# Design of Two Interleaved Error Detection and Corrections using Hsiao Code and CRC

Mong Tee Sim
*Dept. Computer Science*
*University of Colorado Colorado Springs*
Colorado Springs, CO, USA
mong_sim@hotmail.com

Yanyan Zhuang
*Dept. Computer Science*
*University of Colorado Colorado Springs*
Colorado Springs, CO, USA
yzhuang@uccs.edu

***Abstract*—A radiation-induced single-event upset (SEU) is a major disruption to electronics operating in satellites. If not rectified, a single-bit-error can become uncorrectable. In this paper, we present two interleaved Double-Adjacent-Error-Corrections (DAECs) for Error Detection and Correction (EDAC), using the Hsiao Code and Cyclic Redundancy Code (CRC). From our results, both EDACs can correct 100% of the single-bit-errors, double-adjacent-bit-errors, and detect double-bit-errors and up to four-adjacent-bit-errors. The Hsiao Code EDAC encoder design requires less bit-weight than the CRC EDAC, which is an attribute of a high-speed EDAC. The CRC EDAC, in contrast, has a higher error detection rate for both three- and four-bit-errors, at 96.25% and 91.92%, respectively. We also propose two storage formats and algorithm designs that can manage and store the 48-bit codeword in 8-bit and 16-bit memory devices, a typical satellite scenario where board space is scarce. We used Verilog HDL, C++, and ModelSim to create and test our designs.**

***Keywords— Hsiao Code, CRC, Hamming Code, BCH, EDAC***

## I. INTRODUCTION

As a semiconductor's geometry gets smaller, the probability of a bit inversion in memory devices induced by external influences, such as an electrical surge and ionizing radiation increases (Fig. 1 [8]). These anomalies interrupt the regular operation of an electronic device, causing mission failures. An Error Detection and Correction (EDAC) design mitigates these anomalies by detecting and correcting any bit inversions in memory devices such as the static memory (SRAM and SSRAM), and dynamic memory (SDRAM and DRAM).

A safety-critical, fault-tolerant system requires a stringent specification for its EDACs, with minimizing erroneous correction rate as the goal. There are several prominent single error correction and double error detection (SEC-DED) EDACs proposed, such as the Hamming [2], Bose–Chaudhuri–Hocquenghem (BCH), and Hsiao Codes [1]. The Hsiao Code EDAC has the best performance for speed, hardware requirement, and three- and four-bit-error detection. Its higher three- and four-bit-error detection rates reduce the probability of an erroneous EDAC correction.

Fig. 2 shows the Hsiao Code's three- and four-bit-error erroneous correction rates vs. the number of check-bits. We generated the (22, 16), (23, 16), and (24, 16) Hsiao matrices to verify if the number of check-bits used, 6, 7, and 8 bits, in this case, can affect the performance of the EDAC. Based on our findings, the EDAC that uses fewer check-bits has a higher erroneous correction rate, i.e., with six check-bits (6CB), the probability of an erroneous correction is 49.8%. Based on Fig. 1, the likelihood that some multiple-bit corrupting code or data gets into the system's memory devices undetected can be as high as 1.4% per day in outer space [8]. This high erroneous correction rate is intolerable in such safety-critical systems.

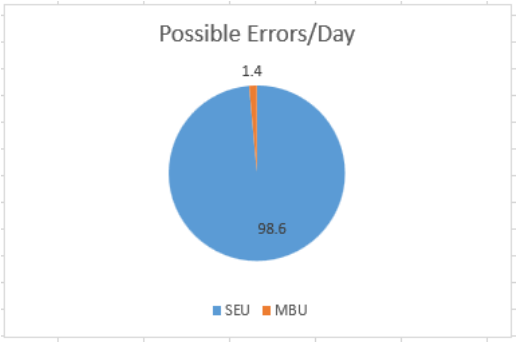


Fig. 1: Possible errors per day induced by radiation in space [8] (SEU: Single-Event Upset, MBU: Multiple-Bit Upset).

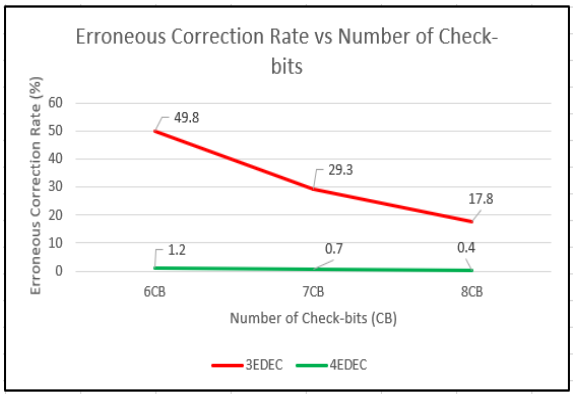


Fig. 2: This diagram shows the erroneous correction rates in percentage vs the number check-bits (3EDEC: three-bit-error detection erroneous correction, 4EDEC: four-bit-error detection erroneous correction).

