MVP ECC : Manufacturing process Variation aware unequal Protection ECC for memory reliability

Seungyeob Lee
Department of Semiconductor and Display Engineering
Sungkyunkwan University
Suwon, Korea

Joon-Sung Yang
Department of Semiconductor Systems Engineering
Sungkyunkwan University
Suwon, Korea

***Abstract*—With a development of process technology, a memory density has been increased. However, a smaller feature size makes the memory susceptible to soft errors. For reliability enhancement, ECC with single bit error correction and double bit error detection is widely used. As multiple bit cell upset become dominant, there is a need for stronger ECC. ECC such as RS or BCH code requires significantly large overhead and longer latency. To overcome the problem, this paper introduces an unequal protection ECC assigning stronger level of protection to weak memory cells and normal level to normal cells. Information from manufacturing characterization test is utilized to identify weak memory cells with low design margins. Instead of equally treating all memory cells, the proposed ECC focuses more on the weak cells since they are more susceptible to soft errors. Compared to conventional ECCs, experimental results show that the proposed ECC considerably enhances memory reliability with the same code length.**

Keywords—Manufacturing Process Variation; Unequal Protection Code; Error Correction Code

I. INTRODUCTION

As a semiconductor technology develops, a device has been scaled down to sub-20 *nm* region. This achieves a high memory density, low power and high performance. To improve the system reliability, an error correction code (ECC) is widely used. As ECC encodes data with additional check-bits, some errors can be corrected or detected. However, with the reduced feature size, memory reliability faces serious problems. [1] shows that soft-error rates (SERs) per device would increase by 7.1-times and multi-cell upset (MCU) rate by 6.7-times from 130 *nm* to 22 *nm* technology. A single error correction double error detection (SEC-DED) code which is one of the most widely used ECCs, cannot tolerate MCUs. The MCU problem could be relieved by interleaving data bits [2]. As the interleaving stores data bits in physically separated memory cells, the error caused by MCU can be changed to multiple single-cell upset (SCU) errors and the SEC-DED code can handle these SCU errors. However, using interleaving technique introduces an area, power and delay overhead. The other ECCs such as Bose-Chaudhuri-Hocquenghem (BCH) code and Reed-Solomon (RS) code that could correct symbol errors, has been suggested to handle MCUs. Although these codes are highly resistant to MCUs, they require significantly large check bits, complex decoder and longer latency. To relieve the overhead, a single error correction double error detection double adjacent error correction (SEC-DED-DAEC) code [3] has been presented. The SEC-DED-DAEC code can deal with double adjacent error with the same code length as SEC-DED code.

Unequal error protection (UEP) codes [4] are ECCs which protect data bits with different protection level. As UEP ECC, in [5], a single error correction double adjacent error detection selective double adjacent error correction (SEC-DAED-SDAEC) code is introduced to protect data packets in Network-on-Chip (NOC). In data packet, since the header is considered more importantly in data packets, it is protected by SEC-DAEC code while the other portion in the same data packet is covered by SEC-DAED code.

In this paper, a new UEP code is proposed considering manufacturing process variations. The proposed code consists of a single error correction double error detection triple adjacent error correction double adjacent error correction (SEC-DED-TAEC-DAEC) and SEC-DED code. This code focuses on physical characteristic of memory cells. Due to manufacturing process variations, the characteristic of memory cells would be different from each other. We refer the memory cell with lower design margins as *weak cell* and the others as *normal cell*. The *weak cell* and *normal cell* pass manufacturing test, however, the *weak cell* would fail first if the operating voltage is applied lower than its normal operation voltage since they have lower margin. In this manner, a manufacturing characterization test can identify *weak cells*. Because data stored in *weak cells* would have low reliability, the proposed method introduces unequal protection capability by assigning SEC-DED-TAEC-DAEC to *weak cells* and SEC-DED to *normal cells*.

This paper is organized as follows. Section II briefly presents a linear block code. The proposed ECC architecture and UEP code generation is presented in Section III. Section IV analyzes hardware complexity and miscorrection probability and a conclusion is given in Section V.

