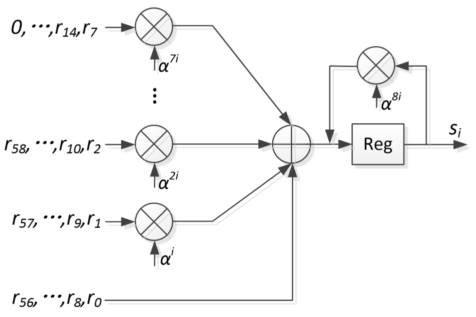

Figure 10. Schematic of the syndrome computation circuit.

(2) Key equation

The error location polynomial $\Lambda(x)$ for the BCH(63, 51, 5) is defined as

$$\Lambda(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 \tag{17}$$

and can be calculated by the SiBM algorithm through the key equation

$$s(x)\Lambda(x) = \Omega(x) \bmod x^4 \tag{18}$$

where $\Omega(x)$ is the error value polynomial. The SiBM algorithm and its formula derivation process are described in detail in Ref. [25].

Figures 11 and 12 show the implementation of the SiBM algorithm for the BCH(63, 51, 5) and its basic processing element (PE), respectively.

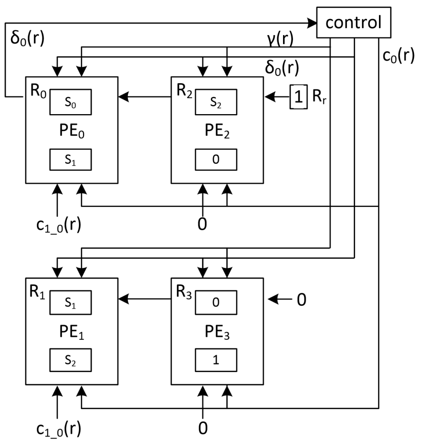

Figure 11. Schematic of the key equation computation circuit.

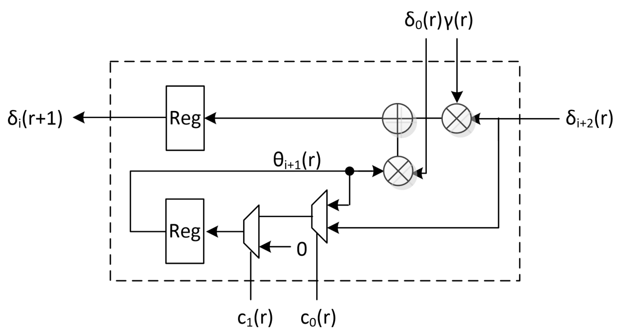

Figure 12. Schematic of the processing element.

After two iterations, the output of the registers ($R_0 \sim R_2$) shown in Figure 11 is the coefficients ($\lambda_0 \sim \lambda_2$) of the error location polynomial $\Lambda(x)$.

(3) Chien search

The Chien search algorithm searches the error location by checking whether $\Lambda(\alpha^i)$ is zero. $\Lambda(\alpha^i)$ can be written as

$$\Lambda\left(\alpha^i\right) = \sum_{j=0}^{2} \lambda_j \alpha^{ij} = \sum_{j=1}^{2} \lambda_j \alpha^{ij} + \lambda_0, \quad i = 0, 1, \cdots, 62. \tag{19}$$

If $\Lambda(\alpha^i)$ is equal to zero, it means that the input data r_i has an error, otherwise there is no error. For 8-bit parallel computing, eight of the above formulas are calculated simultaneously, as shown in Figure 13.

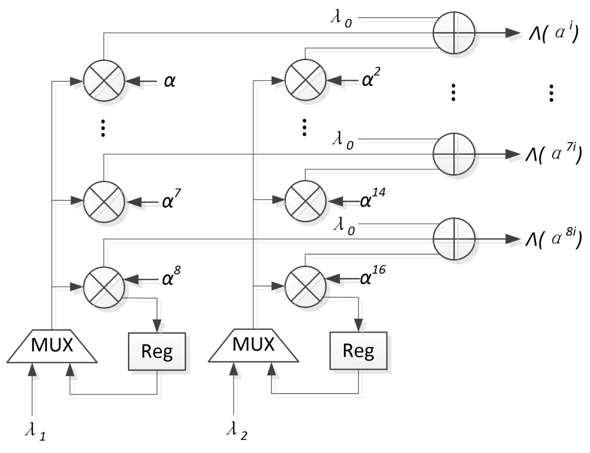

Figure 13. Schematic of the Chien search circuit.