Our approach is to design two fast EDACs that are capable of correcting and detecting these errors. The first EDAC uses the Hsiao Code and requires an encoder matrix with a bit-weight of 56 (Fig. 5). Our second EDAC uses a CRC [9] polynomial that requires an encoder matrix with a bit-weight of 62 (Fig. 9). Although having a higher bit-weight, the CRC-based EDAC has a better three- and four-bit-error detection rate than the Hsiao Code, at 96.25% and 91.92%, respectively.

The rest of this paper starts with our Hsiao Code's EDAC implementation using the Hsiao Code matrix generator in Section II. CRC's EDAC implementation using the CRC generator is described in Section III. Our two interleaved Double-Adjacent-Error-Correction (DAEC) EDAC designs are detailed in Section IV. Our storage format and algorithm designs to manage and store the 48-bit codeword in 8-bit and 16-bit memory devices are introduced in Section V. Section

978-1-7281-5414-5/20/$31.00 ©2020 IEEE

VI shows how we validate our EDACs. We review the related work in Section VII, and finally, Section VIII concludes.

## II. Design of a (24, 8) Hsiao Code's Encoder Matrix

The Hsiao Code implementation is straightforward. However, balancing the Hsiao Code's encoder matrix (also called the H-matrix) rows' bit-weights by hand is not an easy task. Instead, we created a C++ program, a Hsiao Code's matrix generator, to generate the H-matrices compliant with the three Hsiao Code's rules [1]. We also created another C++ program, a code generator, to take the generated matrix as the input and produce the Verilog code for the EDAC's encoder.

| | | | | | |
|---|---|---|---|---|---|
| 0: 1 2 3 | 10: 1 3 8 | 20: 1 7 8 | 30: 2 5 6 | 40: 3 5 6 | 50: 4 6 8 |
| 1: 1 2 4 | 11: 1 4 5 | 21: 2 3 4 | 31: 2 5 7 | 41: 3 5 7 | 51: 4 7 8 |
| 2: 1 2 5 | 12: 1 4 6 | 22: 2 3 5 | 32: 2 5 8 | 42: 3 5 8 | 52: 5 6 7 |
| 3: 1 2 6 | 13: 1 4 7 | 23: 2 3 6 | 33: 2 6 7 | 43: 3 6 7 | 53: 5 6 8 |
| 4: 1 2 7 | 14: 1 4 8 | 24: 2 3 7 | 34: 2 6 8 | 44: 3 6 8 | 54: 5 7 8 |
| 5: 1 2 8 | 15: 1 5 6 | 25: 2 3 8 | 35: 2 7 8 | 45: 3 7 8 | 55: 6 7 8 |
| 6: 1 3 4 | 16: 1 5 7 | 26: 2 4 5 | 36: 3 4 5 | 46: 4 5 6 | |
| 7: 1 3 5 | 17: 1 5 8 | 27: 2 4 6 | 37: 3 4 6 | 47: 4 5 7 | |
| 8: 1 3 6 | 18: 1 6 7 | 28: 2 4 7 | 38: 3 4 7 | 48: 4 5 8 | |
| 9: 1 3 7 | 19: 1 6 8 | 29: 2 4 8 | 39: 3 4 8 | 49: 4 6 7 | |

Fig. 3: The 56x3 array generated in Step 2 (Section II).

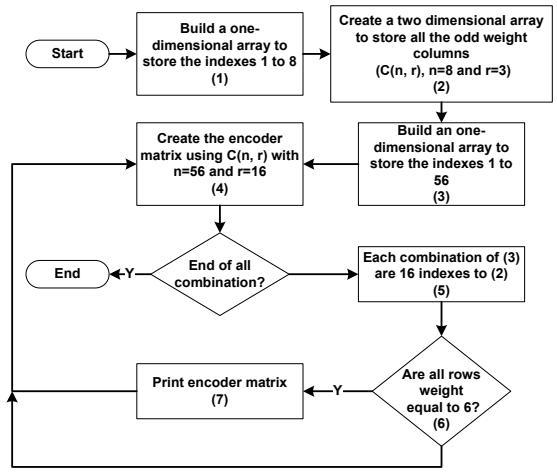


Fig. 4: Hsiao Code' encoder matrix generator.