II. LINEAR BLOCK CODES

Conventional ECCs such as a SEC-DED or SEC-DED-DAEC are binary linear block codes. These codes are generally represented as (n, k) block codes. An (n, k) code means that this code is n-bit long and consists of k-bit message and r-bit (r=n-k) check bits. Linear block codes can be completely defined by

978-3-9815370-8-6/17/$31.00

($r \times n$) parity check matrix H and ($k \times n$) generator matrix G. The H-matrix for a decoding operation can be denoted as

$$H = \left[P_{rxk} : I_{rxr}\right]$$

The G-matrix is used for the encoding operation and represented as

$$G = \left[I_{kxk} : P^T_{kxr}\right]$$

where I is the identity matrix and P^T is the transpose of the P in H-matrix. The encoding operation that takes k-bit message and makes n-bit codeword is done by matrix multiplication of the message and G-matrix. The operation can be expressed as

$$c = mG$$

, where m is a message and c represents a codeword. The codeword is composed of a message and check bit. The first k-bit of the codeword is the message and the following r-bit is the check bit. Similarly, the decoding process is given as

$$S = cH^T$$

, where H^T is the transpose of the H-matrix and S is a syndrome. The syndrome is a result of matrix multiplication of the codeword and H^T, and has k-bit. Since the syndrome contains information on errors, the corrupted bits in the codeword could be detected or corrected using the syndrome. For example, when there are no errors in the codeword, the syndrome is all zero. In a single error case, the syndrome is the same as one of the columns in H-matrix. Hence, if all columns in H-matrix are distinct, all single errors can be corrected. In a double error case, the syndrome is the same as the result of modulo two summation of any two columns in H-matrix that corresponds to corrupted bits in the codeword. To generate a double error correction code, all results of modulo two summation of any two columns in H-matrix and all single columns have to be different from each other. However, the correction of all double bit errors requires high latency, complex decoding logic and more check bits. To avoid these problems, a double adjacent error correction (DAEC) code is more preferred. When DAEC code is generated, H-matrix generally consists of odd-weight columns, hence, the result of modulo two summation of any two columns is even-weighted. The weight of a column is the number of ones in the column. If all results of modulo two summation of adjacent two columns in H-matrix are totally distinct, this H-matrix can serve as DAEC.

III. Proposed Unequal Protection ECC

In this section, a characteristic of the proposed ECC, code design procedure and its architecture are presented.

A. Proposed ECC

The proposed code is a two-level UEP code and it consists of SEC-DED-TAEC-DAEC and SEC-DED code. In the proposed scheme, the (n, k) block code is divided into two partitions which are a *weak region* and *normal region*. The *weak region* is the first half of the data word ($k/2$-bits) and the *normal region* is the rest. The SEC-DED-TAEC-DAEC and SEC-DED code handles the *weak region* and the *normal region* respectively. The proposed code has following characteristics and its correctable/detectable/uncorrectable errors are depicted in Fig. 1.

