Multiple Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DAEC Code

Avijit Dutta and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712

Abstract

Conventional error correcting code (ECC) schemes used in memories and caches cannot correct double bit errors caused by a single event upset (SEU). As memory density increases, multiple bit upsets in nearby cells become more frequent. A methodology is proposed here for deriving an error correcting code through heuristic search that can detect and correct the most likely double bit errors in a memory while minimizing the miscorrection probability of the unlikely double bit errors. A key feature of the proposed ECC is that it uses the same number of check bits as the conventional single error correcting/double error detecting (SEC-DED) codes commonly used, and has nearly identical syndrome generator/encoder area and timing overhead. Hence, there is very little additional cost to using the proposed ECC. The proposed ECC can be used instead of or in addition to bit interleaving to provide greater flexibility for optimizing a memory layout and/or provide better protection from multiple bit upsets. It is also useful for small memories, e.g., content addressable memory or register files, where interleaving is not possible.

1. Introduction

Ionizing radiation from high-energy neutrons and alpha particles can cause a *single-event upset (SEU)* that may alter the state of the system resulting in a *soft error*. Memories, which occupy a large percentage of the area of a chip, are especially sensitive to SEUs. Constant technology process improvement has resulted in very dense memory cells that store information with less capacitance and lower voltage. Consequently, less charge is required to produce one or more soft errors in memories. Recent studies characterizing different bit errors arising from an SEU suggest that 1–5% of the SEUs can cause multiple bit upsets (MBUs) [Maiz 03]. Depending on the underlying technology and the incident particle, several types of multiple-bit errors are possible [Satoh 00], [Makihara 00], [Kawakami 04]. It has been shown that incident neutron particles can react with the die contaminants and generate secondary particles with enough energy to create multiple errors. The distance between the bits in error depends on the initial angle of incidence, die contaminant types, and the scattering angle for the secondary particles. Based on this, the probability of adjacent double bit errors is much higher than other multiple bit errors.

A SEC-DED code [Hamming 50] is capable of correcting one error and detecting all possible double errors. It is commonly used in memories and caches, but cannot correct more than a 1-bit error in a word. In order to correct the most commonly occurring MBUs, this paper proposes a low cost ECC methodology to correct double adjacent bit errors. It involves constructing a single-error-correcting, double-error-detecting, double-adjacent-error-correcting (SEC-DED-DAEC) code by selectively avoiding certain types of linear dependencies in the parity check matrix. A key feature of the proposed SEC-DED-DAEC code is that it uses the same number of check bits and has nearly identical syndrome generator/encoder area and timing overhead as the conventional SEC-DED codes. Consequently, there is very little additional cost to using it. Specific *H*-matrices for 16, 32 and 64-bit data words are given in the paper, and their properties are directly compared with commonly used SEC-DED codes published elsewhere.

While the focus in the paper is on SEC-DED-DAEC codes, the proposed methodology is flexible and can be used to construct codes for correcting any subset of double errors.

2. Related Work

A number of approaches for extending the basic SEC-DED Hamming code [Hamming 50] have been previously proposed. A special class of SEC-DED codes known as Hsiao codes [Hsiao 70] was proposed to improve the speed, cost, and reliability of the decoding logic. The codes constructed in the proposed methodology can be thought of as a special class of Hsiao codes. Another class of SEC-DED codes [Reddy 78], [Chen 83] was proposed to detect any number of errors affecting a single byte. These codes are known as single-error-correcting double-error-detecting single-byte-error-detecting (SEC-DED-SBD) codes. For protecting byte-organized memories, SEC-DED-SBD codes are more suitable than the conventional SEC-DED code.

To provide byte error correction capability, single-byte-error-correcting, double-byte-error-detecting (SBC-DBD) codes [Berlekamp 68], [Reed 60], [Wolf 69], [Bossen 70] [Chen 96] were proposed. These codes perform at a higher order Galois field and consequently the encoding and decoding are more complex. Moreover,

25th IEEE VLSI Test Symmposium (VTS'07)
0-7695-2812-0/07 $20.00 © 2007 IEEE

they require more check bits thereby increasing the size of the memory.