In the following, we introduce the three Hsiao Code's rules and show how we can use them to create our Hsiao Code's EDAC H-matrix:

1. Every column should have an odd number of 1's; i.e., all column vectors are of odd-weights. In the following, we use n to indicate the number of rows in the H-matrix, and r to indicate the number of 1's in each column, which must be an odd number.
   - use the combination formula C(n, r)=((n!)/ (r! (n-r)!)) to generate all possible odd-weighted columns with n=8 and r=3. n=8 because the matrix has eight rows. r=3 because 3 is the smallest odd number next to 1, and 1 is already used in the check-bit columns. We therefore get a total of 56 combinations.

2. The total number of 1's in the H-matrix should be at a minimum.
   - We compute the 1's in the H-matrix as (16 * 3) + 8, or 56 1s' since the first 16 columns of the matrix contain three 1's, and the last eight columns are for check-bits.

3. The number of 1's in each row of the H-matrix should be made equal, or as close as possible, to the average number, i.e., the total number of 1's in the H-matrix divided by the number of rows.
   - We divide the 56 1s' by 8 and get 7 1s' per row. This number includes the check-bit in the last eight columns.

Based on the rules above, we designed our matrix generator in C++. Fig. 4 shows the flowchart of our generator that works as follows:

1. First, build a one-dimensional array to store the row indexes, 1 to 8, since the new matrix has eight rows. Each of the numbers refers to one of the eight rows.

2. a 56x3 array to store all the combinations of the row indexes in the column vectors that each contains a 1. Since each column vector in the H-matrix has three 1's, we choose the combinations of three out of the eight row indexes generated in step 1 (1 to 8) and store each combination in a column vector in this 56x3 array (Fig. 3). There are a total of 56, or C(8, 3), such combinations stored in this array.

3. Build one-dimensional array to store the numbers 1 to 56. This is an array of the column indexes of the 56x3 array generated in step 2.

4. the column indexes to build the H-matrix as follows. Since the H-matrix has 16 columns, not including the eight columns used for the check-bit, and each column contains three 1's, we choose a combination of 16 out of the 56 column indexes generated in step 3 (1 to 56). If we have exhausted all the C(56, 16) combinations, then end the program; otherwise continue to step 5.

5. Use the 16 indexes generated in step 4 to index into the 56x3 array shown in Fig. 3, and obtain a 16x3 sub-array. Use each of the 16 columns to build a 16x8 matrix. For example, if the first column in Fig. 3 is chosen, then we build a column vector in the new matrix containing eight rows, by storing 1's in rows 1, 2, and 3, and 0's in all remaining rows in this column vector.

6. If all the rows in the matrix generated in step 5 contain 6 1s' (not including the check-bit), go to step 7, otherwise go to step 4.

7. Print the encoder matrix, indicating that we successfully generated the matrix; then go to step 4 to check if we have exhausted all the C(56, 16) combinations.

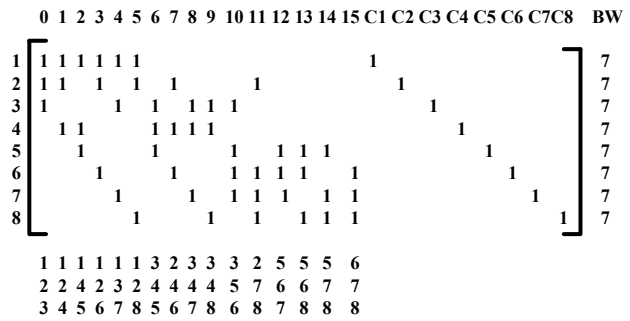


Fig. 5: This diagram shows the Hsiao Code's encoder matrix generated by the matrix generator.

```
//------------------------------------------------------------
// EDACENCODE
//------------------------------------------------------------
module EDACENCODE(
        input  wire [15:0]   d,
        output wire [7:0]    cb
);

assign cb[0] = d[0] ^ d[1] ^ d[2]  ^ d[3]  ^ d[4]  ^ d[5];
assign cb[1] = d[0] ^ d[1] ^ d[3]  ^ d[5]  ^ d[7]  ^ d[11];
assign cb[2] = d[0] ^ d[4] ^ d[6]  ^ d[8]  ^ d[9]  ^ d[10];
assign cb[3] = d[1] ^ d[2] ^ d[6]  ^ d[7]  ^ d[8]  ^ d[9];
assign cb[4] = d[2] ^ d[6] ^ d[10] ^ d[12] ^ d[13] ^ d[14];
assign cb[5] = d[3] ^ d[7] ^ d[10] ^ d[12] ^ d[13] ^ d[15];
assign cb[6] = d[4] ^ d[8] ^ d[11] ^ d[12] ^ d[14] ^ d[15];
assign cb[7] = d[5] ^ d[9] ^ d[11] ^ d[13] ^ d[14] ^ d[15];

endmodule
```

Fig. 6: This diagram shows the Hsiao Code's encoder in Verilog code produced by our code generator.

