Linear Sum Codes for Random Access Memories

TOM FUJA, MEMBER, IEEE, CHRIS HEEGARD, MEMBER, IEEE, AND ROD GOODMAN, MEMBER, IEEE

Abstract—Linear sum codes (LSC's) form a class of error control codes designed to provide on-chip error correction to semiconductor random access memories. They use the natural addressing scheme found on RAM's to form and access codewords with a minimum of overhead.

In this paper, we formally define linear sum codes and examine some of their characteristics. Specifically, we examine their minimum distance characteristics, their error correcting capabilities, and the complexity involved in their implementation. In addition, we look closely at an easily implemented class of single, double, and triple-error correcting linear sum codes.

Index Terms—Error control codes, memory fault tolerance, random access memories, redundancy in RAM's, semiconductor memories.

I. INTRODUCTION

Error control codes have been used for years to improve the reliability of random access memory (RAM) systems. Until recently, however, they have been implemented at the board level [1], [11]; typically, each k-bit word in a computer's memory is stored as an n-bit codeword from an (n, k) binary block code, with each bit stored on a different RAM chip.

As RAM chips have become bigger, with capacities in excess of 1 Mbit, the use of on-chip error control is being considered more and more as a means of constructing reliable memory systems. As chips become more dense, soft (or nonpermanent) error sources such as alpha particle radiation [2] and circuit noise can dominate RAM operation and make them useless; a two-tiered combination of chip-level and board-level coding may well prove to be the most effective means of controlling these soft error sources. In addition, recent results [3] have shown that the traditional means of controlling hard (or permanent) defects detected during the manufacturing process—row/column spare switching—may not be effective for very large memory arrays. Thus, the need for powerful, easily implemented codes for on-chip operation may well become great within a few generations of memory devices.

Two different codes have been used to provide on-chip error correction. The first is a bidirectional parity check code developed by Nippon Telephone and Telegraph [4]--[7] for use in 256K, 1M, and 16M RAM's; this code uses a simple product-code-type design to provide single error correction. The other is a shortened Hamming code implemented on an 256K RAM by Micron Technologies of Boise, ID [8].

Linear sum codes (LSC's) form a general class of codes which are suitable for on-chip error correction [9], [10]. They are a generalization of the bidirectional parity check code which allows for multiple error correction. The linear sum codewords are two-dimensional and similar to product codewords; the difference lies in constraints put on the decoding of such codewords to make them more applicable to on-chip error correction. This constraint limits the error correcting capability of the codes to a function of the sum of the minimum distances of the constituent codes rather than the product.

In this paper, we will define linear sum codes and list their major characteristics; in addition, we will examine some practical issues related to the implementation of one, two, and three-error correcting binary linear sum codes. The larger issues concerning on-chip error control—the cost-effectiveness of on-chip versus board-level coding, for instance—are not addressed. At this point, linear sum codes should be regarded as a “tool” capable of increasing the reliability of RAM systems; their role in an “optimal” error control configuration—if such a thing exists—will be determined by a better understanding of the error mechanisms that afflict semiconductor memories.

II. A DESCRIPTION OF LINEAR SUM CODES

In this section, we define what a linear sum code is and show why such codes are suitable for on-chip error protection. We briefly describe some important characteristics of these codes.