To provide complete double error correction capability, a double-error-correcting triple-error-detecting (DEC-TED) code may be used at the cost of much larger overhead in terms of both the check bits and more complex hardware to implement the error correction and detection [Lin 83], [Berlekamp 68], [Lala 78].

The Reed-Solomon (RS) code and Bose-Chaudhuri-Hocquenghem (BCH) codes are able to detect and correct multiple bytes of errors with very low overhead in terms of additional check bits required. However, these codes typically work at the block level and are applied to multiple words at a time. Other similar codes include the extended Hamming code [Bossen 70] which performs at a higher order Galois field GF (2^K) and can correct up to k-bit burst errors. Other multiple error correcting codes include the optimal rectangular code (ORC), adaptive cross-parity code (APX) code, and others. The general drawbacks with these methods are latency and speed. Most of these codes require several cycles to correct the first error. Moreover, the encoding and decoding are much more complex and require several table lookups for multiplication in higher order fields.

Another class of multiple error-correcting approaches combines coding with circuit level techniques to sense multiple errors in a memory. In [Vargas 94] and [Calin 95], an asynchronous built in current sensor (BICS) on the vertical power lines of a memory along with a parity bit per memory word is used. A conventional SEC-DED code and the BICS approach are combined in [Gill 05] to detect multiple bit upsets affecting the same memory word.

Even though several powerful error correcting codes exist, the SEC-DED code has remained an attractive choice mainly because of its fast and simple encoding/ decoding and low hardware overhead. One of the most commonly used techniques to minimize the probability of multiple bit upsets in a single word is *bit interleaving* [Maiz 04] which is a memory layout architecture in which physically adjacent bits are assigned to different logical words. For k-way interleaving, k adjacent failing bits appear as k single bit errors in k different logical words rather than as a k-bit error in a single logical word. A simple SEC-DED code can be used along with bit interleaving to help protect from multiple bit upsets. However, there can be some limitations/drawbacks for bit interleaving. In some cases, it may negatively impact floorplanning, access time, and/or power consumption. The proposed SEC-DED-DAEC code requires very little overhead and can be used instead of or in addition to bit interleaving to provide greater flexibility for optimizing a memory design. For a fixed depth of interleaving, a larger physical distance between cells in error can be tolerated using the proposed code, or to tolerate a fixed physical distance of cells in error, the required depth of interleaving can be reduced. The proposed methodology places an additional tool in the hands of a memory designer for optimizing a memory layout. Moreover, for small memories, e.g., content addressable memory or register files, interleaving may not be feasible. The proposed coding methodology is particularly useful in this case to provide protection from MBUs.

A class of systematic SEC-DED-DAEC codes was proposed much earlier in [Abramson 59]. However, it was not targeted for memories. Its encoding and decoding are not as efficient as conventional SEC-DEC codes. One check bit is dedicated to differentiate between single and double bit errors. This check bit computes the parity of the entire message and hence incurs a lot of decoding delay and large decoder overhead. Moreover, the encoding and decoding involve the use of a linear finite state machine (LFSM) and hence the latency is increased. Some extensions of the basic code in [Abramson 59] have been suggested. In [Elspas 60], the SEC-DED-DAEC code was extended to higher order fields GF (2^K), and in [Bernstein 63], the code was modified for arithmetic operations.

The ECC methodology proposed in this paper constructs a different SEC-DED-DAEC code from the ones described in [Abramson 59], [Elspas 60], and [Bernstein 63]. The proposed SEC-DED-DAEC codes are targeted for memories and have the same number of check bits and nearly identical encoding and decoding latency as conventional SEC-DED codes. The proposed codes are constructed by selectively avoiding certain type of cycles in the parity check matrix. Moreover it tries to minimize the miscorrection (non-adjacent double error mistaken as an adjacent double error) probability.

