

A Low Cost Code-Based Methodology for Tolerating Multiple Bit Upsets in Memories

Avijit Dutta and Nur A. Touba

**Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas at Austin**

ELECTRICAL & COMPUTER ENGINEERING



Outline

Motivation

Previous work

- Interleaving along with SEC-DED
- Code based approaches
- Circuit level techniques with coding

Proposed approach

- Selective cycle avoidance based SEC-DED-DAEC code

Experimental results

Motivation

☞ Single Event Upsets (SEUs)

- Can result in multiple bit upsets (MBUs)
 - [Ibe et. al. 04], [Maiz 03]

☞ Need techniques to protect memories against MBUs

Multiple Bit Upsets

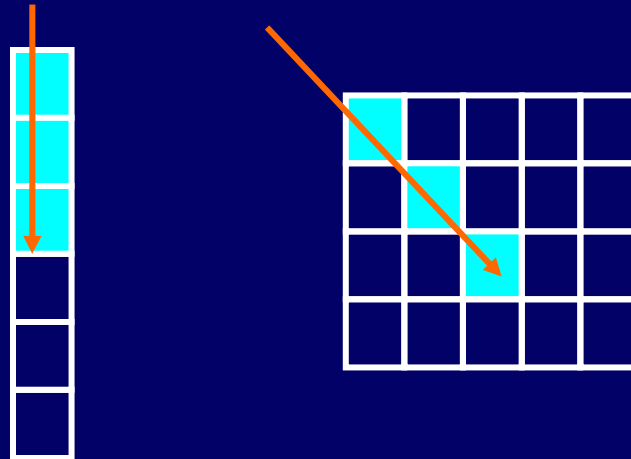
☞ Affects cells in a row

- Causes multiple bit upsets per logical word



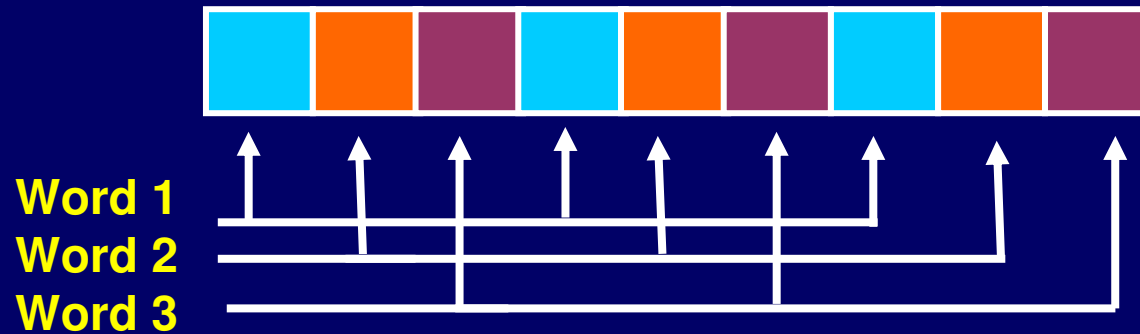
☞ Affects cells in a column / along the diagonal

- Causes single bit upsets per logical word



Previous Work

- 👉 Uses SEC-DED code along with bit level interleaving
 - K-way interleaving
 - Reduces k-bit burst error into k single-bit errors per logical word



- SEC-DED code
 - Corrects single bit error
 - Detects random double bit errors per logical word

Interleaving / SEC-DED

Advantages

- Low check bit overhead
- Low encoder, syndrome decoder, and correction logic overhead
- Fast encoding and decoding

Disadvantages

- Interleaving may negatively impact
 - Floorplanning
 - Increased area and delay
 - Column MUX
 - Power dissipation
 - Unnecessary read access for words in row
- Interleaving not feasible in small memories
 - CAMs, register files, router buffer, etc.