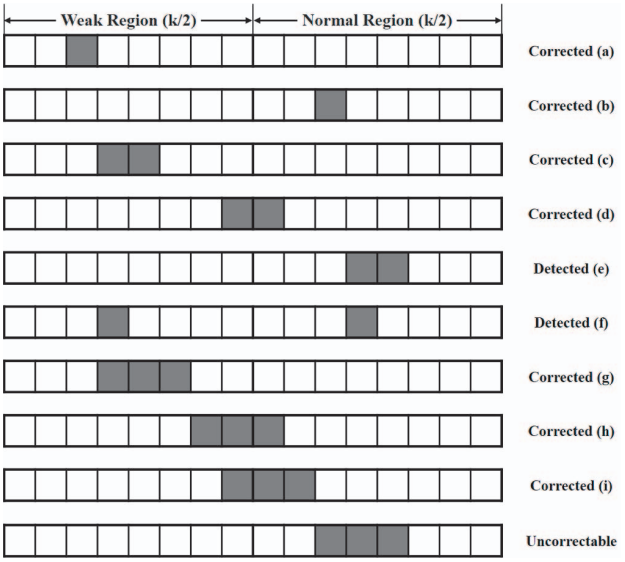


Fig.1. Correctable/Detectable/Uncorrectable Errors by Proposed ECC

1) All single bit errors can be corrected and all double bit errors can be detected (Fig. 1 (a), (b), (e), (f)).
2) All double adjacent bit errors in *weak region* can be corrected (Fig. 1 (c)).
3) All triple adjacent bit errors in *weak region* can be corrected (Fig. 1 (g)).
4) All double adjacent bit errors and triple adjacent bit errors which contain 1 or 2 bits in *weak region* can be corrected (Fig. 1 (d), (h), (i)).

To generate the proposed code, all columns in H-matrix must satisfy the following conditions.

1) All columns are non-zero.
2) All columns are distinct and have odd-weights.
3) All modulo two summation of two adjacent columns C_i, C_{i+1} ($i \leq k/2$ and $k/2$ is a *weak region* size) are distinct and nonzero.
4) All modulo two summation of three adjacent columns C_i, C_{i+1}, C_{i+2} ($i \leq k/2$ and $k/2$ is a *weak region* size) are distinct, nonzero and different from all single columns.

Condition 1 ensures that all single bit error syndromes are not same as the error-free syndrome.

Condition 2 allows single bit error correction and double bit error detection (SEC-DED). A codeword with a corrupted i^{th} bit generates a syndrome that is the same as an i^{th} column of its H-matrix. If all H-matrix columns are uniquely identified, all single errors are correctable. Additionally, as every H-matrix column is odd-weighted, all modulo two summations of any two columns in H-matrix are even-weighted. Therefore, when double bit errors occur in codeword, its syndrome is odd-weighted and it would be different from individual H-matrix columns (i.e., double bit errors are detectable).

Condition 3 provides DAEC for *weak region*. This condition ensures that all the double adjacent bit error syndromes in the *weak region* are unique. Hence, all the double adjacent bit errors in the *weak region* are correctable.

Condition 4 provides TAEC for *weak region*. This condition ensures that all the triple adjacent bit error syndromes in *weak region* are different from all other triple adjacent error syndromes and individual *H*-matrix columns. As triple adjacent bit error and single bit error syndromes have odd-weight, these syndromes should be identified to avoid miscorrection.

Input: *n, k, r, w (weak region length)*, backtrack
Output: *H*-matrix
avail_col = {All *b*-weight columns}
b is odd number ($1 < b \le r$)
avail_num = the number of columns in avail_col
*H*_num = the number of columns in *H*-matrix
Begin Generate *H*-matrix
1. Insert *I*-matrix into *H*-matrix ($(n\text{-}r+1)^{th} \sim n^{th}$ column).
2. *H*_num = *r*
3. Insert all modulo two summation of any two ***I***-matrix columns into DED_syndrome.
// Construct The Normal Region
for Current_col = *n-r* **to** *w+1*
  for *i*=1 **to** avail_num
    1. Insert all DED_syndromes between i^{th} column in avail_col and *H*-matrix columns into temp_DED.
    2. Count shared columns between temp_DED and DED_syndromes.
  1. Select the column that has maximum count
  2. Remove the selected column from avail_col
  3. avail_num--
  4. Insert the selected column into *H*-matrix[Current_col]
  5. *H*_num++, Current_col--
// Construct The Weak Region
While ((Current_col>0) && (backtrack<max_back))
for Current_col = *w* **to** 1
  for *i*=1 **to** avail_num
    1. Take i^{th} column in avail_col(=temp_col)
    2. Calculate DAEC_syndrome
    3. Check whether the DAEC_syndrome is unique
    if (this syndrome is unique)
      1. Insert temp_col into temp_DAEC
      2. DAEC_num++
  if (DAEC_num==0)
    1. backtrack++
    2. *H*_num--, Current_col++
    3. **Go to** "**for** *i*=1 **to** avail_num"
  else
    for i=*1* **to** DAEC_num
      1. Take i^{th} column in temp_DAEC(=temp_col)
      2. Calculate TAEC_syndrome
      3. Check whether the TAEC_syndrome is unique
      if (this syndrome is unique)
        1. Insert temp_col into temp_taec
        2. temp_num++
    if (TAEC_num==0)
      1. Backtrack++
      2. *H*_num--, Current_col++
      3. **Go to** "**for** *i*=1 **to** avail_num"
    else
      1. select minimum miscorrection probability column in temp_TAEC, insert *H*-matrix[Current_col]
      2. Remove the selected column from avail_col
      3. Update DAEC and TAEC_syndromes
      4. avail_num--, Current_col--, *H*_num++