A. Code Definition

Consider a $k_2 	imes k_1$ array of $q$-ary memory cells containing an arbitrary pattern of symbols. (For most cases of interest, $q = 2^b$ for some positive integer $b$; for RAM's like those currently used, $q = 2$.) Call this the information array. To each of the $k_2$ rows add $r_1 = n_1 - k_2$ additional symbols such that each row constitutes a codeword from a systematic $(n_1, k_1)$ linear block code over $F_q$. (Here, $F_q$ is the finite field containing $q$ elements.) Similarly, add to each of the $k_1$ columns $r_2 = n_2 - k_2$ parity symbols such that each column forms a codeword from a (possibly different) systematic $(n_2, k_2)$ array.
constructed for the given constituent codes an
linear sum code over
$k_2$) implemented on the word lines of a RAM; that is, the bit-line
constraint on the decoding of such codes; specifically, we
addition, we include in our definition of linear sum codes a
parity) that appear in a product code are not computed. In
array be a function of only the row and column to which that
structurally suitable for implementation in semiconductor
RAM's. These memory chips are arranged in rectangular
array can be correctly decoded as a function of the
information array can be correctly decoded as a function of the
row and column to which it belongs. The decoding constraint tells
estimates of any symbol in the information segment of a linear sum
codeword; that is, in the row code (resp., column code) there is a codeword of weight $d_{\text{min}}^1$ (resp., $d_{\text{min}}^2$) with only one nonzero symbol in its information segment. In this case, (2.1) holds with equality.

**Proof:** See the Appendix.

This lemma gives one reason why we refer to these codes as sum codes (although the next section gives another, more compelling reason). It is worth noting that many important block codes—including parity checks, Hamming codes, and Reed-Solomon codes—have the property described in Lemma 2.1; thus, the minimum distances of sum codes constructed from these block codes can be computed from (2.1).

2) **Error Tolerating Capabilities of Linear Sum Codes:**

The decoding constraint on linear sum codes suggests that minimum distance is not the best measure of a code's performance; minimum distance gives the number of symbols where two codewords differ, but the decoding constraint tells us that the entire codeword is not available to the decoder.

A better parameter is described as follows. Let $b_{ij}$ be the maximum number of errors that can be tolerated in the $i$th row and $j$th column when decoding the $(i, j)$th symbol in the information array; that is, there exists a decoding function that can map an $i$th row/$j$th column combination corrupted by up to $b_{ij}$ errors onto the correct $(i, j)$th entry. (Note we say tolerated as opposed to corrected; we only want to correct the $(i, j)$th entry.) If we let

$$b = \min_{i \leq k_1, j \leq k_2} b_{ij}$$

then $b$ indicates how many errors can be tolerated in any row/column combination. We will call such a code a $b$ error tolerating linear sum code because as long as no row/column combination contains more than $b$ errors, any bit in the information array can be correctly decoded as a function of the row and column to which it belongs.

To see how $b$ is related to the constituent codes, we must examine still another set of codes—those consisting of all possible row/column combinations.

For a given $(n_1, k_1)$ linear row code $C_1$ over $F_q$ and a given $(n_2, k_2)$ linear column code $C_2$, also over $F_q$, let $C_{ij}$ be the set of all permissible $i$th row/$j$th column combinations. (1 $\leq i \leq k_1$, 1 $\leq j \leq k_2$) That is,

$$C_{ij} = \{ c = [a_1, \ldots, a_{n_1-1}, b_1, \ldots, b_{n_2-1}, x]$$

$$\in F_q^{n_1 \cdot n_2-1} : [a_1, \ldots, a_{n_1-1}, x, a_{n_1}, \ldots, a_{n_1-1}] \in C_1, \ldots \}$$

$$[b_1, \ldots, b_{n_2-1}, x, b_{n_2}, \ldots, b_{n_2-1}] \in C_2 \}$$

where $F_q^{n_1 \cdot n_2-1}$ is the set of all $(n_1 + n_2 - 1)$-tuples over $F_q$.

(Note that we are taking the convention that the $(i, j)$th symbol
in the information array is the last entry in the \((n_1 + n_2 - 1)-\)
tuple. 

Thus, for every \((i, j)\) pair, \(C_{ij}\) is an \((n_1 + n_2 - 1, k_1 + k_2 - 1)\) code over \(F_q\). In particular, if we let 
\[
H_1 = [A_1 \cdots A_{n_1}],
\]
and 
\[
H_2 = [B_1 \cdots B_{n_2}]
\]
be, respectively, parity check matrices for the row code and the column code, then 
\[
H_{ij} = \begin{bmatrix}
A_1 & \cdots & A_{i-1} & A_{i+1} & \cdots & A_{n_1} \\
0 & 0 & 0 & 0 & \cdots & 0
\end{bmatrix}
\]
is a parity check matrix of \(C_{ij}\).