3. Binary Linear block codes

The proposed SEC-DED-DAEC code falls into the category of systematic binary linear block codes. A binary (n, k) linear block code is a k-dimensional subspace of a binary n-dimensional vector space. An n-bit codeword of the code contains r=(n-k) check bits and k data bits. The ($r \times n$) parity-check matrix (H-matrix) completely defines the code. C is a codeword of the code if and only if

$$H.C^T = 0 \qquad (1)$$

where C^T is the transpose of the codeword C. The H-matrix corresponds to a systematic code if it can be represented as

$$H=[P_{r\times k},I_{r\times r}] \qquad (2)$$

where I is the $r \times r$ identity matrix. For a systematic code, the first k-bits of the codeword can be designated as the data bits and the last r bits can be designated as the check bits. For the targeted application, only systematic codes are useful. For a systematic code with a parity check matrix of the form given by Eqn. 2, the generator matrix can be simply obtained as

$$G=[I_{K\times K},P^T] \qquad (3)$$

The *H*-matrix represents a set of linear equations involving the bits of the message. The syndrome is defined as the *r*-bit vector obtained upon multiplying the received *n*-bit message with the *H*-matrix in GF (2). In the error free case, the syndrome is the all-zero vector. An error vector is defined as an *r*-bit vector where the bits that are in error have the value 1 and all the other bits are 0. An erroneous message V_e can be represented as

$$V_e = V + E \quad (4)$$

where *E* is the error vector and *V* is the error free message (i.e., codeword).

$$S = H.V_e = H.(V+E) = H.V + H.E = H.E \quad (5)$$

where *S* is the syndrome for the particular message V_e. In the next section, we will discuss the proposed linear systematic block code.

4. Proposed Code

The proposed SEC-DED-DAEC code has the following properties:

1) All single bit errors can be corrected
2) All double bit errors can be detected
3) All adjacent double bit errors can be corrected
4) The miscorrection probability for non-adjacent double errors is reduced

The characteristics of a linear block code are completely determined by its *H*-matrix. To detect all single bit errors, the corresponding error syndromes should be unique. Note that the syndrome for a single bit error at the bit position *p* is the same as the *p*-th column of the *H*-matrix. To uniquely identify all the single bit errors, all the columns of the *H*-matrix must be unique.

To detect all the double bit errors, the corresponding syndromes should be different from all the single bit error syndromes. The syndrome for a double bit error is given by the exclusive-or (XOR) of the corresponding columns of the *H*-matrix. So there cannot be any 3-cycle in the H-matrix. A *k*-cycle refers to a set of *k* linearly dependent columns of the parity check matrix, i.e., when XORed together, the output is an all-zero column. To be able to correct all the adjacent double bit errors, the syndromes for the adjacent double bit errors should be different from each other and also different from all the single-error syndromes. Next we define the conditions that must be satisfied by the *H*-matrix for the proposed code:

1) No all 0 columns.
2) All columns are distinct
3) No linear dependency involving 3 or less columns i.e., no 2-cycle and 3-cycle are allowed.
4) No linear dependency involving columns C_i, C_j, C_k, C_m where $m>k>j>i$, such that $j=i+1$ and $m=k+l$.
5) Moreover the code tries to minimize the number of 4-cycles involving C_i, C_j, C_k, C_m where $m>k>j>i$, such that $j=i+1$ or $k=j+l$ or $m=k+1$.

Condition 1 ensures that no single bit error case match the error free case.

Condition 2 ensures that all the single error syndromes are unique. Every single error syndrome matches one of the columns of the *H*-matrix. Since all the columns of the *H*-matrix are distinct, the single bit errors are uniquely identifiable and hence correctable. Additionally this condition ensures that there are no pairs of double errors of the form (*i,j*) and (*j,k*) such that the corresponding syndromes are the same. Assume that such double errors exist, then $(C_i \oplus C_j) \oplus (C_j \oplus C_k) = 0$, i.e., $(C_i \oplus C_k) = 0$ but that contradicts the fact that all the columns of the *H*-matrix are distinct. This ensures that syndromes for adjacent errors of the form *(i,i+1)* and *(i+1,i+2)* are different.

Condition 3 ensures that the syndromes for all double bit errors are different from that of the single bit errors. The syndrome for a double bit error is determined by the XOR of the columns corresponding to the erroneous bit positions. If the *H*-matrix is free of 3-cycles then the XOR of any two columns of the *H*-matrix is not identical to any of the columns of the *H*-matrix. This ensures that the syndromes of all the double bit errors are different from the single bit error syndromes, and condition 2 ensures that the double bit error syndromes are non-zero. Hence all the double bit errors are detectable.