Fig.2. *H*-matrix Construction Procedure

To improve code efficiency, two conditions are added. First, the number of *1s* in *H*-matrix needs to be minimized. It is related to decoding logic area, power and latency. Second one is to enhance code accuracy. Since some double bit error syndromes may alias with double adjacent bit error syndromes, the decoding logic could mistake some double nonadjacent bit errors for double adjacent bit errors. It causes a possibility of performing miscorrection. To lower the miscorrection probability, the syndromes shared by double nonadjacent bit error and double adjacent bit error should be reduced. It needs to be noted that these two conditions are optional.

B. Code Construction Procedure

The procedure to construct *H*-matrix for the proposed code is shown in Fig. 2. ($r \times n$) *H*-matrix needs *n* odd-weight columns and 2^{r-1} columns are available. Hence, there are $C_n^{2^{r-1}}$ possible ($r \times n$) *H*-matrices. As it is practical to consider all cases, a pseudo-exhaustive search procedure is used. The construction is performed in 3 stages. As all *H*-matrices should contain ($r \times r$) *I*-matrix, it is inserted in the first stage. The next stage is to generate a SEC-DED code for *normal region*. Since all columns are odd-weights, a SEC-DED condition is always guaranteed. Additionally, the number of DED_syndromes are minimized to give more flexibility for *weak region* construction. The last stage is to construct a SEC-DED-TAEC-DAEC code for *weak region*.

C. Reconfiguration Logic and Contorl Logic

The architecture for the proposed UEP code is illustrated in Fig. 3. It requires reconfiguration logic and control logic.

In manufacturing test, memory cell characterization test can identify the ones with lower margins. The memory cells with low margin, *weak cells*, pass the normal test, however, they would fail first when a lower operating voltage than the normal

