C. L. Chen
M. Y. Hsiao

Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review

This paper presents a state-of-the-art review of error-correcting codes for computer semiconductor memory applications. The construction of four classes of error-correcting codes appropriate for semiconductor memory designs is described, and for each class of codes the number of check bits required for commonly used data lengths is provided. The implementation aspects of error correction and error detection are also discussed, and certain algorithms useful in extending the error-correcting capability for the correction of soft errors such as α-particle-induced errors are examined in some detail.

Introduction

In recent years *error-correcting codes* (ECCs) have been used increasingly to enhance the system reliability and the data integrity of computer semiconductor memory subsystems. As the trend in semiconductor memory design continues toward higher chip density and larger storage capacity, ECCs are becoming a more cost-effective means of maintaining a high level of system reliability [1–4].

A memory system can be made fault tolerant with the application of an error-correcting code; i.e., the mean time between "failures" of a properly designed memory system can be significantly increased with ECC. In this context, a system "fails" only when the errors exceed the error-correcting capability of the code. Also, in order to optimize data integrity, the ECC should have the capability of detecting the most likely of the errors that are uncorrectable.

Error-correcting codes used in early computer memory systems were of the class of *single-error-correcting* and *double-error-detecting* (SEC-DED) codes invented by R. W. Hamming [5]. A SEC-DED code is capable of correcting one error and detecting two errors in a codeword. The double-error-detecting capability serves to guard against data loss. In 1970, a new class of SEC-DED codes called *odd-weight-column* codes was published by Hsiao [6]. With the same coding efficiency, the odd-weight-column codes provide improvements over the Hamming codes in speed, cost and reliability of the decoding logic. As a result, odd-weight-column codes have been widely implemented by IBM and the computer industry worldwide [7–10]. Examples of systems which incorporate these codes are the IBM 158, 168, 303X, 308X, and 4300 series, Cray I, Tandem, etc. There are also various standard part numbers of these codes offered by many semiconductor manufacturers [11] (for example, the AM2960 and AMZ8160 of Advanced Micro Devices, the MC68540 of Motorola, the MB1412A of Fujitsu, and the SN54/74 LS630, LS631 of Texas Instruments).

The number of errors generated in the failure of a memory chip is largely dependent on the chip failure type. For example, a cell failure may cause one error, while a line failure or a total chip failure in general causes more than one error. For ECC applications, the memory array chips are usually organized so that the errors generated in a chip failure can be corrected by the ECC. In the case of SEC-DED codes, the one-bit-per-chip organization is the most effective design. In this organization, each bit of a codeword is stored in a different chip; thus, any type of failure in a chip can corrupt, at most, one bit of the codeword. As long as the errors do not line up in the same codeword, multiple errors in the memory are correctable.

Memory array modules are generally packaged on printed-circuit cards with current semiconductor memory technology, and usually a group of bits from the same card form a portion of an ECC codeword, as illustrated in **Figure 1.** With this

multiple-bit-per-card type of organization, a failure at the card-support-circuit level would result in a byte error, where the size of the byte is the number of bits feeding from the card to a codeword. In this type of configuration, it is important for data integrity that the ECC be able to detect byte errors [12]. A SEC-DED code is in general not capable of detecting all single-byte errors. However, a class of SEC-DED codes capable of detecting all single-byte errors can be constructed [13, 14]. These are called *single-error-correcting double-error-detecting single-byte-error-detecting* (SEC-DED-SBD) codes.

There are certain design applications where the memory array cannot be organized in one-bit-per-chip fashion because of cost or other reasons such as system granularity or power restrictions. As chip density increases, it becomes more difficult to design a one-bit-per-chip memory system. For a multiple-bit-per-chip type of memory organization, a *single-byte-error-correcting double-byte-error-detecting* (SBC-DBD) code [15–20] would be more effective in error correction and error detection.

System reliability generally tends to decrease as the capacity of a memory system increases. To maintain the same high level of reliability, a *double-error-correcting triple-error-detecting* (DEC-TED) code may be used. However, this type of code requires a larger number of check bits than a SEC-DED code and more complex hardware to implement the functions of error correction and error detection [8, 15, 16].

An error-correcting code can be used to correct "soft" errors as well as hard errors. Soft errors are temporary errors such as α-particle-induced errors that disappear during the next memory write operation. With a maintenance strategy that allows the accumulation of hard errors, a high soft error rate would cause a high *uncorrectable error* (UE) rate. To reduce the UE rate that involves soft errors, a SEC-DED code can be modified to correct two hard errors or a combination of one hard and one soft error [21–25].

In this paper we review the current status of error-correcting codes for semiconductor memory applications and present the state of the art by describing the construction of four classes of error-correcting codes suitable for this type of design application. These four classes are SEC-DED codes, SEC-DED-SBD codes, SBC-DBD codes, and DEC-TED codes. For each class of code we provide the number of check bits required for commonly used data lengths, information that is particularly useful to designers for system planning. We also discuss the implementation aspects of error correction and error detection for these classes of error control codes. In addition, we describe a number of algorithms useful in extending the error-correcting capability of codes for the correction of soft errors such as α-particle-induced errors and other temporary errors.

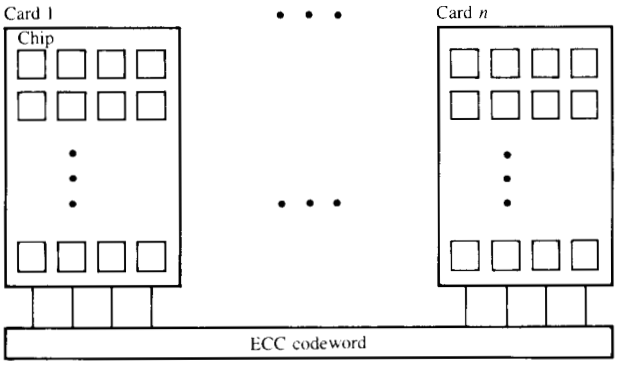


Figure 1 A 4-bit-per-card memory array.

Binary linear block codes

A binary (n,k) linear block code is a k-dimensional subspace of a binary n-dimensional vector space [8, 15, 16]. An n-bit codeword of the code contains k data bits and $r = n - k$ check bits. An $r \times n$ *parity check matrix* $\mathbf{H}$ is used to describe the code. Let $\mathbf{V} = (v_1, v_2, \cdots, v_n)$ be an n-bit vector. Then $\mathbf{V}$ is a codeword if and only if

$$\mathbf{H} \cdot \mathbf{V}' = \mathbf{0}, \tag{1}$$

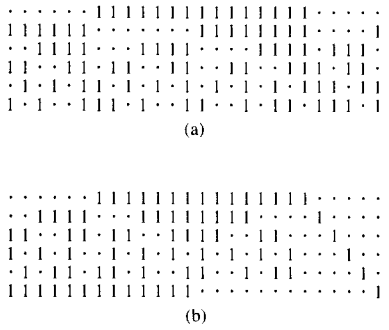
where $\mathbf{V}'$ denotes the transpose of $\mathbf{V}$, and all additions are performed modulo 2.