Other Code Based Approaches

☞ SEC-DED-SBD [Reddy 78, Chen 83]

☞ DEC-TED [Lin 83, Berlekamp 68, Lala 78]

- Higher check bit overhead
- Higher encoding, decoding, correction overhead

☞ SBC-DBD [Berlekamp 68, Reed 60, Wolf 69, Bossen 70]

- Works at higher order Galois Field
- Higher check bit overhead
 - Increases memory size
- Complex encoding, decoding
 - Multiplication involves several table lookups

Other Code Based Approaches

- ☞ Reed-Solomon (RS) code, Bose-Chaudhuri-Hocquenghem (BCH) code, extended Hamming code
 - Lower percentage check bit overhead
 - Works at block level (multiple words at time)
 - Complex encoding, decoding
 - Requires several table lookups
 - Complex algebraic decoder
 - Several cycles to correct first error
 - Large latency and low speed
 - Higher encoding, decoding, correction overhead

Circuit Level Techniques

- ☞ **Combines coding with circuit level techniques**
 - **Asynchronous built-in current sensor (BICS) on vertical power line along with parity check [Vergas 94, Calin 95]**
 - **BICS with SEC-DED [Gill 05]**

Idea

👉 Observation

- Adjacent double bit errors much more likely than other multiple bit errors [Maiz 03]

👉 Single-error-correcting, double-error-detecting, double-adjacent-error-correcting code (SEC-DED-DAEC)

- Can be used instead of or in addition to bit interleaving
- Provides greater flexibility for optimizing memory
- For fixed depth of interleaving, larger physical distance between cells in error can be tolerated
- For fixed physical separation, depth of interleaving can be reduced
- Can be used for small memories (CAM, register files, router buffer) where interleaving not feasible

Previous SEC-DED-DAEC Codes

- ➡ **Developed for communication applications**
- ➡ **Abramson code [Abm 59]**
 - **Inefficient encoding and decoding**
 - **One check bit dedicated to differentiate single errors from double errors**
 - **Computes parity of entire message**
 - **Incurs lot of encoding and decoding delay**
 - **Code decoded using finite state machine**
 - **Increased latency**
- ➡ **Extension of Abramson code to higher order Galois Field [Elspas 60]**
- ➡ **Extension of Abramson code for arithmetic operations [Bernstein 63]**

Proposed Code

New class of SEC-DED-DAEC codes

- Nearly identical encoding and decoding overhead / latency as commonly used SEC-DED code
- Same number of check bits as SEC-DED code
- Code constructed by selectively avoiding certain types of linear dependencies (cycles) in parity check matrix (H-matrix)
- Tries to minimize mis-correction (non-adjacent double error mistaken as adjacent double error) probability

Coding Theory Background

👉 Binary systematic linear block code

- Binary (n, k) code
- k -dimensional subspace of n -dimensional vector space
- Contains $r = (n-k)$ check bits
- $(r \times n)$ parity check matrix (H-matrix) completely defines code
- C is a code word if and only if $H.C^T = 0$
- Syndrome vector $[S]$
 - $S = H.V_f = H.(V+E) = H.V + H.E = H.E$
 - All zero syndrome corresponds to error free case / aliasing

Syndrome Generation

👉 An example of systematic (8,4) SEC-DED code

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M = 1011$$

$$C = M.G = 10110100$$

👉 Error free syndrome

$$S = H.C^T = [0000]^T$$

Syndrome Generation

👉 An example of systematic (8,4) SEC-DED code

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M = 1011$$

$$C = M.G = [10110100]$$

$$C_f = [10100100]$$

$$E = [00010000]$$

👉 Single error correction

$$S = H.C_f^T = [1110]^T \rightarrow \text{error in 4-th bit}$$

Syndrome Generation

➤ An example of systematic (8,4) SEC-DED code

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M = 1011$$

$$C = M.G = [10110100]$$

$$C_f = [10011100]$$

$$E = [00101000]$$

➤ Double error detection

$$S = H.C_f^T \rightarrow [0111]^T \oplus [1000]^T \rightarrow [1111]^T$$

Proposed Code

Properties of proposed code

- All single bit errors corrected
- All double bit errors detected
- All adjacent double bit errors corrected
- Reduced miscorrection probability

Constraints on H-matrix

☞ Constraints on H-matrix

- No all zero column
 - Differentiates error free case from erroneous case
- All columns distinct
 - Uniquely identifies and hence corrects single bit errors
- No linear dependency involving 3 or less columns
 - No 2-cycle or 3-cycle allowed
 - Make double error syndromes different from single error syndromes
- No linear dependency involving columns C_i, C_j, C_k, C_m where $m > k > j > i$ such that $j = i + 1$ and $m = k + 1$ [Forbidden 4-cycles]
 - All adjacent double error syndromes different
- Tries to minimize 4-cycles involving C_i, C_j, C_k, C_m where $m > k > j > i$ such that $j = i + 1$ or $k = j + 1$ or $m = k + 1$