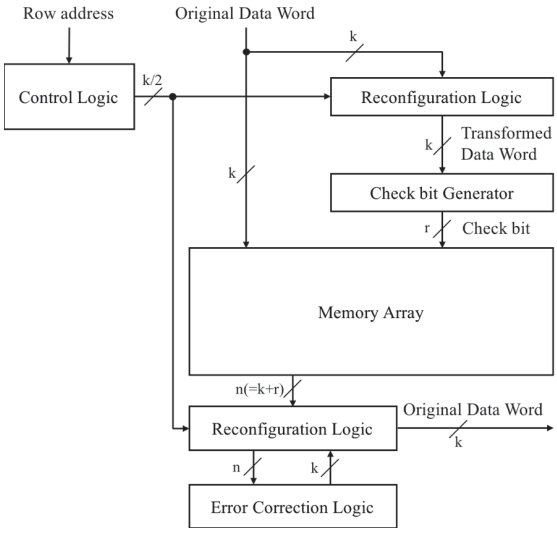


Fig. 3. Architecture of Proposed UEP Code

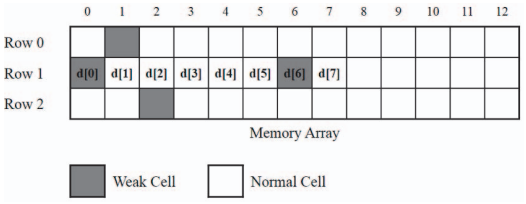


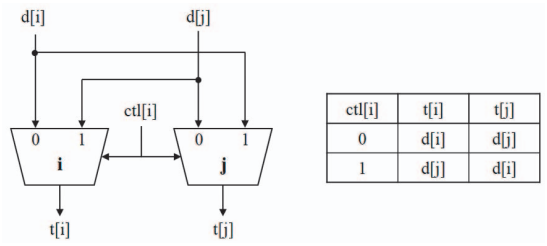
Fig. 4. Memory array

operating voltage is applied. Hence, the *weak cells* are more susceptible to soft errors. Assume that the proposed code is a (*13,8*) block code and the memory array has some *weak cells* highlighted in grey as shown in Fig. 4. When there is a memory write operation with its target address of row 1, the original data word (d[0], d[1], …, d[7]) is stored in the row 1. d[0] and d[6] are stored in *weak cell*, hence, they need to be protected stronger than other data stored in *normal cells*. However, as the proposed code provides a strong protection for the *weak region* that is the first half of the data word (d[0] ~ d[3]), the d[6] cannot be protected by a stronger level.

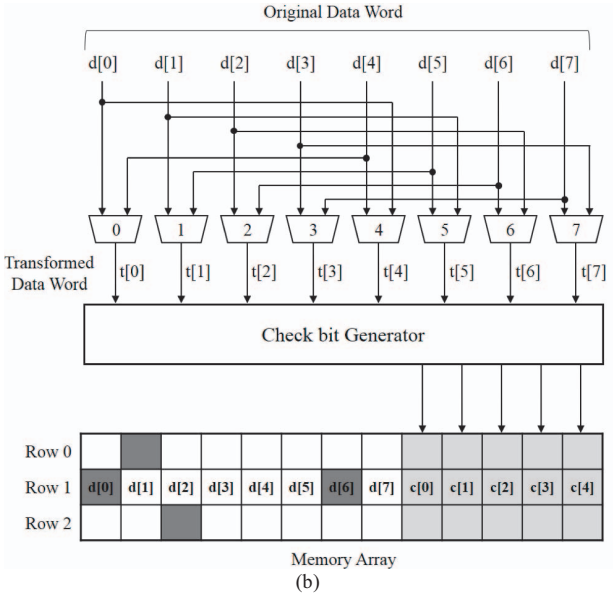
To resolve this problem, the reconfiguration logic is introduced. This logic rearranges the original data word (d[0], d[1], …, d[7]). The transformed data word (t[0], t[1], …, t[7]) that is a rearranged original data word, is used to produce check bits in Fig. 3. Therefore, the first half of the transformed data word (t[0], t[1], t[2], t[3]) can be strongly protected. The reconfiguration logic consists of *k/2* pairs of MUXes (where *k* is the number of the data bits) and has *k/2*-bit control signal. One pair of MUXes is shown in Fig. 5(a) and this shows how the reconfiguration logic works. Two MUXes share 1-bit control signal (ctl[*i*]). If the control signal is zero, two input bits pass through the MUXes (t[*i*] = d[*i*], t[*j*] = d[*j*]). Otherwise, they are switched (t[*i*] = d[*j*], t[*j*] = d[*i*]).

Fig. 5(b) depicts the encoding process that is performed in two steps. The first phase is the reconfiguration logic. There are 4 pairs of MUXes (pair[0] : MUX0 & MUX4 (both receiving d[0] and d[4]), pair[1] : MUX1 & MUX5, pair[2] : MUX2 & MUX6, pair[3] : MUX3 & MUX7) and each pair has 1-bit control signal (ctl[*i*] for pair[*i*]). To provide strong protection for d[0] and d[6], ctl[0] and ctl[2] are respectively set to *0* and *1* and the rest control signal bits are set to *0*. The data word before and after the reconfiguration logic is shown in Fig. 5(c). The second phase of the encoding process is to generate check bits which takes the transformed data word and produce parity bits. As the first half of the transformed data word contains the d[0] and d[6], all data bits stored in *weak cell* can be protected by strong level. It should be noted that the original data word is stored in memory array as shown in Fig. 4 and the data word is transformed only to generate check bits as illustrated in Fig. 5.