The *encoding* process of a code consists of generating r check bits for a set of k data bits. To facilitate encoding, the $\mathbf{H}$ matrix is expressed as

$$\mathbf{H} = [\mathbf{P}, \mathbf{I}_r], \tag{2}$$

where $\mathbf{P}$ is an $r \times k$ binary matrix and $\mathbf{I}_r$ is the $r \times r$ identity matrix. Then the first k bits of a codeword can be designated as the data bits, and the last r bits can be designated as the check bits. Furthermore, the ith check bit can be explicitly calculated from the ith equation of the set of r equations in (1). A code specified by an $\mathbf{H}$ matrix of (2) is called a *systematic code.*

Any binary $r \times n$ matrix $\mathbf{H}$ of rank r can always be transformed into the systematic form of (2). Since the rank of $\mathbf{H}$ is r, there exists a set of r linearly independent columns. The columns of the matrix can be reordered so that the rightmost r columns are linearly independent. Applying elementary row operations [16] on the resultant matrix, a matrix of (2) is obtained. The systematic code obtained is equivalent to the code defined by the original $\mathbf{H}$ matrix. **Figure 2(a)** is an example of the parity check matrix of a (26,20) code in a nonsystematic form. Note that the last six columns of the matrix are linearly independent. The submatrix of the six columns can be inverted. The multiplication of the inverse of the submatrix and the transpose of the parity check matrix results in a matrix of systematic form shown in **Figure 2(b).**



(a)

(b)

Figure 2 (26,20) code: (a) nonsystematic form; (b) systematic form.

Table 1 Average number of uncorrectable errors (UEs) with three memory systems employing different error control schemes: parity check, SEC-DED code, and DEC-TED code.

Time ($\times 10^3$ hrs.)	*Parity check*	*SEC-DED*	*DEC-TED*
0–10	49	3.2	0.56
0–20	81	5.2	0.96
0–30	111	6.9	1.3
0–50	168	9.3	2.0
0–80	253	13	2.9

Parity check: (9,8) code.
SEC-DED: (72,64) code.
DEC-TED: (80,64) code.

A word read from the memory may not be the same as the original codeword written in the same location. Let $\mathbf{U} = (u_1, u_2, \cdots, u_n)$ be the word read from the memory. The difference between $\mathbf{U}$ and the original codeword $\mathbf{V}$ is defined as the *error vector* $\mathbf{E} = (e_1, e_2, \cdots, e_n)$; i.e., $\mathbf{U} = \mathbf{V} + \mathbf{E}$. The ith position of $\mathbf{U}$ is in error if and only if e_i is nonzero.

The *decoding process* consists of determining whether $\mathbf{U}$ contains errors and determining the error vector. To determine whether $\mathbf{U}$ is in error, an r-bit *syndrome* $\mathbf{S}$ is calculated as follows:

$$\mathbf{S} = \mathbf{H}\cdot\mathbf{U}' = \mathbf{H}\cdot(\mathbf{V}' + \mathbf{E}') = \mathbf{H}\cdot\mathbf{E}'. \quad (3)$$

If $\mathbf{S}$ is an all-zeros vector, the word $\mathbf{U}$ is assumed to be error-free. If $\mathbf{S}$ is a nonzero vector, it is used to determine the error vector.

The error-correcting capability of a code is closely related to the *minimum distance* of the code. The *weight* of a codeword is the number of nonzero components in the codeword. The *distance* between two codewords is the number of components in which the two codewords differ. The minimum distance d of the code is the minimum of the distances of all pairs of codewords. For a linear code, the minimum distance of the code is equal to the minimum of the weights of all nonzero codewords [8, 15, 16]. A code is capable of correcting t errors and detecting $t + 1$ errors if and only if $d > 2t + 1$.

In semiconductor memory applications, the encoding and the decoding of a code are implemented in a parallel manner. In encoding, the check bits are generated simultaneously by processing the data bits in parallel. In decoding, the syndrome is generated using the same hardware for the generation of the check bits. The error vector is then generated by decoding the syndrome bits in parallel. Finally, the errors are corrected by subtracting the error vector from the fetched word. The subtraction is accomplished by the bit-by-bit exclusive-or (XOR) of the components of the two vectors.

The reliability function of a memory system that employs an error-correcting code can be handled either analytically or through Monte Carlo methods [1–4, 26–28]. For a system with a simple architecture, an analytical approach may be possible. However, for a memory system consisting of hierarchical arrays, the memory reliability function is too intractable to handle analytically. Monte Carlo methods are considered a general approach to study the effectiveness of error-correcting codes and other fault-tolerant schemes [27, 28].

To demonstrate the reliability improvement obtainable with ECC, we consider three memory systems of four megabytes. The first system consists of eight memory cards and is designed with a parity check on each set of eight data bits. The second system consists of 18 memory cards and is designed with a (72,64) SEC-DED code. The last system consists of 20 memory cards and is designed with an (80,64) DEC-TED code. The memory chips for the systems are 16K-bit chips with 128 bit lines and 128 word lines in each chip. Each memory card contains an array of 32×9 chips for the first system, and an array of 32×4 chips for the other two systems. The failure rates of the chips and the card-support circuits are assumed to be the same as those described in [27]. When a UE occurs, the strategy is to replace the card that contains the UE and that has the largest number of defective cells.

The modeling tool of [27] is used to simulate the reliability of the three memory systems. The results of the simulation are shown in **Table 1.** The improvement factor of ECC over the parity check scheme on the number of UEs is over 15 for SEC-DED code and over 84 for DEC-TED code.

SEC-DED codes

The minimum distance of a single-error-correcting and double-error-detecting (SEC-DED) code is greater than or equal to four. Since an n-tuple of weight three or less is not a codeword, from Eq. (1) the sum of a set of three or fewer columns of the $\mathbf{H}$ matrix must be nonzero. In other words,