Condition 4 along with condition 2, ensures that a syndrome for an adjacent double bit error is different from all other adjacent double bit error syndromes. If we assume that the only errors are single bit errors or adjacent double bit errors, then with an *H*-matrix satisfying conditions 1 through 4, we can uniquely identify the syndromes for all single bit errors and adjacent double bit errors and hence can correct all single bit errors and all double adjacent bit errors and detect all double bit errors.

However the syndromes for the adjacent bit errors are shared with some non-adjacent double bit error syndromes. This is because some 4-cycles are allowed in order to reduce the check-bit overhead. So there is a possibility that a non-adjacent double bit error will be mistaken as an adjacent double bit error and hence will be incorrectly corrected (although the probability of non-adjacent double errors is much less than that of the adjacent double errors). Condition 5 tries to minimize the probability of such an event happening. We call the 4-cycles of the type given by condition 4, *forbidden 4-cycles (4FC)*. We call the 4-cycles of the type C_i, C_j, C_k, C_m where $m>k>j>i$, such that $j=i+1$ or $k=j+l$ or $m=k+1$, *bad cycles (4BC)*, since their presence have a detrimental effect on the capacity of the code. The number of non-adjacent double bit errors is $C_2^n - (n-1)$. For the double errors that are caused by independent SEUs, all the double errors are equally likely and the miscorrection probability is given by:

$$Pr\ (miscorrection) = \frac{\#4BC}{C_2^n - (n-1)} \quad (6)$$

By reducing the number 4BCs, the miscorrection probability can be minimized.

5. Code Design Procedure

The design of the *H*-matrix is essentially a systematic search process to satisfy all the conditions mentioned in the previous section. For an ($r \times n$) matrix, there are $2^{(r \times n)}$ possible choices, so an exhaustive search approach is ruled out for reasonably large values of r and n. Figure 1 shows a *H*-matrix for a (22,16) code. Even for this code, an exhaustive search is not practical even if the domain of columns considered is restricted. The *weight* of a column of the *H*-matrix is defined as the number of 1's in the column. If we limit the *H*-matrix to only weight-3 and weight-1 columns, then there are $C_3^6 = 20$ choices out of which 16 columns can be chosen in $C_{16}^{20} = 4845$ ways. For each choice there are ($16! > 2 \times 10^{13}$) column permutations which should be searched for the best code. So an exhaustive search will have to search ($4845 \times 16! > 2 \times 10^{13}$) matrices for the best code. The search space increases further if arbitrary weighted columns are allowed. Note that For the H-matrix in Fig. 1, no column of weight two is allowed because any weight-2 column will create a 3-cycle with two weight-1 columns.

Constructing the best *H*-matrix for a SEC-DED-DAEC code that satisfies all of the conditions discussed in Sec. 4 is NP-complete. A pseudo-greedy search procedure can be used as shown in Fig. 3. The outer while loop stops once a valid code is found or the maximum backtrack limit is exceeded. The inner while loop finds a set of valid columns (that does not introduce any forbidden cycles) for the current column position. If no valid column is found for the current column position, then the last choice for a column has to be undone. This corresponds to a backtrack. If multiple valid columns are found for the current column position then the one that minimizes the number of bad 4-cycles in the currently constructed code space is chosen. Once an initial *H*-matrix is found, a limited number of column permutations are tried to avoid a local optimum and search for a better *H*-matrix in terms of reduced miscorrection probability.

$$\begin{pmatrix}
1100110001110100100000\\
0001010011011001010000\\
1010100111100011001000\\
1001001110010110000100\\
0110101100001101000010\\
0111011000101010000001
\end{pmatrix}$$

Figure 1. *H*-matrix for proposed (22,16) code

$$\begin{pmatrix}
10111001010110000100101001110000 1000000\\
01010100101100010001110001100001 0100000\\
10101010011000101011000011000011 0010000\\
01010001110001010110000110000111 0001000\\
00100110000010110100101100001110 0000100\\
10000100100101101001011010011100 0000010\\
01001011001011001010010100111000 0000001
\end{pmatrix}$$