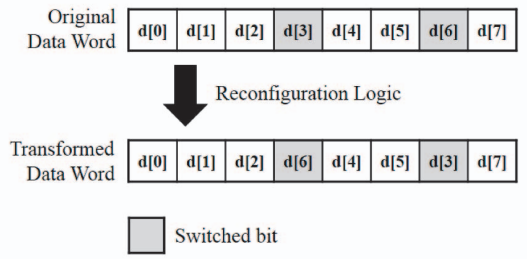
The control logic sends control signals for the reconfiguration logic. The MUX control signals for each memory row needs to be stored in the control logic. However, as the rows in memory array increases, the number of control signals also increases and this would introduce a significant control logic area overhead. To relieve this problem, we divide



(a)



(b)



(c)

Fig. 5. Proposed UEP Code Operation Example (a) One Pair of MUXes for Reconfiguration (b) Encoding Operation (c) Original and Transformed Data Word

the memory array into partitions and each partition shares the same control signal. Fig. 6 shows a memory array with *weak cells* highlighted in grey. It should be noted that the location of *weak cells* would not be completely random. Since the *weak cells* are defined as the cells with low margins by process variations, they would appear as a clustered form rather than completely random. If (16,6) block code is used and we assume that each partition can have *weak cells* only at four bit locations, the memory can be divided into partitions such as *partition_0* (row0 ~ row2 : *weak cells* are at d[1], d[2], d[5] and d[6]), *partition_1* (row3 ~ row7 : *weak cells* are at d[8], d[9], d[11]

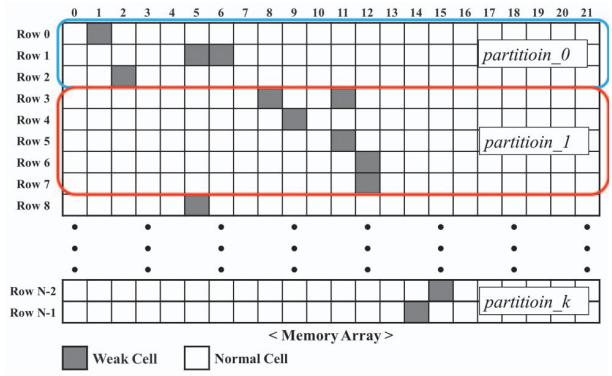


Fig. 6. Memory Partitioning to Reduce Control Data Overhead

and d[12]) and so on. The control signal for *partition_0* is 00000000 and it is shared for row 0 ~ row2. In the same manner, the control signal for *partition_1* can be found as 11011000.

D. Decoding Process

The decoding process are performed in three stages – the reconfiguration logic, the error correction logic, and the reconfiguration logic as depicted in Fig. 3. Since the transformed data word is used to generate a check bit, the error correction logic also requires the transformed data word. To generate the transformed data word, the original data word stored in the memory is fed into the reconfiguration logic. The error correction logic takes the codeword (transformed data word + check bit) and produces the corrected data word. In the last stage of decoding procedure, the corrected data word passes through the reconfiguration logic.

IV. Simulation Results

The proposed architecture is implemented with Verilog HDL and synthesized by Synopsis Design Compiler for 32nm library. For experiments, this paper generates six *H*-matrices which are (16,6), (16,7), (32,7), (32,8), (64,8) and (64,9) UEP codes (SEC-DED-TAEC-DAEC / SEC-DED).

A. Area, Delay and Power Analysis

The proposed codes are compared to the SEC-DED-DAEC code [3] with respect to area, power and delay. Table I shows the comparison results. The area includes the reconfiguration logic, check bit generator and error correction logic. The delay is a time required to perform the decoding process when a read request is given. When a codeword length is the same, the proposed architecture needs more area, delay and power in comparison with the SEC-DED-DAEC code. These overhead would be caused by the additional reconfiguration logic. However, when an additional 1 check bit is allowed, the proposed code require less area, delay and power than the SEC-DED-DAEC code.