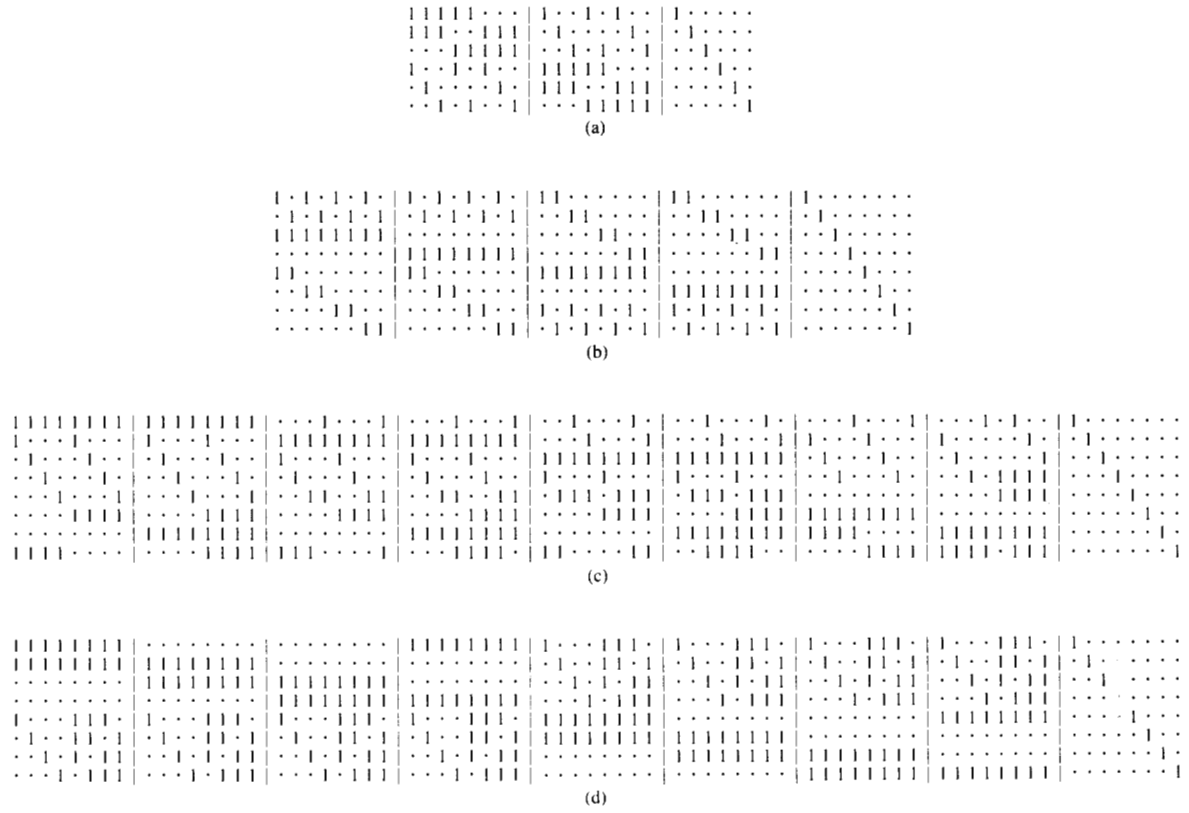


Figure 3 Parity check matrix of some SEC-DED codes: (a) (22,16) code (IBM System/3); (b) (40,32) code (IBM 8130); (c) (72,64) code (IBM 3033); (d) (72,64) code (IBM 3081).

any set of three columns of the **H** matrix are linearly independent. Thus, the **H** matrix of a SEC-DED code must satisfy the following conditions:

A1. The column vectors of the **H** matrix are nonzero and are distinct.
A2. The sum of two columns of the **H** matrix is nonzero and is not equal to a third column of the **H** matrix.

Note that the sum of two odd-weight r-tuples is an even-weight r-tuple. A SEC-DED code with r check bits can be constructed with its **H** matrix consisting of distinct nonzero r-tuples of odd weights. This is an odd-weight-column code of Hsiao [6].

The maximum code length of an odd-weight-column code with r check bits is 2^{r-1}, for there are 2^{r-1} possible distinct odd-weight r-tuples. This maximum code length is the same as that of a SEC-DED Hamming code. The maximum number of data bits k of a SEC-DED code must satisfy $k \leq 2^{r-1} - r$. **Table 2** lists the number of check bits required for a set of data bits. **Figure 3** shows examples of SEC-DED codes used in some IBM systems.

Most of the SEC-DED codes for semiconductor memory applications are *shortened codes* in that the code length is less than the maximum for a given number of check bits. There are various ways of shortening a maximum-length SEC-DED code. Usually a code designer constructs a shortened code to meet certain objectives for a particular application. These objectives may include the minimization of the number of circuits, the amount of logic delay, the number of part numbers, or the probability of miscorrecting triple errors [6].

Table 2 Number of check bits required for SEC-DED codes.

Data bits	*Check bits*
8	5
16	6
32	7
64	8
128	9
256	10

In a write operation, check bits are generated simultaneously by processing the data bits in a parallel manner according to Eqs. (1) and (2). In a read operation, syndrome bits are generated simultaneously from the word read according to Eq. (3). Typically the same XOR tree is used to generate both the check bits and the syndrome bits (see **Figure 4**).

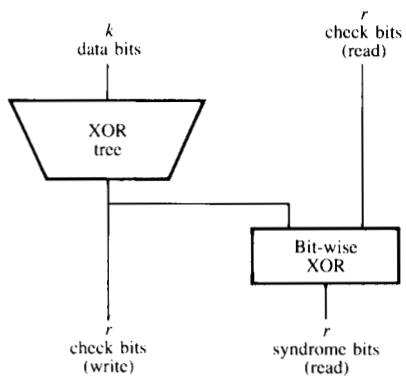


Figure 4 Generation of check bits and syndrome bits.

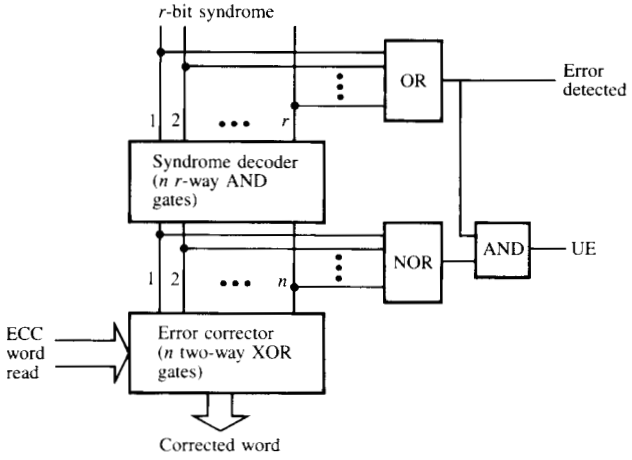


Figure 5 Error detection and correction block diagram.

An algorithm for correcting single errors and detecting multiple errors is described as follows:

1. Test whether **S** is **0**. If **S** is **0**, the word is assumed to be error-free.
2. If $\mathbf{S} \neq \mathbf{0}$, try to find a perfect match between **S** and a column of the **H** matrix. The match can be implemented in n r-way AND gates.
3. If **S** is the same as the ith column of **H**, the ith bit of the word is in error.
4. If **S** is not equal to any column of **H**, the errors are detected as uncorrectable (UE).

This algorithm applied to a SEC-DED code corrects all single errors and detects all double errors. Multiple-bit errors may be detected or falsely corrected. The extent of multiple errors detected depends on the structure of the code.

As shown in **Figure 5**, hardware implementation of the error correction and detection mainly consists of an r-way OR gate for testing nonzero syndrome, n r-way AND gates for decoding syndromes, an n-way NOR gate for generating UE signal, and n two-way XOR gates for inverting the code bit in error. Additionally, an n-bit data register and control logic for timing are required.