Now, let us partition \(C_{ij}\). For each \(s \in F_q\), define 
\[
C_{ij}^s = \{ c \in C_{ij} : c_{1,n_2-1-s} = s \},
\]
the set of all possible \(i\)th row/\(j\)th column combinations that contain the symbol \(s\) at the row/column intersection. It is easy to verify that \(C_{ij}^s\) is a subgroup of \(C_{ij}\) and that \(\{C_{ij}^s : s \in F_q\}\) are the \(q\) cosets of \(C_{ij}\).

Now, suppose we wish to decode the \((i, j)\)th entry in the information array using only the \(i\)th row and the \(j\)th column. The following is a minimum-distance decoding algorithm.

1) Decode the received row/column pair into the closest codeword in \(C_{ij}\). 
2) The estimate \(\hat{w}\) of the \((i, j)\)th entry in the information array is given by \(\hat{w} = \hat{c}_{1,n_2-1}\).

A decoding error will be made only if sufficient symbol errors occur to cause the decoded \((n_1 + n_2 - 1)\)-tuple to be in the wrong coset. Thus, the relevant parameter is not the minimum distance of \(C_{ij}\) but rather the distance between the cosets. We define this parameter as 
\[
d_{ij} = \min_{x \in C_{ij}^0} \min_{y \in C_{ij}} \|x - y\|
\]
where \(H_{ij}\) is a parity check matrix for \(C_{ij}\).

If we let \(w = (r - s)^{-1}(c - y)\), then 
\[
d_{ij} = \min_{w \in H_{ij}^\perp} \|w\| = \min_{w_{n_1 + n_2 - 1} = 1} \|w\|.
\]

Thus, \(d_{ij}\) is equal to the weight of the minimum weight codeword in \(C_{ij}^0\). In fact, it follows that \(d_{ij}\) is equal to the weight of the minimum weight codeword in \(C_{ij}\) for any nonzero \(s\).

Finally, if we define the tolerating distance as 
\[
d_i = \min_{1 \leq i \leq n_1} d_{ij} \quad (2.2)
\]
then the following lemma holds.

**Lemma 2.2:** Consider a linear sum code with tolerating distance \(d_i\) as defined in (2.2). Such a code is capable of tolerating \([d_i - 1/2]\) or fewer errors in any row/column combination.

**Proof:** Suppose we want to decode the \((i, j)\)th bit in the information array. As long as there is a total of at most \([d_i - 1/2]\) errors in the \(i\)th row and \(j\)th column, the codeword in \(C_{ij}\) that is closest to the received row/column pair will be in the same coset of \(C_{ij}^0\) as the codeword that was originally written. Thus, the decoding algorithm described above will yield the correct result. QED

Of course, the definition of \(d_i\) given in (2.2) is not one that is easily computed. This is taken care of in the following lemma.

**Lemma 2.3:** Consider an \((n_1, k_1, n_2, k_2)\) linear sum code over \(F_q\) with tolerating distance \(d_i\). Let \(d_{\text{min}}^\text{row}\) and \(d_{\text{min}}^\text{col}\) be the minimum distances of, respectively, the row code and the column code. Then 
\[
d_i \geq d_{\text{min}}^\text{row} + d_{\text{min}}^\text{col} - 1.
\]

**Proof:** Define \(C_{ij}^i(1 \leq i \leq k_1, 1 \leq j \leq k_2)\) as above. By construction, every codeword in \(C_{ij}^i\) is made up of a nonzero row codeword and a nonzero column codeword. Thus, 
\[
d_{ij} \geq d_{\text{min}}^\text{row} + d_{\text{min}}^\text{col} - 1
\]
for all \(i\) and \(j\) \((1 \leq i \leq k_1, 1 \leq j \leq k_2)\); so, 
\[
d_i = \min_{ij} d_{ij} \geq d_{\text{min}}^\text{row} + d_{\text{min}}^\text{col} - 1.
\]