Figure 2. *H*-matrix for proposed (39,32) code

```
Input:  n, maxIter, maxBacktrack, maxPermute
Output:  H-matrix

avail_col = All 1-weight columns, followed by 3-weight
columns, followed by 5-weight columns, and so forth up to
the largest weight columns being considered
currentCol = 0;  backtrack = 0
while ( currentCol < n ) {
   Iter = 0
   validColPool[currentCol] = {}
   while ( iter < maxIter ) {
      Iter++
      C = Untried least-weighted column from avail_col
      Check for existence of forbidden 4-cycles
      if ( ! 4FCfound ) {
         validColPool[currentCol] =
               validColPool[currentCol] ∪ C
      }
   }
   if ( empty(validColPool[currentCol]) ) {
      backtrack++
      if ( backtrack > maxBacktrack ) {
         return   // no code found
      } else {
         currentCol--
         if ( currentCol < 0 )  currentCol = 0;
         continue;
      }
   } else {
      sCol = selectMin4BC(validColPool[currentCol])
      add sCol to H-matrix
      currentCol++
      backtrack=0;
   }
}
permuteC = 0;  orig4BC=count4BC(H-matrix);
while ( permuteC < maxPermute) {
   permuteC++
   permuteColumns()
   Check for existence of forbidden 4-cycles
   if ( (!4FC)&&(count4BC(H-matrix)<orig4BC)) {
      H-matrix ← current H-matrix;
   }
}
```

Figure 3. Pseudo-greedy search algorithm

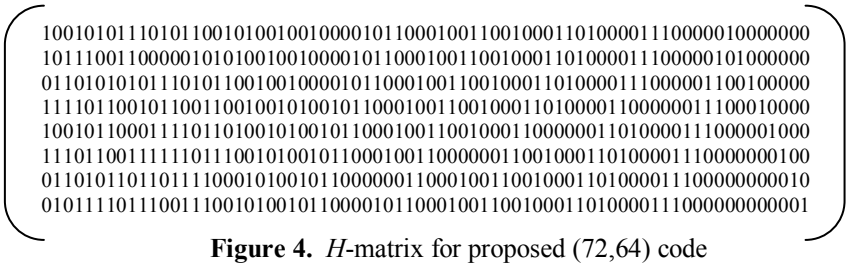

Figure 4. *H*-matrix for proposed (72,64) code

Table 1. Comparison of proposed SEC-DED-DEAC code with other codes

(n,k)	Codes	2-Input XOR Gates	Max Logic Depth	Forbidden 4-Cycles (4FC)	Total 4-Cycles	Bad 4-Cycles (4BC)
(22,16)	SEC-DED (IBM system/3)	48	4	13	252	122
	Hsiao Code [Hsiao 70]	48	4	8	252	120
	SEC-DED-DAEC in [Abramson 59]	70	5	0	252	128
	Proposed SEC-DEC-DAEC (Fig. 1)	48	4	0	251	118
(39,32)	SEC-DED (IBM 8130)(40,32)	96	4	82	776	254
	Hsiao Code [Hsiao 70]	96	4	23	1363	425
	SEC-DED-DAEC in [Abramson 59]	132	6	0	1343	386
	Proposed SEC-DED-DAEC (Fig. 2)	96	4	0	1363	379
(72,64)	SEC-DED (IBM 3081)	256	6	230	8912	1292
	Hsiao Code [Hsiao 70]	208	5	122	8392	1399
	SEC-DED-DAEC in [Abramson 59]	296	7	0	8194	1335
	Proposed SEC-DED-DAEC (Fig. 4)	224	5	0	8289	1316

The number of the 2-input XOR gates required for the encoding/decoding can be computed from the *H*-matrix. It is equal to $\sum_{\#rows}$ *(row weight − 1)*. The encoding and decoding delays are determined by the maximum logic-depth of the encoder and the decoder circuit which is equal to log_2 *(max. 1's in any row)*. Figure 2 shows an *H*-matrix for the (39,32) code constructed using the search process as discussed above using only weight-1 and weight-3 columns. Another *H*-matrix is shown for a (72,64) code in Fig. 4. In this case, weight-1, weight-3, and weight-5 columns are used.