A UE signal can also be generated based on the logical OR of the minterms of all UE syndromes. A subset of all UE syndromes is the set of even-weight syndromes caused by even numbers of errors. This subset of syndromes can be recognized by an r-way XOR gate.

The failure of a common logic support in the memory may result in an all-ones or an all-zeros pattern in a codeword. In this case, the error vector in general contains a multiple number of errors that are not detectable by a SEC-DED code. To prevent this kind of data loss, the code can be constructed or modified so that an all-ones or an all-zeros n-tuple is not a codeword. For example, if the check bits are inverted before the codeword is written into the memory, then all the codewords stored in the memory are nonzero. In general, the detection of all-ones and all-zeros errors can be achieved by inverting a subset of the check bits [9].

SEC-DED-SBD codes

In some applications it is required that the memory array chips be packaged in a b-bits-per-chip organization. A chip failure or a word-line failure in this case would result in a byte-oriented error that contains from 1 to b erroneous bits. Byte errors can also be caused by the failures of the supporting modules at the memory card level. The class of SEC-DED codes that are capable of detecting all single-byte errors (SEC-DED-SBD codes) may be used to maintain data integrity in these applications.

The **H** matrix of a SEC-DED-SBD code can be divided into N blocks of $r \times b$ submatrices, $\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_n$, where $\mathbf{B}_i$ represents the parity checks for byte position i. From (3), the syndrome of a byte error at position i is a sum of the columns of $\mathbf{B}_i$ that correspond to the bit error positions within the byte. The syndromes of all possible byte errors at position i are the sum of all possible combinations of the columns of $\mathbf{B}_i$. Let $\langle \mathbf{B}_i \rangle$ denote the sums of all possible nonzero linear combinations of the columns of $\mathbf{B}_i$. Each member of $\langle \mathbf{B}_i \rangle$ should be nonzero and should not be equal to a column of $\mathbf{B}_j$, for $j \neq i$. Otherwise, the byte error at position i will be mistaken as no error or as a correctable single error at position j. Thus, the **H** matrix of a SEC-DED-SBD code must satisfy the conditions A1 and A2 given previously, as well as the following condition:

A3. Each vector of $\langle \mathbf{B}_i \rangle$ is nonzero and is not equal to a column vector of $\mathbf{B}_j$, for $j \neq i$.

For $b \leq 4$, most of the SEC-DED codes for practical applications can be *reconfigured* to detect single-byte errors. The reconfiguration involves the regrouping or rewiring of the

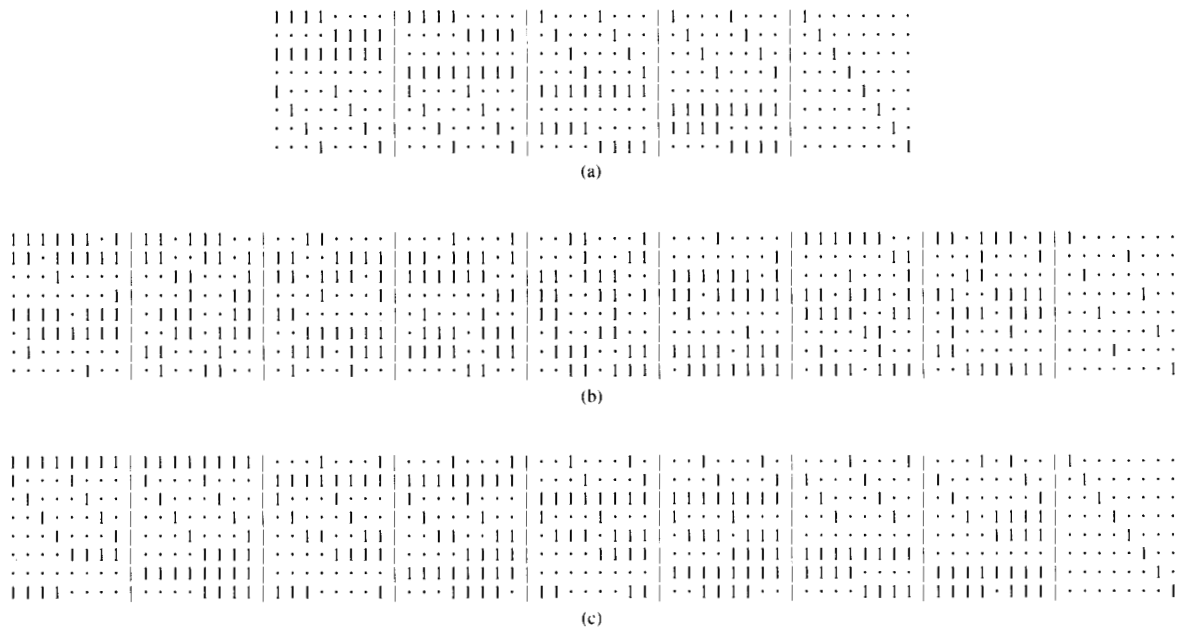


Figure 6 Examples of SEC-DED-SBD codes: (a) (40,32) code, $b = 4$; (b) (72,64) code, $b = 4$; (c) (72,64) code, $b = 3$ and $b = 4$.

bit positions of the original code. Since the same encoding and decoding hardware can be used, no additional hardware is required if a SEC-DED code can be reconfigured for single-byte error detection. **Figure 6** illustrates some examples of SEC-DED-SBD codes. The codes in Figs. 6(a) and (b) are obtained from those in Figs. 3(b) and (d) by reconfiguration, and the code in Fig. 6(c) is the same as that in Fig. 3(c). The (72,64) codes of Fig. 6 are those used in IBM systems 3081 and 3033.

Techniques for the construction of SEC-DED-SBD codes have been presented in [13, 14]. Let $N(r,b)$ be the code length in b-bit bytes. For $b = 3$, it is shown in [14] that optimal codes with $N(r,3) = \lfloor 2^{r-1}/3 \rfloor$, where $\lfloor x \rfloor$ denotes the integer part of x, can be constructed. For other values of b, the construction of the longest code for a given r is an open question. A list of the code lengths of some known SEC-DED-SBD codes is given in **Table 3.**

Table 3 Code length in bytes for some SEC-DED-SBD codes.

r \ b	3	4	5	6	7	8	9
$b+1$	2	2	3	3	3	3	3
$b+2$	5	6	7	8	9	10	11
$b+3$	10	12	15	16	18	20	22
$b+4$	21	26	31	36	41	46	51
$b+5$	42	52	63	72	82	92	102
$b+6$	85	106	127	148	169	190	211

Table 4 All binary 3-tuples expressed as elements of $GF(8)$.

$0 = 0\ 0\ 0$
$x^0 = 1\ 0\ 0$
$x^1 = 0\ 1\ 0$
$x^2 = 0\ 0\ 1$
$x^3 = 1\ 1\ 0$
$x^4 = 0\ 1\ 1$
$x^5 = 1\ 1\ 1$
$x^6 = 1\ 0\ 1$

SBC-DBD codes

For a memory system packaged in a b-bits-per-chip organization, the reliability provided by a SEC-DED code may not be acceptable. To increase the reliability, a byte-oriented error-correcting code may be used [15–20, 29]. In this section, we discuss the construction and implementation of single-byte-error-correcting and double-byte-error-detecting (SBC-DBD) codes.