B. Miscorrection Probability

As addressed in Section III, the miscorrection probability arises when some double bit errors and double adjacent bit errors share the syndromes. It may cause miscorrection of some double nonadjacent bit errors, hence, it should be minimized. The miscorrection probability can be calculated by:

$$\Pr(miscorrction) = \frac{\#DAEC - (n-1)}{C_2^n - (n-1)}$$

In this equation, ${}_nC_2$ is the number of all double bit errors when the codeword is *n*-bit long and *#DAEC* is the number of total DAEC operations. As the *n-1* is the number of double adjacent bit errors, the denominator is the number of all double non-adjacent bit errors and the numerator is the number of miscorrection operations. Table II shows the miscorrection probability comparisons. The proposed codes have low

Table I. Area, Delay and Power Analysis

Code	*k*	*r*	Area (μm^2)	Delay (ns)	Power (μW)
Dutta [3]	16	6	673.86	2.62	124.07
Proposed	16	6	822.63	2.95	152.62
Proposed	16	7	583.57	2.29	83.62
Dutta [3]	32	7	1289.79	3.91	251.79
Proposed	32	7	1499.16	4.24	284.52
Proposed	32	8	1278.97	2.31	244.52
Dutta [3]	64	8	2460.06	6.42	490.53
Proposed	64	8	2850.09	6.19	522.00
Proposed	64	9	2439.68	2.90	488.13

Table II. Code Comparison – Miscorrection Probability

Code	Data-bits	Check-bits	Mis. Prob.
SEC-DED-DAEC [3]	16	6	64.3%
Proposed code	16	6	20.6%
DAEC-SEC-DED [3]	16	7	47.2%
SEC-DED-TAEC-4AED [6]	16	7	29.0%
Proposed code	16	7	8.2%
SEC-DED-DAEC [3]	32	7	57.3%
DAEC-SEC-DED [7]	32	7	57.3%
Proposed code	32	7	22.8%
SEC-DED-TAEC-4AED [6]	32	8	24.8%
Proposed code	32	8	7.6%
SEC-DED-DAEC [3]	64	8	54.6%
Proposed code	64	8	23.6%
DAEC-SEC-DED [7]	64	9	36.2%
Proposed code	64	9	9.7%

Table III. Miscorrection Probability in Proposed Codes

k	*r*	Total Region	Weak Region
16	6	20.6%	0.0%
32	7	22.8%	18.1%
64	8	23.6%	19.8%
16	7	8.2%	0.0%
32	8	7.6%	0.0%
64	9	9.7%	3.44%

miscorrection probability and this tells that the proposed method achieves highly reliable memory operations.

Table III shows the miscorrection probability of the proposed codes with different k and r. The analysis proves that the proposed UEP codes can significantly increase the reliability of *weak cells*. Fig. 7 to Fig. 12 are H-matrices by the proposed method.

V. Conclusion

In this paper, the UEP code that consists of SEC-DED-TAEC-DAEC and SEC-DED code is presented and evaluated. The proposed code to improve memory reliability provides *weak cells* strong protection. The *weak cells* are memory cells that would fail first when the lower operating voltage is applied than its normal operation voltage. They have low design margins than other cells by process variations and the manufacturing characterization test can identify them.

The different protection levels for *weak cells* and *normal cells* significantly enhances memory reliability. The results show that the proposed code has low miscorrection probability compared to other codes. This proves that the memory reliability can be efficiently improved by considering process variations instead of just assuming random soft errors at any locations.

VI. Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01058856) and Samsung Research Fund.

References

[1] Ibe, Eishi, et al. "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule." *IEEE Transactions on Electron Devices* 57.7 (2010): 1527-1538.G. Eason, B. Noble, and I. N. Snedd

[2] Reviriego, Pedro, Juan Antonio Maestro, and Catalina Cervantes. "Reliability analysis of memories suffering multiple bit upsets." *IEEE Transactions on Device and Materials Reliability* 7.4 (2007): 592-601.