After Chien search, the roots ($e(x)$) of $\Lambda(x)$ are calculated. Then errors in the decoded codeword $\hat{c}(x)$ can be corrected by Formula (14).

3.3. CRC

Since the BCH(63, 51, 5) is not sensitive to errors greater than two bits, CRC code can be used as a supplement for error detection of other error cases shown in Table 1. The probability of missed detection of the CRC32 is 2^{-32}, which can cover almost all error cases in Table 1.

CRC code is mainly used for the binary data. The data polynomial $m(x)$ and the generator polynomial $g(x)$ of the CRC32 are defined as

$$m(x) = m_0 + m_1 x + \cdots + m_{63} x^{63} \tag{20}$$

$$\begin{aligned} g(x) =& x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} \\ & + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \end{aligned} \tag{21}$$

where m_{63} is zero.

3.3.1. Encoding

The encoding process of the CRC32 is similar to the BCH(63, 51, 5). The codeword polynomial $c(x)$ is given by

$$c(x) = p(x) + x^{32} m(x) \tag{22}$$

where the remainder is

$$p(x) = x^{32} m(x) \bmod g(x) \tag{23}$$

Let

$$\begin{aligned} M_i =& m_{8i+7} x^7 + m_{8i+6} x^6 + \cdots + m_{8i+1} x + m_{8i}, \\ i =& 0, 1, \cdots, 7. \end{aligned} \tag{24}$$

Then

$$\begin{aligned} p(x) =& x^{32} m(x) \bmod g(x) \\ =& ((((((((((M_7 x^8 + M_6) x^8 + M_5) x^8 + M_4) x^8 \\ & + M_3) x^8 + M_2) x^8 + M_1) x^8 + M_0) x^8 \\ & + 0) x^8 + 0) x^8 + 0) x^8 \bmod g(x) \end{aligned} \tag{25}$$

The formula for the iterations of the encoder is

$$\begin{aligned} p_i(x) =& (p_{i-1}(x) x^8 + M_i) \bmod g(x) \\ =& p_{i-1}(x) x^8 \bmod g(x) + M_i \end{aligned} \tag{26}$$

Accordingly, the schematic of the 8-bit parallel CRC32 encoder is shown in Figure 14.

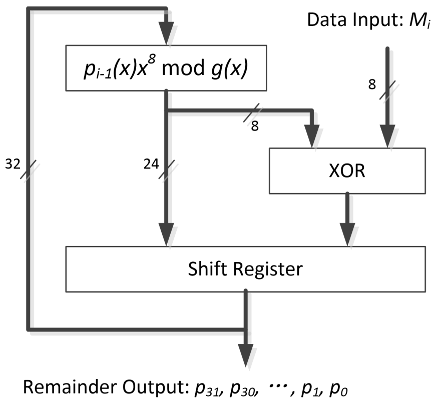
Data Input: M_i
$p_{i-1}(x)x^8$ mod $g(x)$
8
8
32
24
XOR
Shift Register
Remainder Output: $p_{31}, p_{30}, \cdots, p_1, p_0$

Figure 14. Schematic of the 8-bit parallel cyclic redundancy check 32 (CRC32) encoder.

After 13 iterations, the codeword vector is given by

$$\begin{aligned} (c_{95}, c_{94}, \cdots, c_1, c_0) =& (m_{63}, m_{62}, \cdots, m_1, m_0, \\ & p_{31}, p_{10}, \cdots, p_1, p_0) \end{aligned} \tag{27}$$

3.3.2. Decoding

The decoding process of the CRC32 is similar to its encoding process. The novel remainder is

$$\hat{p}(x) = r(x) \bmod g(x) \tag{28}$$

where $r(x)$ is the received input data from memory.

The schematic of the 8-bit parallel CRC32 decoder is shown in Figure 15.

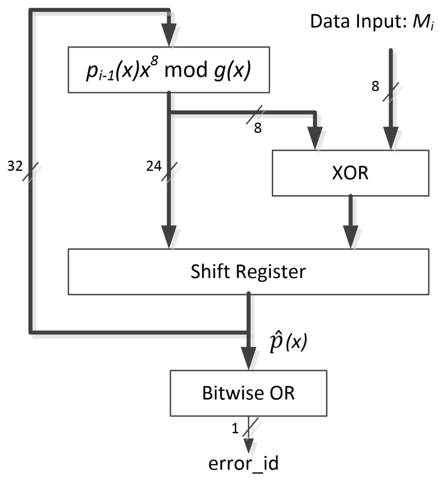

Figure 15. Schematic of the 8-bit parallel CRC32 decoder.