A codeword of a SBC-DBD code consists of N b-bit bytes. A binary b-tuple is considered an element of the finite field GF(2^b) of 2^b elements [8, 15, 16]. For example, all binary 3-tuples can be assigned as the elements of GF(8), as shown in **Table 4.** In the finite-field representation of b-tuples, the sum of two elements is the bit-by-bit XOR of the two associated b-tuples. The product of two elements X^i and X^j is X^k with $k = i + j \bmod (2^b) - 1$. For example, $X^3 + X^6 = (1\ \ 1\ \ 0) + (1\ \ 0\ \ 1) = (0\ \ 1\ \ 1) = X^4$, and $X^3 \cdot X^6 = X^2$ from Table 4.

With the finite-field representation, an SBC-DBD code is a linear code over GF(2^b) with a minimum distance $d \geq 4$. The

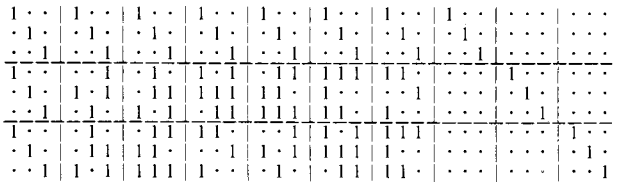


Figure 7 (10,7) SBC-DBD code with $b = 3$.

Table 5 Number of check bits required for SBC-DBD codes.

Bits per byte	*Data bits per ECC word*			
	16	*32*	*64*	*128*
2	8	10	10	12
3	9	12	12	12
4	12	12	14	16
$b > 5$	$3b$	$3b$	$3b$	$3b$

code can also be defined by the parity check matrix **H** of (1) and (2), with the components of the matrices and vectors considered elements of GF(2^b). Let $\mathbf{h}_i$, $1 \le i \le N$, be the column vectors of the **H** matrix. The SBC-DBD code must satisfy the following conditions:

B1. $h_i \neq X \cdot h_j$ for $i \neq j$, $X \in$ GF(2^b).
B2. $h_i + X_1 \cdot h_j \neq X_2 \cdot h_f$, for distinct i, j, f, and $X_1, X_2 \in$ GF(2^b).

Let r be the number of check bytes of an SBC-DBD code over GF(2^b). For $r = 3$, a code of length $N = 2 + 2^b$ bytes can be constructed by extending a Reed-Solomon code of length $(2^b) - 1$ [15–19]. The parity check matrix of the code can be expressed as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \cdots & \mathbf{T}^{2^b-2} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^4 & \cdots & \mathbf{T}^{2(2^b-2)} & \mathbf{O} & \mathbf{O} & \mathbf{I} \end{bmatrix}, \tag{4}$$

where **I** is the $b \times b$ identity matrix, **O** is a $b \times b$ all-zero matrix, **T** is the $b \times b$ companion matrix of X, and X is a *primitive element* of GF(2^b) [15, 16]. If X is a root of the *primitive polynomial* $P(X) = a_0 + a_1X + a_2X^2 + \cdots, + a_{b-1}x^{b-1}$, the companion matrix of X is

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & \cdots & 0 & a_0 \\ 1 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & \cdots & 0 & a_2 \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & \cdots & 1 & a_{b-1} \end{bmatrix}.$$

For example, the companion matrix of X in Table 4 is

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

and the **H** matrix for a (10,7) SBC-DBD code with $b = 3$ is shown in **Figure 7.**

Using the **H** matrix of Eq. (4), the last three column positions of **H** can be designated as the positions of check bytes and the other column positions of **H** can be designated as data byte positions. The check bytes can be generated with an XOR tree just as in the case of SEC-DED codes. The syndrome can also be generated with the same XOR tree. For decoding, the syndrome **S** is divided into three parts, $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$. Each $\mathbf{S}_i$ consists of b bits and represents the parity check equations for the ith row of (4). From (3), if **E** is a single-byte error pattern at data byte position i, then **E** is a unique solution to the following three equations:

$$\mathbf{S}_1 = \mathbf{E}',$$

$$\mathbf{S}_2 = \mathbf{T}^i \cdot \mathbf{E}',$$

$$\mathbf{S}_3 = \mathbf{T}^{2i} \cdot \mathbf{E}'.$$

On the other hand, if **E** is a byte error pattern at check byte position i, where $i = 1$, 2, or 3, then $\mathbf{E} = \mathbf{S}_i'$ and the other two subsyndromes are zeros. The following steps can be taken to find the correctable single-byte error patterns and to detect multiple uncorrectable byte errors.

1. If **S** is a zero vector, assume that there is no error. If **S** is nonzero, go to step 2.
2. If one of the subsyndromes $\mathbf{S}_i \neq \mathbf{0}$, and the other two subsyndromes are zero, $i = 1, 2, 3$, the check byte position i with error pattern S is assumed. Otherwise, go to step 3.
3. Assume that $\mathbf{E} = \mathbf{S}_i'$. Find i that satisfies $0 \le i < N - 4$, $\mathbf{T}^i \cdot \mathbf{E}' = \mathbf{S}_2$, and $\mathbf{T}^{2i} \cdot \mathbf{E}' = \mathbf{S}_3$. If i has a solution, the byte error with pattern **E** at data byte position i is assumed. If i has no solution, then an uncorrectable error is detected.

A block diagram for the generation of the error pointers for the code of Fig. 7 is shown in **Figure 8.**

The extended Reed-Solomon codes defined in Eq. (4) are optimal in that no other SBC-DBD codes with three check bytes contain more data bytes. However, there exists only one code for a given byte size b. When b is small, the code may be too short for memory applications. For example, the code for $b = 2$ can only accommodate six data bits. This code certainly is not practical for most applications. In order to increase the code length for a given b, additional check bits are required.

Techniques for the construction of SBC-DBD codes for $r > 3$ can be found in [15, 16, 30, 31]. **Table 5** lists the minimum number of check bits required for some known SBC-DBD codes.

DEC-TED codes

A memory system with a large capacity or with high chip failure rates may use a double-error-correcting and triple-error-detecting (DEC-TED) code to meet its reliability requirements. A DEC-TED code is also attractive for a memory with a high soft error rate. Although there are schemes [21–25], to be discussed in a subsequent section, for a SEC-DED code to correct hard-hard and hard-soft types of double errors, these schemes cannot correct double soft errors and they require the interruption of a normal memory read operation. With a DEC-TED code, any combination of hard and soft double errors, including double soft errors, can be corrected automatically without system interruption.

A minimum distance of a DEC-TED code is at least equal to six. The parity check matrix **H** of a DEC-TED code must have the property that any linear combination of five or fewer columns of **H** is not an all-zeros vector.