Code Design

- ➡ **Input: code length (n), maxbacktrack, maxpermute**
- ➡ **Output: H-matrix**
- ➡ **Start with set of available columns**
 - **Store all 1-weight columns, followed by 3-weight columns, followed by 5-weight column and so on up to largest weight column**
- ➡ **Choose column from available column pool**
 - **Check for 3-cycle, forbidden 4-cycle**
 - **If multiple choices, choose one that minimizes number of bad 4 cycles**
 - **If forbidden cycles not found, add column to H-matrix**
- ➡ **Else if no satisfying column found then backtrack**
 - **Undo last choice of column**
 - **Repeat all steps**
- ➡ **Once satisfying H-matrix found, perform limited number of column permutations to minimize bad 4-cycles without introducing any forbidden cycles**

Code Design

👉 H-matrix for proposed (22,16) SEC-DED-DAEC code

| Available columns | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |



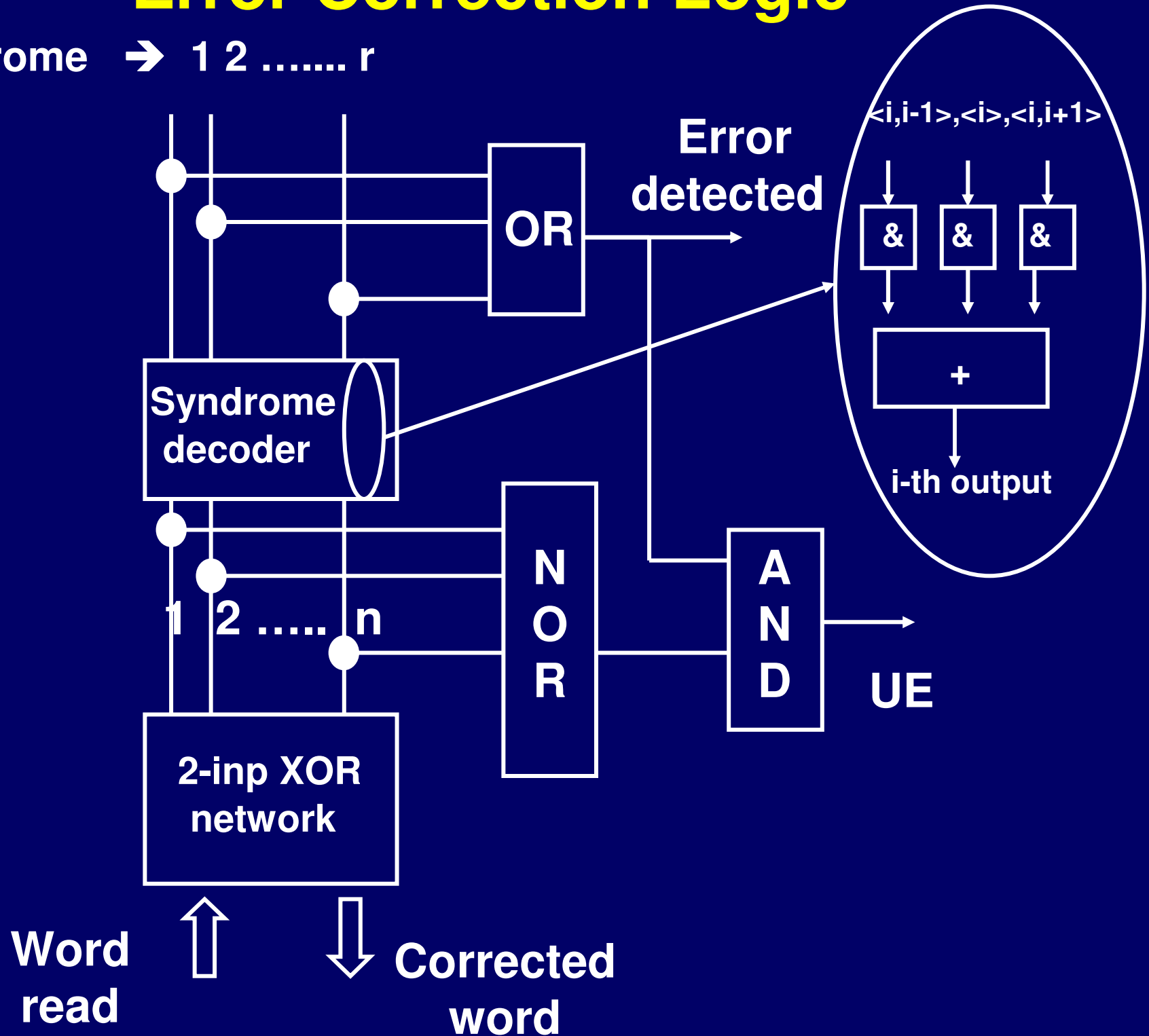
| H-matrix | | | | | | | | | | | | | | | | | | | | | |
|----------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | 0 | 1 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | 0 | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 1 | 0 |
| | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 1 |