To get the opposite inequality, let \(c_1\) and \(c_2\) be minimum weight codewords for the row code and the column code, respectively. Let \(\hat{i}\) and \(\hat{j}\) be the positions of the first nonzero components in \(c_1\) and \(c_2\), \(1 \leq \hat{i} \leq k_1, 1 \leq \hat{j} \leq k_2\). Once again, because the constituent codes form linear vector spaces, we can assume that these nonzero components are both the symbol ‘1.’ Construct the codeword \(c^* \in C_{ij}^i\) consisting of \(c_1\) in row \(\hat{i}\) and \(c_2\) in column \(\hat{j}\). Clearly, \(c^* \in C_{ij}^i\), and \(\|c^*\| = d_{\text{min}}^\text{row} + d_{\text{min}}^\text{col} - 1\). Thus, 
\[
d_i \leq d_{ij} - d_{\text{min}}^\text{row} - d_{\text{min}}^\text{col} - 1.
\]

And so, equality holds. QED

Consider, then, the class of linear sum codes whose constituent codes have the property described in Lemma 2.1. This result tells us that, to decode any symbol in the information array of such a code, we can do no better looking at the entire codeword than we can just looking at the row/column to which the desired symbol belongs.
III. A CLASS OF BINARY LINEAR SUM CODES

In this section, we will take a close look at a particularly simple class of binary (i.e., $q = 2$) linear sum codes—those codes which can be constructed from parity checks, Hamming codes, and extended Hamming codes. We will show how these three simple codes can be used to implement single, double, and triple-error tolerating LSC’s.

We will tailor our analysis to issues concerning efficient implementation of these codes. The decoding algorithm given in Section II-B-2 is a very general means of decoding a symbol in the information array of a linear sum codeword; in this section, we will examine easily implemented decoding algorithms which are specific to the codes we will be studying. In addition, we will determine what code rates can be achieved for each of the codes under consideration.

In our analysis, we will make some assumptions about the dimensions of the LSC’s we are studying. Specifically, we assume that the row codes are $(n_1, k_1)$ linear codes and that $k_1 = 2^l_1$ for some nonnegative integer $l_1$; similarly, we assume that the column codes are $(n_2, k_2)$ linear codes with $k_2 = 2^l_2$. These assumptions are made because in a random access memory the rows and columns are defined in terms of a binary address; thus, the number of these rows and columns will always be a power of two.

A. Single-Error Tolerating Binary LSC’s

The simplest linear sum code is one in which the two constituent codes are single-error detecting (SED) parity checks; such codes have minimum distance two, and thus an LSC constructed from two SED codes is a single-error tolerating code since the tolerating distance $d_{t} = 2 + 2 - 1 = 3$. We will refer to such a code as an SED/SED code.

In the binary case, this code is in fact the bidirectional parity check code of [5]-[7]. Nippon Telephone and Telegraph has designed a 256K RAM and a 1M RAM which use this type of code to provide on-chip single-error protection. The advantage of this scheme is its high rate and its simple decoding algorithm. However, it is limited in that it is only capable of correcting a single error in each codeword; thus, in the event of a hard defect in a codeword, there is no soft error protection.

Let us begin our analysis of the SED/SED codes by determining the maximum achievable rate for a fixed information array size. Suppose we want to provide single-error protection. The advantage of a hard defect in a codeword, there is no soft error correcting a single error in each codeword; thus, in the event

we define $R(l_1, l_2)$ to be the rate of an $(n_1, k_1, n_2, k_2)$ SED code with $k_1 = 2^l_1$ and $k_2 = 2^l_2$, then

$$ R^*(L) = \max \{ R(l_1, l_2) : l_1 + l_2 = L \} $$

$$ = \begin{cases} \frac{2^L}{2^L + 2(2^{l_2+1}-1)} & \text{for } L \text{ even} \\ \frac{2^L}{2^L + 3(2^{l_2+1})} & \text{for } L \text{ odd} \end{cases} $$

If there is at most one error in the $i$th row and the $j$th column, $w = \hat{w}$. If there is at most one error in the $i$th row and the $j$th column, $w = \hat{w}$.