A class of DEC-TED binary linear block codes can be constructed according to the theory of BCH codes [8, 15, 16, 32, 33]. Let X be a root of a primitive binary polynomial $P(X)$ of degree m. The powers of X can be considered elements of GF(N), $N = 2^m$, and can be expressed as binary m-tuples. A binary code defined by (1) with the following parity check matrix is a DEC-TED code:

$$\mathbf{H} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \cdots & \mathbf{1} \\ \mathbf{1} & \mathbf{X} & \mathbf{X}^2 & \cdots & \mathbf{X}^{N-2} \\ \mathbf{1} & \mathbf{X}^3 & \mathbf{X}^6 & \cdots & \mathbf{X}^{3(N-2)} \end{bmatrix}. \tag{5}$$

The powers of X in **H** are expressed in m-tuples. Since there are $2m + 1$ linearly independent row vectors in **H**, the number of check bits of the code is $2m + 1$. The code length is equal to $N - 1$. The code can be extended to length N by adding a column of 1 followed by $2m$ zeros. **Figure 9(a)** shows the parity check matrix of a (31,20) code constructed from Eq. (5).

A full-length BCH code can be shortened by deleting a number of columns from its **H** matrix. The shortened code has a minimum distance at least as large as the original code. The number of check bits of the shortened code may be less than the original code when proper bit positions are deleted [34–35]. In particular, let **Y** be a row vector in the space generated by the row vectors of **H**. Deleting the column positions of **H** where the corresponding positions of **Y** are ones, then the shortened **H** matrix has one fewer linearly independent row vector and the shortened code has one fewer check bit than the original code. **Table 6** presents a list of the number of check bits required for some DEC-TED BCH codes.

The **H** matrix defined by (5) can be transformed into the systematic form of (2) for the generation of check bits (see Fig. 9 for example). Let **H1** be the parity check matrix in systematic form, and **T** be an $r \times r$ transformation matrix that satisfies

$$\mathbf{H} = \mathbf{T} \cdot \mathbf{H1}. \tag{6}$$

The generation of check bits from matrix **H1** can be imple-

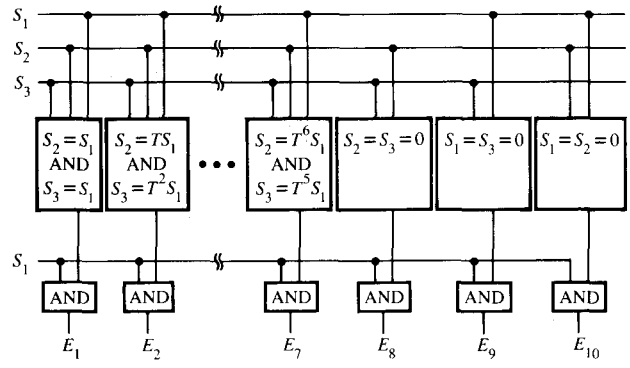


Figure 8 Generation of error vectors for a (10,7) SBC-DBD code with $b = 3$.

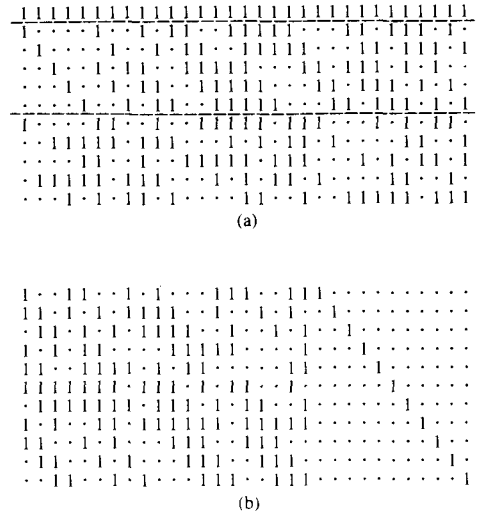


Figure 9 Parity check matrix of a (31,20) code: (a) nonsystematic form **H**; (b) systematic form **H1**.

Table 6 Number of check bits required for DEC-TED BCH codes.

Data bits	*Check bits*
8	9
16	11
32	13
64	15
2^m	$2m + 3$

Table 7 Example of locating erasures.

Direction of stuck faults	*1 0 - - - - - 1*
T_1 (WRITE)	1 1 0 0 1 1 0 0
T_1 (READ)	1 0 0 0 1 1 0 1
T_2 (WRITE)	0 0 1 1 0 0 1 1
T_2 (READ)	1 0 1 1 0 0 1 1
T_1 (READ) + T_2 (READ)	0 0 1 1 1 1 1 0
ERASURE ERROR	1 1 0 0 0 0 0 1

mented with an XOR tree. For decoding, it is convenient to define the syndrome **S** from (3) with the **H** matrix instead of the **H1** matrix. The syndrome can be generated using an XOR tree associated with the **H** matrix. Thus, two separate XOR trees are used to generate check bits and syndrome bits. The syndrome can also be generated by first generating **S1** from Eq. (3) with the **H1** matrix, then multiplying matrix **T** by **S1**. Using this approach, the same XOR tree can be used to generate check bits and **S1**. The validity of this procedure follows directly from Eq. (6).

The syndrome **S** can be divided into three parts, $\mathbf{S}_0$, $\mathbf{S}_1$, and $\mathbf{S}_2$, where $\mathbf{S}_0$ consists of one bit, and $\mathbf{S}_1$ and $\mathbf{S}_2$ consist of m bits. Let the bit positions of the code be assigned as the powers of X. Assume that $\mathbf{E}_1$ and $\mathbf{E}_2$ are the positions of two erroneous bits. Then $\mathbf{S}_0 = 0$ and $\mathbf{S}_1 = \mathbf{E}_1 + \mathbf{E}_2$, $\mathbf{S}_2 = \mathbf{E}_1^3 + \mathbf{E}_2^3$. Since $\mathbf{S}_1^3 + \mathbf{S}_2 = \mathbf{E}_1^2\mathbf{E}_2 + \mathbf{E}_1\mathbf{E}_2^2 = \mathbf{E}_1\mathbf{E}_2\mathbf{S}_1$, the error positions $\mathbf{E}_1$ and $\mathbf{E}_2$ are roots of the quadratic equation

$$\mathbf{S}_1 y^2 + \mathbf{S}_1^2 y + (\mathbf{S}_1^3 + \mathbf{S}_2) = 0. \tag{7}$$

On the other hand, if there is only one error, then $\mathbf{S}_0 = 1$ and the error position is the root of the linear equation $y + \mathbf{S}_1 = 0$.

The major part of the error correction is to find the error positions from the syndrome. Once the error positions are known, the errors are corrected by inverting the data bits at the error positions. The error positions are determined by solving Eq. (7). If $\mathbf{S}_0 = 0$, and Eq. (7) has two solutions, then the solutions are the positions of two errors. If $\mathbf{S}_0 = 1$, and Eq. (7) degenerates to a linear equation, then the solution is the position of a single error. Uncorrectable errors are detected if Eq. (7) has no solution when $\mathbf{S}_0 = 0$, or Eq. (7) does not degenerate into a linear equation when $\mathbf{S}_0 = 1$.