Our code generator, a C++ program only requires the bits from columns 0 to 15 from the H-matrix in to create our encoder in Verilog code (Fig. 6)

## III. Design of a (24, 8) Cyclic Redundancy Code's Encoder Matrix

Cyclic Redundancy Codes (CRCs) are conventional methods for error detection in networking and other applications. For networking, the interest is the Hamming Distance (HD), which is the least possible number of bit inversions in a message that can create an error undetectable by that message's CRC-based Frame Check Sequence. For example, if a CRC polynomial has an HD of 6 for a given network, all possible combinations of 1- to 5-bit errors (where a bit error is an inversion of a bit value) will be detected. At least one combination of 6 bits that, when corrupted within a message, is undetectable by that CRC.

We design our next EDAC based on a CRC polynomial to leverage the CRC detection capability. To increase the information rate, we compute the 16-bit message with an 8-bit CRC generator. To reduce the bit-weight in the encoder matrix, we set the CRC's seed value to zero. When a two-input XOR gate has an input equal to a logical zero, its output value is equal to the other input's value. Based on this Boolean state, we eliminate the XOR gates required for the seed value, thus reducing the bit-weight of our CRC matrix. We developed a CRC-based [9] encoder matrix generator in C++ to generate our EDAC encoder matrix with these criteria.

Fig. 7 and 8 show the statistical information of the CRC encoder matrices created by our generator. Due to space limitations, we only show 13 out of the 50 CRC encoder matrices generated. Fig. 7 shows each encoder's capability in detecting three- and four-bit-errors. Fig. 8 shows the minimum, the average, and the maximum bit-weights of the matrices' rows (not including the eight bits in the check-bit columns). Based on these statistics, we choose the polynomial, x^8 + x^3 + x^2 + 1, that provides the best average result in Fig. 7, and Fig. 8. This is the third polynomial in both Figures.

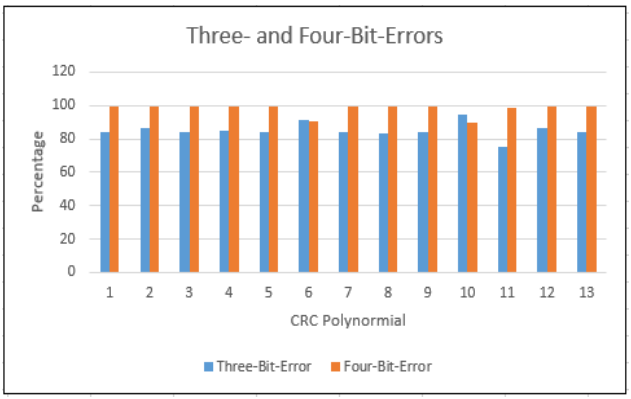


Fig. 7: This diagram shows the CRC's EDAC three- and four-bit-error detection capability.

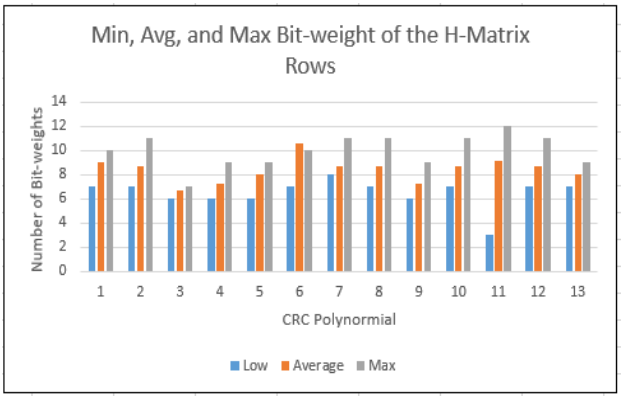


Fig. 8: This diagram shows the CRC's H-Matrix rows minimum, average, and the maximum bit-weights.

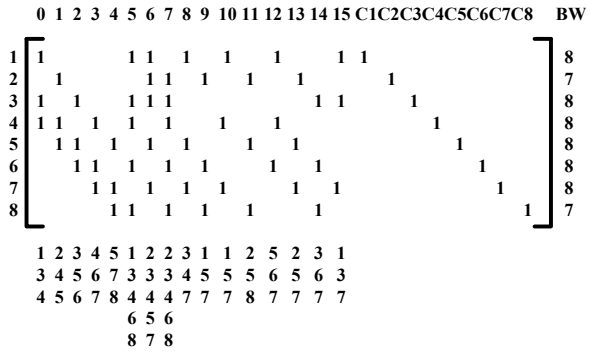


Fig. 9: This diagram shows the CRC's EDAC encoder matrix generated by our CRC matrix generator with polynomial= x^8 + x^3 + x^2 + 1.