B. Double-Error Tolerating Binary LSC’s

From the results presented in Section I, we know that to design a two-error-tolerating linear sum code we must use constituent codes which satisfy

$$ d_{min}^1 + d_{min}^2 \geq 6, $$

where $d_{min}^1$ and $d_{min}^2$ are the minimum distances of the constituent codes. There are two obvious alternatives which satisfy (3.1) with equality:

1) Let both constituent codes be single-error correcting (SEC) Hamming codes ($d_{min} = 3$). We will refer to a code so constructed as an SEC/SEC linear sum code.

2) Let one of the constituent codes be a single-error-correcting/double-error-detecting (SEC-DED) extended Hamming code ($d_{min} = 4$) and let the other be a single-error-detecting (SED) parity code ($d_{min} = 2$). We will refer to such a code as an SEC-DED/SED linear sum code.

In this section, we will analyze these two alternatives. Specifically, we will demonstrate easily implemented decoding algorithms which attain the two-error-tolerating goal, and we will determine what code rates are achievable for a fixed information array size.

1) Decoding Algorithms: In this section, we will describe specific, easily implemented algorithms for decoding a bit in the information array of both an SEC/SEC code and an SEC-DED/SED code. Each of the algorithms described will correctly estimate the $(i, j)$th bit in the information array provided there is at most two errors in the $i$th row and the $j$th column.

a) SEC/SEC Codes: We wish to estimate the $(i, j)$th bit in the information array based on the $i$th row and the $j$th column.
Let \( S_i \) be the syndrome of the \( i \)th row, and let \( S_j \) be the syndrome of the \( j \)th column. For simplicity's sake, let us assume that the syndromes are computed such that they "point" to the estimated error position; that is, \( S_i = m \) for \( 1 \leq m \leq n_1 \) implies that when the \( m \)th bit in the \( i \)th row is inverted, the \( i \)th row constitutes a valid codeword from the row code. Of course, \( S_j = 0 \) implies that the \( j \)th row already is a valid codeword. Similarly define \( S_j \).

Just as in Section III-A, assume that \( w \) was written into the \((i, j)\)th position in the information array and that at a later time we read back a possibly corrupted version \( y \). Our estimate \( \hat{w} \) of \( w \) is then computed according to the following algorithm:

- if \((S_i = 0) \) or \((S_j = 0)\) \( \hat{w} = y \)
- else if \((S_i = j) \) or \((S_j = i)\) \( \hat{w} = y + 1 \)
- else \( \hat{w} = y \).

**b) SEC-DED/SED Codes:** Again we want to estimate the \((i, j)\)th bit in the information array based on the \( i \)th row and the \( j \)th column.

Assume that the SEC-DED code is on the rows of the sum codeword; define \( S_i \) as the syndrome of the \( i \)th row. Just as in the last section, choose the \( S_j \) such it "points" to a single error; in addition, if a double error is detected, then set \( S_j = -1 \). Define \( S_j \) as the modulo-two sum of the \( j \)th column; thus, an odd number of errors in the \( j \)th column will yield \( S_j = 1 \). The following algorithm generates our estimate \( \hat{w} \):

- if \((S_i = j) \) or \((S_i = i)\) \( \hat{w} = y + 1 \)
- else \( \hat{w} = y \).

**2) Achievable Code Rates:** In this section, we will compare the two-error-tolerating binary codes in terms of optimal rate for a fixed information array size.

In our analysis, we assume that the constituent codes are an \((n_1, k_1)\) code and an \((n_2, k_2)\) code, and \( k_1 = 2^{l_1} \) and \( k_2 = 2^{l_2} \) for integer values \( l_1, l_2 \geq 0 \). Thus, there are \( 2^L \) bits in the information array, where \( L = l_1 + l_2 \).