After 13 iterations, if the bitwise-OR result of $\hat{p}(x)$ is equal to zero, it indicates that the input $r(x)$ has no error, otherwise an error is detected.

3.4. Proposed EDAC Process

The proposed EDAC process includes data encoding, periodic readback checking and scrubbing, respectively.

(1) Data encoding: Each 51 bits of data received through the UART is encoded by the BCH_EN to 63 bits. After adding one bit zero, the new 64 bits data are encoded by the CRC_EN to 96 bits. The codeword vector stored in memory array is

$$\begin{aligned}(c_{95}, c_{94}, \cdots, c_1, c_0) = &(0, m_{50}, m_{49}, \cdots, m_1, m_0, \\ &p_{b11}, p_{b10}, \cdots, p_{b1}, p_{b0}, \\ &p_{c31}, p_{c30}, \cdots, p_{c1}, p_{c0}),\end{aligned} \tag{29}$$

where p_{bi} and p_{ci} means that the remainder values of the BCH_EN and the CRC_EN, respectively. In addition, as each 51 bits of data are compiled to 96 bits, the memory overhead of the proposed scheme will increase by 88.24%.

(2) Periodic readback checking: Check the data stored in memory array periodically through the FPGA readback function. The errors in Table 1 are divided into three cases, i.e., no errors, single-bit or double-bit errors and multiple-bit errors which have more than two bits errors. The corresponding decoding process are:

- No errors: Both the BCH_DE and the CRC_DE indicate that there are no errors.
- Single-bit or double-bit errors: If the decoding result of the BCH_DE is wrong, the decoded 51 bits data are encoded again by the BCH_EN and written to the blank area of the memory array. Using new encoded data and the old CRC redundant bits, the CRC_DE can identify whether the error has been corrected by the BCH_DE. If yes, overwrite the original data in the memory array with the new encoded data. Otherwise, it means that there are multiple-bit errors and the corresponding memory address should be marked.

- Multiple-bit errors: Other than the above multiple-bit errors, if the decoding result of the BCH_DE is right and the decoding result of the CRC_DE is wrong, it also means that there are multiple-bit errors and the corresponding memory address should be marked.

(3) Scrubbing: According to the marked memory addresses, memory array can be partially rewritten through the UART.

Obviously, the probability of the multiple-bit errors in Table 1 is relatively small, which means that the scrubbing step is rarely implemented. Therefore, the proposed memory scrubbing method is very time-saving.

4. Experimental Results

The proposed EDAC scheme was implemented in an In-orbit controller chip (Bsv5cbrh) of Beijing Microelectronics Technology Institute as described in Section 3. The EDAC system shown in Figure 7 was verified both in function and timing. Meanwhile, a PCB-level fault injection system based on Figure 1 was developed to validate the proposed scheme, including Bsv5cbrh, Xq5vsx95t, Xcf32p, etc., as shown in Figure 16. The proposed approach of the in-orbit controller was implemented based on the process described in Section 3.4. Correspondingly, the relationship between EDAC capability and the use of EDAC resources is shown in Table 3.