Fig. 9 shows our generated H-matrix from our CRC generator. The generated CRC encoder matrix complies with the first rule of Hsiao Code [1]: "Every column has an odd number of 1's; i.e., all column vectors are of odd weights". We now have both a (24, 16) Hsiao Code and (24, 16) CRC encoder matrices. In the next section, we show the generated designs of our interleaved EDACs using both the Hsiao Code and CRC EDACs capable of detecting various bit-errors, as shown Fig. 12.

## IV. Design an Interleaved (48, 32) EDAC

Fig. 10 shows the architecture of our interleaved EDAC encoder design. Our EDAC uses two identical single error correction and double error detection (SEC-DED) EDACs configured as the even and odd encoders, syndromes, and decoders. For example, two Hsiao Code encoder matrices generated in Section II, or two CRC encoder matrices generated in Section III can be used in Fig. 10 as the two identical encoders. This configuration improves the error correction and detection rates. For example, if a double-bit-error has one error occurring on an even-bit, and the other on an odd-bit of a codeword, this error can be corrected. Fig. 11 shows the Verilog code of the interleaved even and odd encoders, an equivalent to the diagram shown in Fig. 10.

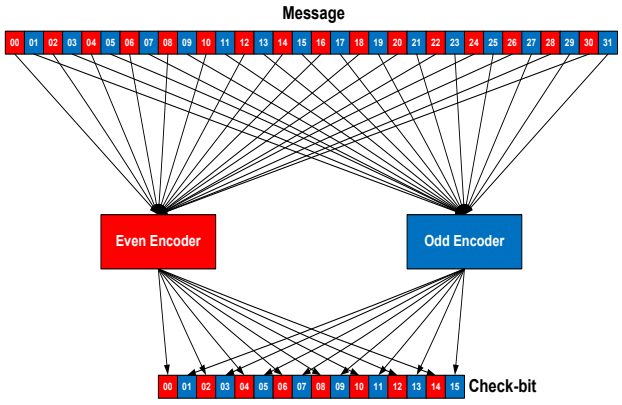


Fig. 10: Our Interleaved Double-Adjacent-Error-Correction EDAC's encoder connection. The Even Encoder encodes the message's even bits, and the Odd Encoder encodes the message's odd bits. The check-bits from both encoders are stored in an interleaved fashion to improve the error detection and correction capabilities of our EDACs.

```
//----------------------------------------------------------------------
// Interleave EDACENCODE
//----------------------------------------------------------------------
module I_ENCODE(
        input  wire [31:0]   d,
        output wire [15:0]   cb
);

    wire [15:0] eD = {d[30],d[28],d[26],d[24],d[22],d[20],d[18],d[16],
                      d[14],d[12],d[10],d[8],d[6],d[4],d[2],d[0]};
    wire [7:0]  eCB;

    wire [15:0] oD  = {d[31],d[29],d[27],d[25],d[23],d[21],d[19],d[17],
                       d[15],d[13],d[11],d[9],d[7],d[5],d[3],d[1]};
    wire [7:0]  oCB;

    assign cb[15:0] = {oCB[7],eCB[7],oCB[6],eCB[6],oCB[5],eCB[5],oCB[4],eCB[4],
                       oCB[3],eCB[3],oCB[2],eCB[2],oCB[1],eCB[1],oCB[0],eCB[0]};

    EDACENCODE Even(
        .d      (eD),
        .cb     (eCB)
    );

    EDACENCODE Odd(
        .d      (oD),
        .cb     (oCB)
    );

endmodule
```

Fig. 11: The interleaved encoders in Verilog code is an equivalent to the diagram shown in Fig. 10.

Fig. 12 shows the error detection and correction metrics of our EDAC designs (Hsiao Code and CRC). Our EDACs can correct all single-bit-errors. For double-bit-errors, our EDACs can *correct* all the double-adjacent-bit-errors. Additionally, double-bit-errors have one error occurring on an even-bit and the other on an odd-bit of a codeword (see Fig. 12, "Double-bit-error 100% correctable) can be *corrected*. Since our EDAC is based on single-error-correction and double-error-detection, our design can also *detect* all the remaining combinations of double-bit-errors (see Fig. 12, "Double-bit-error 100% Detectable"). All three-bit-errors that are detectable, i.e., when two errors occur on two odd-bits and one on an even-bit of a codeword, and vice versa, are detected by our EDAC. Our EDAC can also detect up to four-adjacent-bit-errors. For other detection rate capabilities, see Fig. 16.

| Type of Errors | Codeword | | Description |
|---|---|---|---|
| | Even Bits | Odd Bits | |
| Single-bit error | One error bit | | 100% Correctable |
| | | One error bit | |
| Double-bit-error | One error bit | One error bit | |
| | Two error bits | | 100% Detectable |
| | | Two error bits | |
| Three-bit-error | Two error bits | One error bit | |
| | One error bit | Two error bits | |
| | Three error bits | | (Figure 14) |
| | | Three error bits | |
| Four-bit-error | Two error bits | Two error bits | 100% Detectable |
| | One error bit | Three error bits | (Figure 14) |
| | Three error bits | One error bit | |
| | Four error bits | | |
| | | Four error bits | |

Fig. 12: The correctable and detectable errors of our EDAC designs.