We will analyze both SEC/SEC and SEC-DED/SED codes; define \( Q(l_1, l_2) \) to be the number of redundant cells in an \((n_1, k_1, n_2, k_2)\) two-error-tolerating linear sum code when \( k_1 = 2^{l_1} \) and \( k_2 = 2^{l_2} \). Similarly define \( R(l_1, l_2) \) to be the rate of such a code; thus,

\[
R(l_1, l_2) = \frac{2^{l_1 + l_2}}{2^{l_1 + l_2} + Q(l_1, l_2)}.
\]

Finally, define \( R^*(L) \) to be the highest rate for an \((n_1, k_1, n_2, k_2)\) two-error-tolerating linear sum code when \( k_1 = 2^{l_1} \) and \( k_2 = 2^{l_2} \); and \( L = l_1 + l_2 \) is fixed, i.e.,

\[
R^*(L) = \max \{ R(l_1, l_2) : l_1 + l_2 = L \}.
\]

**a) SEC/SEC Codes:** First consider the case where both the row code and the column code are SEC binary Hamming codes. It can be shown that for \( L \geq 4 \), the optimal rate is achieved by making the information array as square as possible. We state this more precisely in the following lemma.

**Lemma 3.1:** For an SEC/SEC code with \( L \geq 4 \),

\[
R^*(L) = R \left( \frac{L}{2} \right) \cdot \left( \frac{L}{2} \right)
\]

for even \( L \)

\[
= \begin{cases} 
\frac{2^L}{2^L + 1} & \text{for odd } L, \\
\frac{2^L}{2^L + \left( \frac{3L+5}{2} \right)} & \text{for odd } L.
\end{cases}
\]

To prove this, we will make use of the following lemma.

**Lemma 3.2:** For fixed \( L \geq 4 \), consider the even function \( M_i : R \rightarrow R \) defined by

\[
M_i(x) = 2^{(L/2-1)} \left( \frac{L}{2} + x + 1 \right) + 2^{(L/2)} x \left( \frac{L}{2} - x + 1 \right).
\]

Then \( M_i(x) \) is a convex \( U \) function on \([2 - (L/2), (L/2) - 2]\) with a unique minimum at \( x = 0 \).

**Proof:** See the Appendix.

To see why this lemma is important, note that for an \((n, k = 2^2)\) linear block code \( (L \geq 2) \), it takes \( f + 1 \) parity bits to provide single-error correction (i.e., \( n - k = f + 1 \)). For \( L = 2 \), this is just the \((7, 4)\) Hamming code; for \( L \geq 3 \) it is a shortened version of the standard \((2^{L-1} - 1, 2^{L-1} - (L + 1) - 1)\) Hamming code. Thus, for \( l_1, l_2 \geq 2 \)

\[
Q(l_1, l_2) = Q(l_1, l_1) = 2^{l_1}(l_1 + 1) + 2^{l_2}(l_2 + 1),
\]

and so for \( L \geq 4 \)

\[
Q(i, L - i) = M_i \left( \frac{L}{2} - i \right) \text{ for } i = 2, 3, \cdots, L - 2.
\]

**Lemma 3.2** implies that for even \( L \geq 4 \),

\[
\min_{2 \leq i \leq \frac{L}{2} - 2} Q(i, L - i) = M_i \left( \frac{L}{2} - 1 \right),
\]

and for odd \( L \geq 5 \),

\[
\min_{2 \leq i \leq \frac{L}{2} - 2} Q(i, L - i) = M_i \left( \frac{3L+5}{2} \right).
\]

To find \( Q(0, L) \) and \( Q(1, L - 1) \), we note that the relevant codes are the \((3, 1)\) repetition code and the \((5, 2)\) shortened Hamming code. Thus, for \( L \geq 3 \), \( Q(0, L) = 2^{L-1} + L + 1 \) and \( Q(1, L - 1) = 3(2^{L-1}) + 2L \); both of these are greater than \( M_i(0) \) for \( L \) even, \( L \geq 4 \), and both are greater than \( M_i(1/2) \) for \( L \) odd, \( L \geq 5 \). Thus, Lemma 3.1 is proven.