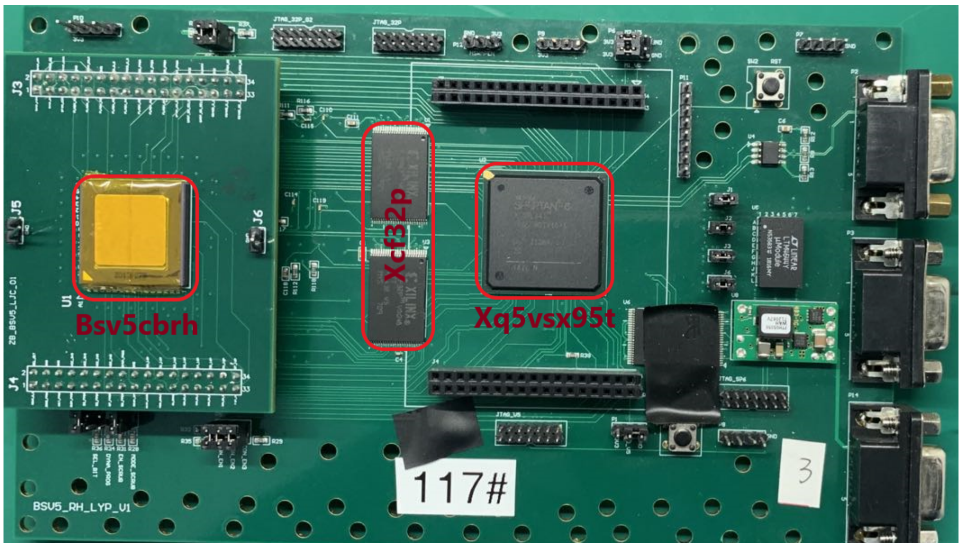

Figure 16. A PCB-level fault injection system.

Table 3. Relationship between EDAC capability and the use of EDAC resources.

	No Errors	SED	SEC	DED	DEC	MED		MEC	
BCH_EN		✓	✓	✓	✓		✓	✓	✓
BCH_DE	✓	✓	✓	✓	✓	✓	✓	✓	✓
CRC_EN								✓	✓
CRC_DE	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scrubbing								✓	✓

Note: SED: single error detection, SEC: single error correction, DED: double error detection, DEC: double error correction, MED: multiple error detection, MEC: multiple error correction.

Based on the description of Section 3.4, it can be indicated from the results of the BCH_DE and the CRC_DE if there were no errors. For single-bit errors, all the locations in the memory were simulated to ensure 100% fault coverage. The result of the CRC_DE should be correct in this case. The BCH_EN was used to encode the decoded data from the BCH_DE, and the BCH_DE was used twice to check whether the error can be corrected or not. For double-bit errors in each 63-bit, pseudo-random sequence (i.e., Pseudo-Noise

Code) to simulate the fault injection address was used, in which case the corresponding 32-bit data checked by the CRC_DE should also be correct. The use of EDAC resources in this case was the same as the use of EDAC resources to handle single-bit errors. For multiple-bit error detection, there was no need to rewrite the whole memory or the memory corresponding to the marked address, it could be judged by the combination of the results from the BCH_DE and the CRC_DE. Accordingly, the uses of EDAC resources were similar to the two cases described earlier (i.e., no errors and single-bit errors). For multiple-bit error correction, in addition to the previous case, memory scrubbing function was triggered to ensure 100% error correction, in which case all EDAC resources were used.

To measure the performance of the proposed EDAC, a comparison between the state-of-the-art EDAC schemes and the proposed EDAC scheme is shown in Table 4. The overall memory overhead is an increase of 88.24% in stored data, which is smaller than when using TMR or EDAC-TMR methods. The overall EDAC overhead is an increase of 100% in EDAC module, which is significantly smaller than when using EDAC-TMR method shown in Ref. [26]. The time overhead of the proposed EDAC is greater than the time overhead of TMR but less than the time overhead of EDAC-TMR methods shown in Table 4. Furthermore, since it inherited the error correction and detection capability of the cascaded "BCH(63, 51, 5) and CRC32" codes, the proposed EDAC scheme could correct single-bit and double-bit errors and detect multiple-bit errors. By memory scrubbing circuit, the proposed EDAC scheme could also correct multiple-bit errors. Hence, the proposed EDAC scheme had the highest error detection and correction capability shown in Table 4.

Table 4. Comparison of experimental results.

EDAC Type	TMR	EDAC-TMR 1 [4]	EDAC-TMR 2 [26]	The Proposed EDAC
EDAC overhead	0	0	800%	100%
Memory overhead	200%	200%	100%	88.24%
Time overhead	t_V	$t_E + t_V$	$t_E + 2t_V$	t_E
EDAC Capability	MEC-MED	DEC-DED	DEC-DED	MEC-MED
Severe errors correction	NO	YES	YES	YES
Corrected in-orbit errors	98.7%	100%	100%	100%

Note: t_E: EDAC delay time, t_V: Voter delay time.