Encoding / Decoding

- ➡ Use XOR network corresponding to G-matrix to encode message
- ➡ Generate syndrome using XOR network corresponding to H-matrix
- ➡ If all zero syndrome
 - No error detected else one or more errors
- ➡ If syndrome matches any column of H-matrix
 - Single error detected
 - Error position given by column position
 - Corresponding bit flipped for correction
- ➡ Else if syndrome matches any of $(n-1)$ adjacent double error syndromes
 - Double adjacent error detected
 - Corresponding bits flipped for error correction
- ➡ Else uncorrectable error occurred

Error Correction Logic

r-bit syndrome \rightarrow 1 2 r



Experimental Results

| (n, k) | codes | 2-inp XOR gates | Max logic depth | Forbidden 4-cyls (4FC) | Bad 4-cyls (4BC) | Total 4-cyls |
|---------|---|-----------------|-----------------|------------------------|------------------|--------------|
| (22,16) | SEC-DED (IBM system/3) | 48 | 4 | 13 | 122 | 252 |
| | Hsiao code [Hsiao 70] | 48 | 4 | 8 | 120 | 252 |
| | SEC-DED- DAEC in [Abramson 59] | 70 | 5 | 0 | 128 | 252 |
| | Proposed SEC-DED- DAEC | 48 | 4 | 0 | 118 | 251 |

Experimental Results

| (n, k) | codes | 2-inp XOR gates | Max logic depth | Forbidden 4-cyls (4FC) | Bad 4-cyls (4BC) | Total 4-cyls |
|----------------|--|--------------------------------|--------------------------------|---------------------------------------|---------------------------------|-------------------------|
| (39,32) | SEC-DED (IBM system/3) | 96 | 4 | 82 | 254 | 776 |
| | Hsiao code [Hsiao 70] | 96 | 4 | 23 | 425 | 1363 |
| | SEC-DED-DAEC in [Abramson 59] | 132 | 6 | 0 | 386 | 1343 |
| | Proposed SEC-DED-DAEC | 96 | 4 | 0 | 379 | 1363 |

Experimental Results

| (n, k) | codes | 2-inp XOR gates | Max logic depth | Forbidden 4-cyls (4FC) | Bad 4-cyls (4BC) | Total 4-cyls |
|---------|--------------------------------------|-----------------|-----------------|------------------------|------------------|--------------|
| (72,64) | SEC-DED (IBM system/3) | 256 | 6 | 230 | 1292 | 8912 |
| | Hsiao code [Hsiao 70] | 208 | 5 | 122 | 1399 | 8392 |
| | SEC-DED- DAEC in [Abramson 59] | 296 | 7 | 0 | 1335 | 8194 |
| | Proposed SEC-DED- DAEC | 224 | 5 | 0 | 1316 | 8298 |

Conclusion

- 👉 **Proposed SEC-DED-DAEC code**
 - **Uses same number of check bits as SEC-DED code**
 - **Similar encoding and decoding overhead / latency**
- 👉 **Can correct all single bit errors and double adjacent bit errors resulting from single event upset (SEU)**
- 👉 **Provides additional flexibility in memory error protection, memory layout optimization**
- 👉 **Can be used instead of or in addition to interleaving to protect memory from multiple bit upsets caused by SEU**