[3] Dutta, Avijit, and Nur A. Touba. "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code." *25th IEEE VLSI Test Symposium (VTS'07)*. IEEE, 2007.

[4] Masnick, Burt, and Jack Wolf. "On linear unequal error protection codes." *IEEE Transactions on Information Theory* 13.4 (1967): 600-607.

[5] Dutta, Avijit, and Nur A. Touba. "Reliable network-on-chip using a low cost unequal error protection code." *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*. IEEE, 2007.

[6] Neale, Adam, Maarten Jonkman, and Manoj Sachdev. "Adjacent-mbu-tolerant sec-ded-taec-yaed codes for embedded SRAMs." *IEEE Transactions on Circuits and Systems II: Express Briefs* 62.4 (2015): 387-391.

[7] Reviriego, Pedro, et al. "A Method to Design SEC-DED-DAEC Codes With Optimized Decoding." *IEEE Transactions on Device and Materials Reliability* 14.3 (2014): 884-8

$$\begin{bmatrix}
1010010100111000100000\\
1111101000011100010000\\
1001001011100010001000\\
0100101111001001000100\\
1000110010110111000010\\
1111010101000111000001
\end{bmatrix}$$

Fig. 7. H-matrix for proposed (16,6) code

$$\begin{bmatrix}
0110101011001100100000\\
1010001010110000010000\\
0011000101111000010000\\
0100010011100110001000\\
1001011000011101000100\\
0000100100000011000010\\
1101110100000011000001
\end{bmatrix}$$

Fig. 8. H-matrix for proposed (16,7) code

$$\begin{bmatrix}
11111001010100010011000001110000010000000\\
10111111001001110000100110001000010000000\\
01010011111110111000011000000100001000000\\
10011110010010000111111100000010000100000\\
01001001101001001101110011000010000100000\\
10100000100100100010001111011110000010000\\
10011101000011001100000000111111000000001
\end{bmatrix}$$

Fig. 9. H-matrix for proposed (32,7) code

$$\begin{bmatrix}
10101010000001011111001100001000100000000\\
00100101010001001100110010010000010000000\\
01010010110100000011111001100000000100000\\
10010010011000101000000111111100000100000\\
11010001000110100110000000000010000010000\\
00100100001010010001100000111001000001000\\
01011100101010000000011110001110000100000010\\
00011001100101110000000000000011100000001
\end{bmatrix}$$

Fig. 10. H-matrix for proposed (32,8) code

$$\begin{bmatrix}
01111110001011100100100101001000011000000011100111000000001110001000000\\
00100100110010010111101010111110110110000100001000000000000000000001000000\\
11111001001111110010011000000010001101111001110000000000111000001000000\\
10010111011101001001000100001011000011111110000000001111000111000001000\\
11100111111000111000010010010101000000001111000001110011100000100000100\\
01011111110100101110010001110001100000010000100010111100010010010000100\\
11011010100100001001101001000100100000100000011100100110001101110000010\\
10101001111100101000000110100000011111000000011111011001110001110000001
\end{bmatrix}$$

Fig. 11. H-matrix for proposed (64,8) code

$$\begin{bmatrix}
01010101010100010101001000000010000111001111110000000010011000001000000000\\
11010111100001110010001101000010000000110000001000011100000000000010000000\\
00101010010100100000110010010100000001111110000100100000000000000001000000\\
00100000001010101111010101111100111110000110000011000000000000000000100000\\
10101010101010001000001000100001000000011000000000001111111000000000010000\\
01011100000000010000000010101001000011100011111111111000000011100000001000\\
01011000001001000010110010001011110100000011000000001110100111010000000100\\
11100111010011000001000100010000101000000001000011100010101101100000000010\\
00111110100100001100100001000100011000000000011110000001101001110000000001
\end{bmatrix}$$

Fig. 12. H-matrix for proposed (64,9) code