## V. Codeword Storage Format and Algorithm

A hardware- or software-based EDAC provides methods to ensure data reliability in memory devices (see Fig. 13) [10]. The cost for more reliable data is lower information rate because an EDAC creates additional bits for error detection and correction. These extra bits are known as the check-bit. Depending on the EDAC implementation, the number of bits used for the check-bit varies. In our research, we use a 16-bit check-bit (or two 8-bit check-bits).

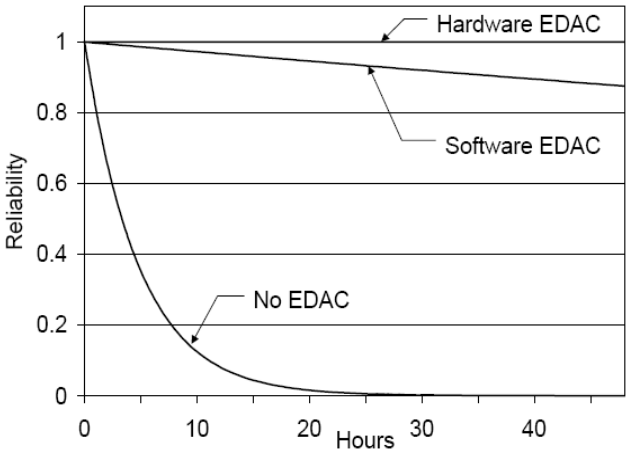


Fig. 13: The data reliability over time in memory devices with hardware EDAC, software EDAC, and no EDAC [10].

When the EDAC corrects an error, the fault-tolerant memory controller writes the corrected codeword (message and check-bit) back to the same memory location. This process is known as memory scrubbing. A 48-bit codeword would need six 8-bit, three 16-bit, or one 32-bit and one 16-bit memory device to store the message (code or data) and the check-bit. In a scenario where a satellite's board space is scarce, these memory devices' configurations are not desirable. Instead, we recommend using an 8-bit or a 16-bit memory device to store the 48-bit codewords, as shown in the following.

Our approach breaks down the 48-bit codeword into six bytes, with four bytes for the message, and two bytes for the

check-bit in an 8-bit memory device configuration. Fig. 14 shows the memory space allocation for the messages and check-bits, at a 2:1 ratio. For the 16-bit memory device, we break down the 48-bit codeword into three 16-bit words, two 16-bit words for the message, and one 16-bit word for the check-bit. Fig. 15 shows the storage format for the messages and check-bits at a 2:1 ratio.

| Address | Data (8-bit) | Assignment | Size |
|---|---|---|---|
| FFFF_FFFF | | Check-bit | 33.33% for Check-bits |
| FFFF_FFFE | | Check-bit | |
| FFFF_FFFD | | Check-bit | |
| FFFF_FFFC | | Check-bit | |
| FFFF_FFFB | | Check-bit | |
| FFFF_FFFA | | Check-bit | |
| FFFF_FFF9 | | Check-bit | |
| FFFF_FFF8 | | Check-bit | |
| … | … | … | |
| … | … | … | 66.66% for Message |
| 0000_000F | | Message | |
| 0000_000E | | Message | |
| 0000_000D | | Message | |
| 0000_000C | | Message | |
| 0000_000B | | Message | |
| 0000_000A | | Message | |
| 0000_0009 | | Message | |
| 0000_0008 | | Message | |
| 0000_0007 | | Message | |
| 0000_0006 | | Message | |
| 0000_0005 | | Message | |
| 0000_0004 | | Message | |
| 0000_0003 | | Message | |
| 0000_0002 | | Message | |
| 0000_0001 | | Message | |
| 0000_0000 | | Message | |

Fig. 14: The format for storing codewords in an 8-bit wide memory device.

| Address | Data (16-bits) | Assignment | Size |
|---|---|---|---|
| FFFF_FFFE | | Check-bit | 33.33% for Check-bits |
| FFFF_FFFC | | Check-bit | |
| FFFF_FFFA | | Check-bit | |
| FFFF_FFF8 | | Check-bit | |
| … | … | … | |
| … | … | … | 66.66% for Message |
| 0000_000E | | Message | |
| 0000_000C | | Message | |
| 0000_000A | | Message | |
| 0000_0008 | | Message | |
| 0000_0006 | | Message | |
| 0000_0004 | | Message | |
| 0000_0002 | | Message | |
| 0000_0000 | | Message | |

Fig. 15: The format for storing codewords in a 16-bit wide memory device.