**b) SEC-DED/SED Codes:** It is not as simple to maximize the rate of an SEC-DED/SED code for a fixed information array size \( 2^L \) as it was for the SEC/SEC code. Since the two constituent codes are of a different type, one might suspect that the information array must be skewed in some way to maximize the code rate. This is in fact the case.
and the precise skewing needed is given in the following lemma.

**Lemma 3.3:** Fix $L \geq 2$ and define $m = m(L)$ to be the nonnegative integer satisfying $2^m - m \leq L < 2^m - 1 - (m - 1)$. Then

$$R^*(L) = \frac{L + m}{2},$$

where

$$l_1^* = \left\lfloor \frac{L + m}{2} \right\rfloor$$

and

$$l_2^* = \left\lfloor \frac{L - m}{2} \right\rfloor .$$

Thus,

$$R^*(L) = \frac{2^L}{2^L + 2\lfloor (L + m)/2 \rfloor + 2\lfloor (L - m)/2 \rfloor + 2}.$$

To prove Lemma 3.3, let us first restrict ourselves to the case $l_1 \geq 2$. Since it requires $l_1 + 2$ parity bits to provide single-error correction and double-error detection to $2^{l_1}$ bits when $l_1 \geq 2$, it is easy to see that

$$Q(l_1, l_2) = 2^{l_1 + 2L - l_2} + 2$$

for $l_2 \geq 2$.

Taking this cue, define for $L \geq 2$ a function $N_L : R \rightarrow R$ thusly

$$N_L(x) = 2^x + 2^{L-x}(x+2).$$

To prove Lemma 3.3, it would be nice to show that $N_L(x)$ is convex on the interval $[2, L]$ and that a unique minimum occurs in a small neighborhood around $x = (L + m)/2$. However, since we are really only interested in integer values of $x$, it is sufficient to prove the following easily proven weaker condition.

**Lemma 3.4:** The function $N_L(x)$ defined above satisfies the following inequalities:

$$N_L(x) \leq N_L(x-1) \quad \text{for } x \in \left[3, \frac{L + m}{2} \right]$$

$$N_L(x) \leq N_L(x+1) \quad \text{for } x \in \left[\frac{L + m}{2}, L-1 \right].$$

**Proof:** See the Appendix.

Lemma 3.4 tells us that

$$\min_{2 \leq l_1 < L} Q(l_1, L - l_1)$$

$$= \left\{ Q(l_1^*, l_2^*) \text{ for } L + m \text{ even} \right\}$$

$$= \left\{ \min\{Q(l_1^*, l_2^*), Q(l_1^* + 1, l_2^* - 1)\} \text{ for } L + m \text{ odd} \right\}$$

Simple calculations show that $Q(l_1^*, l_2^*) \leq Q(l_1^* + 1, l_2^* - 1)$; thus,

$$Q(l_1^*, l_2^*) = \min_{2 \leq l_1 < L} Q(l_1, L - l_1).$$

It only remains to show that we can achieve no better rates by letting $l_1 = 0$ or $l_1 = 1$. For $l_1 = 0$, the SEC-DED code is a $(4,1)$ repetition code; thus, $Q(0, L) = 3(2^L - 1)$. For $l_1 = 1$, the SEC-DED code is a $(6,2)$ shortened, extended Hamming code, and so $Q(1, L - 1) = 2^{L-1} + 2$. Both of these are larger than $Q(l_1^*, l_2^*)$ for $L \geq 2$; thus,

$$Q(l_1^*, l_2^*) = \min_{0 \leq l_1 < L} Q(l_1, L - l_1)$$

and so Lemma 3.3 is proven.