Table 1 shows the number of XOR gates and maximum logic depth for the syndrome generator, number of forbidden 4-cycles, total number of 4-cycles, and the number of bad cycles (4BCs) for the (22, 16), (39, 32) and (72,64) codes for both the proposed code and the SEC-DED-DAEC code described in [Abramson 59] as well as some Hsiao codes and SEC-DED codes commonly used in industry. Note that the SEC-DED code and the Hsiao code cannot correct adjacent double bit errors because of the existence of forbidden cycles (4FCs). Using a random double error correcting (DEC) code can increase the memory size considerably and hence is not an attractive choice for memory ECC. For example to protect a 32-bit word, a DEC code needs at least 11 check bits and 14 check bits are needed to protect a 64-bit word. The proposed code also has minimal logic depth among the codes and also minimum check bit overhead. The total number of bad 4-cycles is lower for the proposed code than for [Abramson 59], and consequently it has a lower miscorrection probability.

6. Encoding/Decoding Algorithm

The proposed code is systematic. During encoding, the data bits can be directly copied and the check bits are generated using an XOR network corresponding to the *G*-matrix. The decoding algorithm is as follows:

1) Generate the syndrome using an XOR network corresponding to the *H*-matrix.
2) If the syndrome is the all zero vector, then no error is detected, otherwise one or more errors occurred.
3) If the syndrome matches any of the *H*-matrix columns, then a single error is detected and the error position is the corresponding column position. The corresponding bit should be flipped to correct the error.
4) Else if the syndrome matches any of the *n-1* adjacent double error syndromes, then a double adjacent error is detected and the corresponding bit positions are generated using the error correction logic.
5) Else an uncorrectable error (UE) (i.e., a double non-adjacent error or more than two errors) has occurred.

The only additional overhead with respect to a conventional SEC-DED code comes from step 4 of the decoding step. Figure 5 shows the basic error detection and correction block diagram. If a non-zero syndrome is encountered, then the OR gate flags an error indication. If the syndrome matches any of the single error syndromes

then the syndrome decoder generates a 1 in the erroneous bit position. Otherwise, if the syndrome matches any of the adjacent double error syndromes, then the decoder generates 1's at the erroneous adjacent bit positions. Otherwise the output of the syndrome decoder is the all zero output. The syndrome decoder consists of 3-input OR gates whose inputs are driven by outputs of *r*-input AND gates. The *i*-th output of the decoder is 1 if and only if a single error occurred at the *i*-th bit or a double-adjacent error occurred at <*i*,*i*+1> bits or <*i*-1,*i*> bits. The outputs of the decoder are used to generate the corrected word, by using *n* 2-input XOR gates. If the syndrome is non-zero and does not match any of the single or double-adjacent error syndromes, then the UE signal is flagged.

7. Conclusions

The ECC methodology described in this paper adds the ability to correct adjacent errors at very little cost over conventional SEC-DED codes. The only drawback is the possibility of miscorrection for a small subset of multiple errors, however MBUs caused by a single SEU have a much higher probability of occurring than having multiple independent SEUs accumulating in the same word (this is especially the case if memory scrubbing is used). While bit interleaving is commonly relied upon to protect memories from MBUs, the proposed methodology provides another tool for a memory designer to use. In some instances, it may be an attractive alternative to bit interleaving to allow for a more optimized memory layout, or it can be used in addition to bit interleaving to provide an additional layer of protection from MBUs.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. CCR-0426608.

References

[Abramson 59] Abramson N. M., "A Class of Systematic Codes for Non-Independent Errors", *Proc. IRE Trans. on Information Theory*, Vol. IT-5, pp. 150-157, Dec. 1959.

[Bernstein 63] Bernstein, A., and W. Kim,"Single and Double Adjacent Error-correcting codes for arithmetic Units", *IEEE Tran. on Information Theory*, Vol. 9, pp. 209-210, Mar. 1963.

[Berlekamp 68] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[Bossen 70] Bossen, D. C.,"b-Adjacent Error Correction", *IBM Journal of Research and Development,* Vol. 14, pp. 402-408, Jul. 1970.

[Calin 95] Calin, Th., F. L. Vargas, and M. Nicolaidis,"Upset Tolerant CMOS Using Current Monitoring: Prototype and Test Experiments", *Proc. Int. Test Conference*, pp. 45-53, 1995.