### *A. An Algorithm for the 48-bit Codeword in an 8-bit or a 16-bit Memory Device.*

The algorithm for storing the message and check-bit in an 8-bit (see Fig. 14) or a 16-bit (see Fig. 15) memory device works as follows:

1. The EDAC computes the 16-bit check-bit from the 32-bit message (data or code). Fig. 16
2. The memory controller writes the 32-bit message as follows:
   - 8-bit memory device— writes the 32-bit message in four bytes.
   - 16-bit memory device—writes the 32-bit message in two 16-bit words

   For this example, the writing of the message starts from address 0000_0000, as shown in Fig. 14, and Fig. 15, bottom blue color).
3. After writing the fourth byte, the address is 0000_0003h.
4. The controller takes the last address (0000_0003h), shifts the address value to the right by one bit, and inverts the address. The address after inversion is the write address for the check-bit. In this case, the address is FFFF_FFFEh, as shown Fig. 14 and Fig. 15, top blue color.
5. The memory controller writes the lower byte of the 16-bit check-bit to address FFFF_FFFEh and the higher byte to FFFF_FFFFh.

## VI. Validation

We used Verilog HDL and ModelSim to create, simulate, and test our designs. Fig. 16 shows the test results for each EDAC—Hsiao Code, and CRC. We also compiled the different designs using Intel Quartus 18.1.0 Build 625 09/12/2018 SJ Lite Edition (Cyclone V, 5CGXFC7C7F23C8). All four designs in Fig. 16 require less than 1/ 56,480 (< 1 %) Logic utilization (in ALMs).

| EDAC Algorithm | Hsiao Code | | CRC | |
|---|---|---|---|---|
| Row Bit-weight (min, avg, max) | (6, 6, 6) | | (6, 6.75,7) | |
| (n, k) | (24, 16) | (24, 16) x 2 | (24, 16) | (24, 16) x 2 |
| Single Bit Correction | 100% | 100% | 100% | 100% |
| • Total Test per Codeword | 24 | 48 | 24 | 48 |
| | | | | |
| Double Bit Detection | 100% | 100% | 100% | 100% |
| • Total Test per Codeword | 276 | 1128 | 276 | 1128 |
| • Corrected | 0 | 576 | 0 | 576 |
| • Detected | 276 | 552 | 276 | 552 |
| • Faulty Correction | 0 | 0 | 0 | 0 |
| | | | | |
| Three Bit Detection | 82.21% | 95.84% | 83.99% | 96.25% |
| • Total Test per Codeword | 2024 | 17296 | 2024 | 17296 |
| • Faulty Correction | 360 | 720 | 324 | 648 |
| | | | | |
| Four Bit Detection | 99.15% | 91.03% | 99.24% | 91.92% |
| • Total Test per Codeword | 10626 | 194580 | 10626 | 194580 |
| • Faulty Correction | 90 | 17460 | 81 | 15714 |

Fig. 16: EDACs test results.

For single-bit-error correction tests, we tested each design using random data with one-bit inversion (a total of 48 tests). For double/three/four-bit-error detection, the number of tests is C(48, r), with r=2, 3, and 4. This results in a total of 1128/17296/194580 tests for the double/three/four-bit-error detection.

The four-bit-error detection rate drops when we combine two identical (24, 16) EDACs to form our interleaved EDAC. This reduction in the four-bit-error detection rate is because of

the (24, 16) EDAC's low three-bit-error detection rate, also shown in Fig. 16. For example, the syndrome sets the error-flag if there is an error. When an even-bit-error is detected, the even error-flag is set; similarly, when an odd-bit-error is detected, the odd error-flag is set. However, the syndrome logic does not know what type of error has occurred. In the case of a four-bit-error, it could be that a three-bit-error has occurred on some even-bits, and a one-bit-error on an odd-bit (or vice versa). In this case, the decoder begins to correct the error based on the set error-flag, not knowing a three-bit-error has occurred. Since the three-bit-error detection rate is low, an erroneous correction could happen. Therefore, having an EDAC with a higher detection rate for three- and four-bit-error is essential to reducing incorrect code or data correction by the EDAC.

Our CRC-based EDAC performs better in the detection rate for the three- and four-bit-errors than the Hsiao Code. However, one advantage of the Hsiao Code's EDAC in this work is that it has less bit-weight per row (Fig. 5). This parameter might be a key consideration for a high-performance system.