There are various schemes for solving Eq. (7) [36–38]. The equation can be solved algebraically using hardware that implements finite-field operations as in [36]. It can also be solved by substituting all possible solutions into the equation, as in [38]. Another approach is to store the error positions of correctable errors in a table. The syndrome is used as the address to the table of error positions [37].

Extended error correction

Errors in semiconductor memory can be broadly divided into hard errors and soft errors [24, 25]. Hard errors are caused by stuck faults or by permanent physical damage to the memory devices. Soft errors are temporary errors or α-particle-induced errors that will be erased during the next data storage operation. For this discussion, the errors that will stay in their locations during the next few write cycles are considered hard errors.

Error-correcting codes can be used to correct hard as well as soft errors. However, the maintenance strategy for a system may allow the hard errors to accumulate. The presence of errors in the memory increases the probability of uncorrectable errors (UE) due to the lineup of multiple errors in a codeword. The UE rate can be reduced by repair service scheduled periodically. It can also be reduced by extending the conventional error correction to some of the otherwise uncorrectable errors. The latter approach is especially attractive when the soft error rates are high, because it does not require the replacement of memory components. The extended error-correction schemes are discussed in this section.

The errors for which locations but not values are known are called *erasures* [15, 16]. Erasures are easier to correct than random errors. Let t and e be the number of random errors and erasures, respectively, that a code is capable of correcting; then the minimum distance d of the code must satisfy [15, 16],

$$2t + e < d. \tag{8}$$

For example, a SEC-DED code is capable of correcting one random error and one erasure.

In memory applications, the hard errors can be considered erasures if their locations can be identified. To locate the erasures of a particular word in the memory, we may apply some test patterns to the memory. Assume that any binary pattern can be written into the memory. An example is shown in **Table 7** for finding the locations of erasures with two test patterns, T_1 and T_2, of length 8, where T_2 is the complement of T_1. Before the test patterns are written into and read out of the memory, the word originally stored in the memory is read out and stored in a temporary storage. The erasure vector is obtained by the complement of T_1(READ) + T_2(READ). The locations of the erasures are indicated by the ones in the erasure vector. Since T_1 can be arbitrarily chosen, we may also use the word that originally stored in the memory as T_1. This approach for locating the erasures, known as the *double complement algorithm*, saves one write and one read operation. An example of the algorithm is shown in **Table 8.**

Some system designs permit only the codewords to be written into the memory [21, 22, 25]. If the complement of a

codeword is not a codeword, then the approaches just described for the identification of erasures are not applicable. In this case, one solution is to design codes with some special properties [21, 22]. Another solution is to employ three test patterns in locating the erasures [25]. The test patterns are chosen in such a way that they contain at least one 1 and one 0 in every bit position. It can be shown that three test patterns are sufficient to satisfy this condition for any linear code.

Once the locations of the erasures are identified, algorithms can be designed to correct the hard and soft errors, provided that the number of errors satisfies Eq. (8). Assume that the double complement algorithm is applicable for locating the erasures. The following procedure can be used to correct up to two hard errors or a combination of one hard and one soft error for a SEC-DED code:

1. Read word T_1 from a memory location.
2. If a single error in T_1 is detected by the ECC logic, the error in the word is corrected, and the corrected codeword is sent out to its destination.
3. If uncorrectable errors in T_1 are detected by the ECC logic, the complement of T_1 is written into the same memory location. Then the word from the same memory location is read and complemented. Let the resultant word be T_3 (see Table 8).
4. If a single error in T_3 is detected by the ECC logic, the error is corrected. The corrected word is sent out to its destination and is also written into the same memory location.
5. If no error is detected by the ECC logic, T_3 is assumed error free. T_3 is sent out to its destination and is also written into the same memory location.
6. If uncorrectable errors are detected by the ECC logic, the original word is declared uncorrectable.

Note that double soft errors are not correctable by this procedure. All single errors are corrected at the normal speed. The correction of hard-hard and hard-soft types of double errors takes more time because additional write and read operations are involved. The procedure can be modified or refined to correct additional multiple hard errors [21, 24] at the expense of speed and cost. The procedure can also be extended to correct multiple errors beyond the random error-correcting capability of SBC-DBD codes and DEC-TED codes.

The procedure just described derives the information on erasures at the time when the double error occurs. A different method is to store the information on the erasure errors in a table [22]. This approach increases the speed of correcting double errors. However, the table has to be constantly updated to reflect the true status of the erasures in the memory.

There are other schemes for the correction of multiple erasures [39–41]. These schemes involve the design of codes with additional check bits, which are used to mask the erasures in decoding. For example, a (76,64) code can be designed to correct double erasures and single random errors, and to detect double random errors [40].

Table 8 Example of double complement algorithm.

Original word = T_1 (WRITE)	1 1 0 0 1 1 0 0
Hard and soft errors	H – – – – – S –
T_1 (READ)	0 1 0 0 1 1 1 0
T_2 (WRITE) = $\overline{T_1}$ (READ)	1 0 1 1 0 0 0 1
T_2 (READ)	0 0 1 1 0 0 0 1
T_1 (READ) + T_2 (READ)	0 1 1 1 1 1 1 1
Erasure error	1 0 0 0 0 0 0 0
$T_3 = T_2$ (READ)	1 1 0 0 1 1 1 0
Soft error = $T_3 + T_1$ (WRITE)	0 0 0 0 0 0 1 0

Conclusions

Advances in semiconductor technology have brought about very high levels of integration, especially in the memory area where circuit densities are up to 256K bits per chip. In VLSI memory, higher density usually means a reduced signal-to-noise margin. It also increases the likelihood of soft errors due to radiation and other sources. Error-correcting codes have provided a very effective solution to these problems. They have become an essential part of modern memory design. In the future, the ECC could even be an integral part of the memory chips that manufacturers would offer.

In this paper, we have described the essentials of the principal error-correcting codes used in semiconductor memory design applications. The class of SEC-DED codes is currently most widely used throughout the industry. However, more powerful codes such as SBC-DBD and DEC-TED codes are quite likely to be used in future commercial systems.

Acknowledgment

Contributions made by D. C. Bossen are gratefully acknowledged.

References

1. L. Levine and W. Myers, "Semiconductor Memory Reliability with Error Detecting and Correcting Codes," *Computer* **9,** 43–50 (October 1976).
2. B. Richard, "Automatic Error Correction in Memory Systems," *Computer Design* **15,** 179–182 (May 1976).
3. P. K. Lala, "Error Correction in Semiconductor Memory Systems," *Electron. Eng.* **18,** 49–53 (January 1979).
4. A. V. Ferris-Prabhu, "Improving Memory Reliability through Error Correction," *Computer Design* **18,** 137–144 (July 1979).
5. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.* **29,** 147–160 (April 1950).
6. M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM J. Res. Develop.* **14,** 395–401 (July 1970).
7. M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, "Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress," *IBM J. Res. Develop.* **25,** 453–465 (September 1981).