[Chen 68] Chen, C. L.," Error Correcting Codes with Byte Error Detection Capability", *IEEE Trans. On Computers,* Vol. C-32, pp. 615-621, May 1983.

[Chen 96] Chen, C. L.,"Symbol Error Correcting Codes for Memory Applications", *Proc. of Fault Tolerant Computing Systems,* pp. 200-207, 1996.

[Elspas 60] Elspas, B.,"A Note on p-nary Adjacent-error-correcting Codes", *IEEE Trans. on Information Theory*, Vol. 6, Mar. 1960.

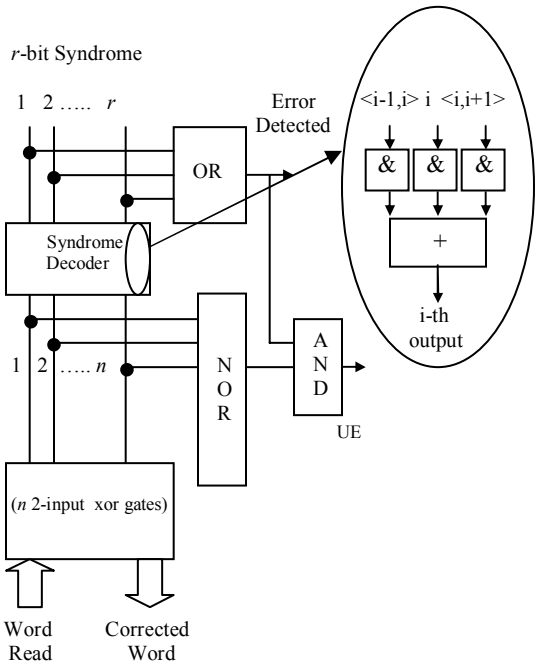

Figure 5. Error detection and correction block diagram

[Gill 05] Gill, B., M. Nicolaidis, and C. Papachristou," Radiation Induced Single-Word Multiple-bit Upsets Correction in SRAM", *Proc. of Int. Online Test Symposium*, pp. 266-271, Jul. 2005.

[Hamming 50] Hamming, R.W., "Error Correcting and Error Detecting Codes", *Bell Sys. Tech. Journal*, Vol. 29, pp. 147-160, Apr. 1950.

[Hsiao 70] Hsiao, M. Y., "A Class of Optimal Minimum Odd-weight-column SEC-DED codes", *IBM Journal of Research and Development*, Vol. 14, pp. 395-401, 1970.

[Kawakami 04] Kawakami, Y., et al., "Investigation of Soft Error Rate Including Multi-Bit Upsets in Advanced SRAM Using Neutron Irradiation Test and 3D Mixed-mode Device Simulation", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 945-948, Dec. 2004.

[Lala 78] Lala, P. K.," An Adaptive Double Error Correction Scheme for Semiconductor Memory Systems," *Digital Processes,* Vol. 4, pp. 237-243, 1978.

[Lin 83] Lin, S., and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

[Maiz 03] Maiz, J., S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multibit Soft Error Events in Advanced SRAMs", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 519-522, Dec. 2003.

[Makihara 00] Makihara, A., et al., "Analysis of Single-Ion Multiple-Bit Upset in High-Density DRAMS", *IEEE Trans. on Nuclear Science*, Vol. 47, No. 6, Dec. 2000.

[Reddy 78] 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.

[Reed 60] Reed, I. S., and G. Solomon,"Polynomial Codes Over Certain Fields",*J. Soc. Ind. Appl. Mat.,* Vol. 8, pp. 300-304, Jun. 1960.

[Satoh 00] Satoh, S., Y. Tosaka, S.A. Wender, "Geometric Effect of Multiple-bit Soft Errors Induced by Cosmic-ray Neutrons on DRAMs", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 310-312, Jun. 2000.

[Vargas 94] Vargas, F. L., and M. Nicolaidis," SEU-Tolerant SRAM Design Based On Current Monitoring", *Proc. Int. Symposium on Fault Tolerant Computing*, pp. 106-115, June 1994.

[Wolf 69] Wolf, J. K.," Adding Two Information Symbols to Certain Non-Binary BCH Codes and Some Applications", *Bell Systems Technical Journal*, Vol. 48, pp. 2405-2424, 1969.