## VII. Related Work

Babitha and Divya's single error correction, double error detection, double adjacent error correction, and triple adjacent error detection (SEC-DED-DAEC-TAED) EDAC used a modified Hamming Code. The state of the EDAC could not be verified since no validation results were presented [11]. Jun and Lee proposed an SEC-DED-DAEC EDAC [12] that requires the least redundancy bits relative to the other EDACs in their work. One limitation of having reduced redundancy bits is that the percentage of erroneous corrections is very high for the three- and four-bit-errors—there was no data to support the three- and four-bit-error detection capabilities.

Hsiao proposed a class of optimal minimum odd-weighted-column SEC-DED code known as the Hsiao Code [1] in the 1970s to improve the speed, cost, and reliability of the decoding logic. Although several proposed SEC-DED EDACs [2] exist in the literature, the Hsiao Code remains the most optimized SEC-DED EDAC.

Others proposed various classes of SEC-DED codes [3], [4] that can detect any number of errors in a single byte. These codes are known as single-error-correction double-error-detection single-byte-error-detection (SEC-DED-SBD) codes. An EDAC with a double-error-correction and triple-error-detection code may be used, at the cost of higher overhead of check-bits and more sophisticated hardware to implement the error correction and detection [5], [6], [7]. This type of implementation may result in a slower EDAC that is not suitable in a high-performance system.

## VIII. Discussion and Conclusion

Our two interleaved Double-Adjacent-Error-Correction (DAEC) EDACs can correct single-bit-errors, double-adjacent-bit-errors, and double-bit-errors that one error occurs on an even-bit and the other on an odd-bit of a codeword. For error detection, our EDACs can detect up to four-adjacent-bit-errors. The detection rates of three-bit-errors are also improved compared to the individual (24,16) EDACs. Both the Hsiao Code and CRC-based EDAC approaches show promising results with minimum hardware requirements. Since the maximum bit-weight per row is low (six for Hsiao Code and seven for CRC), our EDACs are suitable in a high-performance computing system.

We also show how to manage the 48-bit codeword when board space is scarce. The storage format and algorithm allow us to support this class of EDACs with minimum hardware support. One caveat to these storage implementations is that the memory size must be of Width*2^N bits where Width=8 or 16. Since commercial memory devices come in this format, our codeword-storage designs work well with such devices.

## References


[1] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," in IBM Journal of Research and Development, vol. 14, no. 4, pp. 395-401, July 1970), DOI: 10.1147/rd.144.0395

[2] R. W. Hamming, "Error detecting and error correcting codes," in The Bell System Technical Journal, vol. 29, no. 2, pp. 147-160, April 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[3] Reddy, S.M., "A Class of Linear Codes for Error Control in Byte-per-Package Organized Memory Systems", IEEE Trans. On Computers, Vol. C-27, pp. 455-458, May. 1978.

[4] Chen, C. L., "Error Correcting Codes with Byte Error Detection Capability", IEEE Trans. On Computers, Vol. 32, pp. 615-621, May 1983.

[5] Lin, S., and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice-Hall, 1983.

[6] Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968.

[7] Lala, P. K., "An Adaptive Double Error Correction Scheme for Semiconductor Memory Systems," Digital Processes, Vol.4, pp 237-243, 1978.

[8] Hiller, Caleb, Balyan, Vipin, and Zhang, Yagang, "Error Detection and Correction On-Board Nanosatellites Using Hamming Codes," Hindawi, Journal of Electrical and Computer Engineering, Volume 2019, Article ID 3905094, 15 pages. https://doi.org/10.1155/2019/3905094.

[9] S. Babaie, A. K. Zadeh, S. H. Es-hagi and N. J. Navimipour, "Double Bits Error Correction Using CRC Method," 2009 Fifth International Conference on Semantics, Knowledge and Grid, Zhuhai, 2009, pp. 254-257. DOI: 10.1109/SKG.2009.77.

[10] P. P. Shirvani, N. R. Saxena and E. J. McCluskey, "Software-implemented EDAC protection against SEUs," in IEEE Transactions on Reliability, vol. 49, no. 3, pp. 273-284, Sept 2000. DOI: 10.1109/24.914544.

[11] Babitha Antony, Divya S, "Modified Hamming Codes with Double Adjacent Error Correction along with Enhanced Adjacent Error Detection," International Journal of Innovative Research in Computer and Communication Engineering, IJIRCCE , August 2015, DOI: 10.15680/IJIRCCE.2015. 0308056

[12] Ho-yoon Jun and Yong-surl Lee, "Single error correction, double error detection and double adjacent error correction with no mis-correction code," IEICE Electronics Express, Vol.10, No.20, 1- 2013, DOI: 10.1587/elex.10.20130743.