Based on the proposed EDAC scheme, the encoding delay for the cascaded "BCH_EN and CRC _EN" was only 21 clock cycles, while the decoding delays for the BCH_DE and the CRC_DE were 20 clock cycles and 13 clock cycles, respectively. In addition, instead of rewriting all data in memory in a certain interval, the proposed EDAC scheme automatically scrubbed out single-bit errors and double-bit errors through the proposed EDAC circuit, and overrode the data through the UART with the marked memory addresses. For memory scrubbing based on the EDAC circuit, the proposed design could correct errors after 96 bits data were detected, while the design based on 4D parity codes must correct errors after a whole memory block was detected. For memory scrubbing through the UART, the memory addresses were marked only if there were multiple-bit errors in each 96-bit block, severe errors, and hardware errors shown in Table 1, which meant that the probability of memory scrubbing through the UART was less than 1.408%. Therefore, compared to the error-correcting time based on the EDAC circuit using 4D parity codes and the rewriting time for the whole memory, the proposed scrubbing method was relatively time-saving.

5. Conclusions

This paper has presented a novel EDAC scheme of the In-orbit controller chip (Bsv5cb rh) for memory in space applications. The EDAC scheme is based on the combination of the cascaded "BCH(63, 51, 5) and CRC32" codes and an improved scrubbing method. This scheme is sufficient to handle the typical SEU rate at LEO environment. The design is verified both in function and timing. The overall system cost is significantly smaller than the present state-of-the-art EDAC schemes. Experiments to simulate the error cases in Table 1 have shown that 100% of the errors can be detected and corrected.

Author Contributions: Conceptualization, M.C., C.G. and L.C.; formal analysis, M.C., C.G., W.L., F.Z., and X.H.; methodology, C.G. and J.X.; writing—original draft, M.C. and C.G.; writing—review and editing, L.C., W.L., F.Z., X.H. and J.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bentoutou, Y.; Mohammed, A.S. A Review of in-orbit Observations of Radiation-Induced Effects in Commercial Memories on board ALSAT-1. *World Acad. Sci. Eng. Technol. J.* **2012**, *6*, 652–654.
2. Yoshinaga, T.; Watanabe, M. Optically Reconfigurable Gate Array with a Triple Modular Redundancy. In Proceedings of the 2019 6th International Conference on Space Science and Communication (IconSpace), Johor Bahru, Malaysia, 28–30 July 2019; pp. 276–279.
3. Guo, C.; Xu, J.; Jiang, S.; An, Y.; Yue, L. A Novel Universal Radiation-hardened FFT Chip Design. In Proceedings of the 2018 International Conference on Radiation Effects of Electronic Devices (ICREED), Harbin, China, 7–11 May 2018; pp. 1–4.
4. Bentoutou, Y. Program Memories Error Detection and Correction On-Board Earth Observation Satellites. *World Acad. Sci. Eng. Technol.* **2010**, *4*, 933–936.
5. Tausch, H.J.; Puchner, H. Analysis of Hamming EDAC SRAMs Using Simplified Birthday Statistics. *IEEE Trans. Nucl. Sci.* **2015**, *62*, 1771–1778. [CrossRef]
6. Ahmad, S.; Zahra, M.; Farooq, S.Z.; Zafar, A. Comparison of EDAC schemes for DDR memory in space applications. In Proceedings of the 2013 International Conference on Aerospace Science Engineering (ICASE), Islamabad, Pakistan, 21–23 August 2013; pp. 1–5.
7. Zhang, W.; Kang, J.; Dong, Z.; Zhu, Y. RS-LDPC Concatenated Coding for NAND Flash Memory: Designs and Reduction of Short Cycles. In Proceedings of the 2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP), Shanghai, China, 12–15 September 2020; pp. 342–347. [CrossRef]
8. Sandell, M.; Ismail, A. Machine Learning for LLR Estimation in Flash Memory With LDPC Codes. *IEEE Trans. Circuits Syst. Express Briefs* **2021**, *68*, 792–796. [CrossRef]
9. Wang, W.; Ho, C.; Chang, Y.; Kuo, T.; Lin, P. Scrubbing-Aware Secure Deletion for 3-D NAND Flash. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2790–2801. [CrossRef]
10. Khayatzadeh, M.; Saligane, M.; Wang, J.; Alioto, M.; Blaauw, D.; Sylvester, D. 17.3 A reconfigurable dual-port memory with error detection and correction in 28nm FDSOI. In Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 31 January–4 February 2016; pp. 310–312.
11. Zhang, Y.; Khayatzadeh, M.; Yang, K.; Saligane, M.; Pinckney, N.; Alioto, M.; Blaauw, D.; Sylvester, D. 8.8 iRazor: 3-transistor current-based error detection and correction in an ARM Cortex-R4 processor. In Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 31 January–4 February 2016; pp. 160–162.
12. Zhang, Y.; Khayatzadeh, M.; Yang, K.; Saligane, M.; Pinckney, N.; Alioto, M.; Blaauw, D.; Sylvester, D. iRazor: Current-Based Error Detection and Correction Scheme for PVT Variation in 40-nm ARM Cortex-R4 Processor. *IEEE J. Solid-State Circuits* **2018**, *53*, 619–631. [CrossRef]
13. Cannon, M.J.; Keller, A.M.; Rowberry, H.C.; Thurlow, C.A.; Pérez-Celis, A.; Wirthlin, M.J. Strategies for Removing Common Mode Failures From TMR Designs Deployed on SRAM FPGAs. *IEEE Trans. Nucl. Sci.* **2019**, *66*, 207–215. [CrossRef]
14. Middlestead, R.W. Coding for improved communications. In *Digital Communications with Emphasis on Data Modems: Theory, Analysis, Design, Simulation, Testing, and Applications*; Wiley: Hoboken, NJ, USA, 2017; pp. 261–337.
15. Kaur, J.; Chopra, S.R. Performance evaluation of space codes using 16-QAM technique. In Proceedings of the 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), Noida, India, 10–11 August 2017; pp. 1–6
16. Liang, X.; Zhou, H.; Wang, Z.; You, X.; Zhang, C. Segmented successive cancellation list polar decoding with joint BCH-CRC codes. In Proceedings of the 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 29 October–1 November 2017; pp. 1509–1513.
17. Xiao, L.; Sun, Z.; Zhu, M. Efficient encoding and decoding algorithm used in Reed-Solomon codes for multiple fault-tolerance memories. In Proceedings of the 2011 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference, Harbin, China, 26–30 July 2011; Volume 2, pp. 1569–1572.
18. Poolakkaparambil, M.; Mathew, J.; Jabir, A.M.; Pradhan, D.K.; Mohanty, S.P. BCH code based multiple bit error correction in finite field multiplier circuits. In Proceedings of the 2011 12th International Symposium on Quality Electronic Design, Santa Clara, CA, USA, 14–16 March 2011; pp. 1–6.