8. S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.
9. G. R. Basham, "New Error-Correcting Technique for Solid-State Memories Saves Hardware," *Computer Design* **15,** 110–113 (October 1976).
10. D. Morris, "ECC Chip Reduces Error Rate in Dynamic RAMS," *Computer Design* **19,** 137–142 (October 1980).
11. D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Digital Equipment Corporation, Bedford, MA, 1982.
12. D. C. Bossen, L. C. Chang, and C. L. Chen, "Measurement and Generation of Error Correcting Codes for Package Failures," *IEEE Trans. Computers* **C-27,** 201–204 (March 1978).
13. S. M. Reddy, "A Class of Linear Codes for Error Control in Byte-per-Package Organized Memory Systems," *IEEE Trans. Computers* **C-27,** 455–458 (May 1978).
14. C. L. Chen, "Error Correcting Codes with Byte Error Detection Capability," *IEEE Trans. Computers* **C-32,** 615–621 (July 1983).
15. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968.
16. W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*, 2nd ed., MIT Press, Cambridge, MA, 1972.
17. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Ind. Appl. Math.* **8,** 300–304 (June 1960).
18. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which are Invariant under the Affine Group and Their Applications," *Info. Control* **11,** 475–496 (November 1967).
19. J. K. Wolf, "Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications," *Bell Syst. Tech. J.* **48,** 2405–2424 (1969).
20. D. C. Bossen, "b-Adjacent Error Correction," *IBM J. Res. Develop.* **14,** 402–408 (July 1970).
21. W. C. Carter and C. E. McCarthy, "Implementation of an Experimental Fault-Tolerant Memory System," *IEEE Trans. Computers* **C-25,** 557–568 (June 1976).
22. C.-E. W. Sundberg, "Erasure and Error Decoding for Semiconductor Memories," *IEEE Trans. Computers* **C-27,** 696–705 (August 1978).
23. P. K. Lala, "An Adaptive Double Error Correction Scheme for Semiconductor Memory Systems," *Digital Processes* **4,** 237–243 (1978).
24. R. Nelson, "Effortless Error Management," *Computer Design* **21,** 163–168 (February 1982).
25. D. C. Bossen and M. Y. Hsiao, "A System Solution to the Memory Soft Error Problem," *IBM J. Res. Develop.* **24,** 390–397 (May 1980).
26. W. K. Mikhail, R. W. Bartoldus, and R. A. Rutledge, "The Reliability of Memory with Single-Error Correction," *IEEE Trans. Computers* **C-31,** 560–564 (June 1982).
27. C. L. Chen and R. A. Rutledge, "Fault-Tolerant Memory Simulator," *IBM J. Res. Develop.* **28,** 184–195 (1984, this issue).
28. M. R. Libson and H. E. Harvey, "A General-Purpose Memory Reliability Simulator," *IBM J. Res. Develop.* **28,** 196–205 (1984, this issue).
29. S. J. Hong and A. M. Patel, "A General Class of Maximal Codes for Computer Applications," *IEEE Trans. Computers* **C-21,** 1322–1331 (December 1972).
30. T. T. Dao, "Design and Implementation of a Non-Binary Code for Byte-Organized Memory with Binary and Quaternary Logics," *Proceedings of the 8th IEEE International Symposium on Multi-Valued Logic*, Rosemont, IL, May 1973, pp. 24–26.
31. S. Keneda and E. Fujiwara, "Single Byte Error Correcting Double Byte Error Detecting Codes for Memory Systems," *IEEE Trans. Computers* **C-31,** 596–602 (July 1982).
32. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Group Codes," *Info. Control* **3,** 68–79 (March 1960).
33. A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres* **2,** 147–156 (1959).
34. J. M. Goethals, "On the Golay Perfect Binary Code," *J. Comb. Theory* **11,** 178–186 (September 1971).
35. C. L. Chen, "On Shortened Finite Geometry Codes," *Info. Control* **20,** 216–221 (April 1972).
36. T. H. Howell, G. E. Gregg, and L. Rabins, "Table Lookup Direct Decoder for Double-Error Correcting BCH Codes Using a Pair of Syndromes," U.S. Patent No. 4,030,067, June 14, 1977.
37. J. T. Yamato and T. K. Tama, "Error Correcting and Controlling System," U.S. Patent No. 4,107,652, August 15, 1978.
38. P. Golan and J. Hlavicka, "New Method for Parallel Decoding of Double-Error Correcting Group Codes," *Proceedings of the 13th International Conference on Fault-Tolerant Computing*, Milan, Italy, June 1983, pp. 338–341.
39. B. S. Tsybakov, "Defects and Error Correction," *Problemy Peredachi Informatsii* **11,** 21–30 (1975).
40. A. V. Kuznetsov, T. Kasami, and S. Yamamura, "An Error Correcting Scheme for Defective Memory," *IEEE Trans. Info. Theory* **IT-24,** 712–718 (November 1978).
41. C. L. Chen, "Linear Codes for Masking Memory Defects," presented at the *IEEE International Symposium on Information Theory*, St. Jovite, Quebec, Canada, September 26–30, 1983.

Received June 30, 1983; revised September 26, 1983

C. L. (Jim) Chen *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Chen is a senior engineer working on error-correcting codes and fault-tolerant memory systems. Before joining IBM in 1974, he held a postdoctoral position at the University of Hawaii and was a faculty member of the University of Illinois. He received his Ph.D. degree in electrical engineering from the University of Hawaii. Dr. Chen is a member of the Institute of Electrical and Electronics Engineers. He has received three IBM Invention Achievement Awards and one IBM Outstanding Innovation Award for his work on error-correcting codes.

M. Y. (Ben) Hsiao *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Hsiao is a senior technical staff member and manager of the Laboratory Engineering Analysis Department. His current professional interests include research and development in computer reliability, availability, serviceability, error-correcting codes, error detection, failure-isolation techniques, and system engineering analysis. He joined IBM in Poughkeepsie in the Advanced Reliability Technology Department in 1960. From 1965 to 1967, he was on educational leave to the University of Florida, after which he returned to IBM as advisory engineer in the Reliability and Diagnostic Engineering Department. In 1969, he was promoted to senior engineer and manager of the Reliability Technology Department. He assumed his present position in 1979. Dr. Hsiao received his B.S. in electrical engineering in 1956 from Taiwan University, Taipei, his M.S. in mathematics in 1960 from the University of Illinois, and his Ph.D. in electrical engineering in 1967 from the University of Florida. He has seven IBM Invention Achievement Awards, two IBM Outstanding Innovation Awards, and a Corporate Award in the areas of error-correction codes, error detection, and failure-isolation techniques. He has authored and co-authored two books published in 1964 and 1968. Dr. Hsiao is a Fellow of the Institute of Electrical and Electronics Engineers and a member of the Fault Tolerant Computing Committee and IFIPS Committee on Reliable Computing and Fault Tolerance.