19. Imran, M.; Al-Ars, Z.; Gaydadjiev, G.N. 4-D parity codes for soft error correction in aerospace applications. In Proceedings of the 2011 IEEE 6th International Design and Test Workshop (IDT), Beirut, Lebanon, 11–14 December 2011; pp. 104–109.
20. Anne, N.B.; Thirunavukkarasu, U.; Latifi, S. Three and four-dimensional parity-check codes for correction and detection of multiple errors. In Proceedings of the International Conference on Information Technology: Coding and Computing, Las Vegas, NV, USA, 4–7 April 2004; Volume 2, pp. 837–842.
21. Bentoutou, Y. A real time low complexity codec for use in low Earth orbit small satellite missions. *IEEE Trans. Nucl. Sci.* **2006**, *53*, 1022–1027. [CrossRef]
22. Bentoutou, Y.; Djaifri, M.; Mohammed, A.S. Design and Implementation of a quasi-cyclic Codec for Random Access Memories on board Alsat-1. *Acta Astronaut. Elsevier Sci.* **2010**, *66*, 954–961. [CrossRef]
23. Mousavi, M.; Pourshaghaghi, H.R.; Corporaal, H.; Kumar, A. Scatter Scrubbing: A Method to Reduce SEU Repair Time in FPGA Configuration Memory. In Proceedings of the 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Frascati, Italy, 19–21 October 2020; pp. 1–6.
24. Li, Y.; Chen, Z.; Shen, Y.; Wang, Y. Improved hybrid scrubbing scheme for spaceborne static random access memory-based field programmable gate arrays. *J. Eng.* **2019**, *2019*, 6024–6027. [CrossRef]
25. Liu, W.; Rho, J.; Sung, W. Low-Power High-Throughput BCH Error Correction VLSI Design for Multi-Level Cell NAND Flash Memories. In Proceedings of the 2006 IEEE Workshop on Signal Processing Systems Design and Implementation, Banff, AB, Canada, 2–4 October 2006; pp. 303–308.
26. Bentoutou, Y. A Real Time EDAC System for Applications Onboard Earth Observation Small Satellites. *IEEE Trans. Aerosp. Electron. Syst.* **2012**, *48*, 648–657. [CrossRef]