One final word should be made about the SEC-DED/SED configurations which achieve $R^*(L)$. Lemma 3.3 tells us that for $L \geq 2$, a $(2^L + 1, 2^L + 1, 2^L + 1, 2^L)$ code attains the best rate possible of any SEC-DED/SED code with $2^L$ information bits. However, it does not say that this configuration is the only one which achieves this rate. A careful reading of the proof of Lemma 3.4 indicates that this configuration is in fact uniquely optimal except when $L$ is of the form $2^m - m$ for integer $m$; in this case, both $(l_1^*, l_2^*)$ and $(l_1^* - 1, l_2^* + 1)$ achieve $R^*(L)$. For instance, in the case $L = 12 = 2^4 - 4$ the best rate can be attained with either an $(n_1 = 266, k_1 = 256, n_2 = 17, k_2 = 16)$ SEC-DED/SED code or an $(n_1 = 137, k_1 = 128, n_2 = 33, k_2 = 32)$ SEC-DED/SED code.

**C. Triple-Error-Tolerating Binary LSC's**

Finally, let us consider an LSC constructed from two binary single-error-correcting, double-error-detecting (SEC-DED) extended Hamming codes, each with a minimum distance of four. Such a sum code has $d_s = 7$ and thus is capable of tolerating three errors, we will refer to such a code as a SEC-DED/SEC-DED binary linear sum code.

We begin our analysis by determining which rates can be achieved for fixed information array size $2^L$. In fact, in this respect, our analysis for SEC-DED/SEC-DED binary codes is essentially identical to our analysis of SEC/SEC codes in Section III-B-2-a. By identical arguments we claim that the maximum achievable rates are attained by letting the information array be as square as possible; that is, if we let $R(l_1, l_2)$ be the rate of an $(n_1, k_1, n_2, k_2)$ binary SEC-DED/SEC-DED code with $k_1 = 2^{l_1}$ and $k_2 = 2^{l_2}$, then the following lemma holds.

**Lemma 3.5:** For $L \geq 4$, $R^*(L) = \max\{R(l_1, l_2) : L - l_1 = L\}$

$$= R \left( \left[ \frac{L}{2} \right], \left[ \frac{L}{2} \right] \right)$$

$$= \begin{cases} \frac{2^L}{2^L + 2\left\lfloor \frac{L}{2} \right\rfloor + 2\left\lfloor \frac{L}{2} \right\rfloor + 1} & \text{for even } L \\ \frac{2^L}{2^L + 2\left\lfloor \frac{3L + 11}{2} \right\rfloor + 2\left\lfloor \frac{L}{2} \right\rfloor} & \text{for odd } L \end{cases}$$

Now consider the decoding of an element in the information array of an SEC-DED/SEC-DED codeword. Assume that $w$
was written into the \((i, j)\)th position in the information array and that a possibly corrupted version \(y\) is read back. Let \(S_i\) and \(S_j\) be, respectively, the syndromes for the \(i\)th row and the \(j\)th column; just as in Section III-B-1-a, assume that the syndromes “point” to a single error and that a double error is indicated by a syndrome of \(-1\). Let \(\hat{w}\) be our estimate of \(w\); then execution of the following algorithm will result in \(\hat{w} = w\) provided there is at most three errors in \(i\)th row and the \(j\)th column:

\[
\text{if } ((S_i = 0) \text{ or } (S_j = 0)) \hat{w} = y \\
\text{else if } ((S_i = j) \text{ or } (S_j = i) \text{ or } (S_i = S_j = 1)) \hat{w} = y \oplus 1 \\
\text{else } \hat{w} = y.
\]

**IV. Implementation and Complexity of Binary Sum Codes**

In this section, we investigate the practical aspects of implementing on-chip binary sum code error correction. We describe the organization and structure of sum-coded RAM’s and assess the complexity of implementing different row/column codes in terms of the number of gates needed.

**A. Implementing Sum Codes**

Fig. 2 shows a possible organization of a one-bit-wide dynamic RAM chip encoded with a linear sum code. The chip consists of a rectangular array of one bit memory cells, sense amplifiers, refresh circuits, selector and addressing logic, as well as the ECC circuitry.

The memory cells are organized as \(m\) \(N\)-bit word lines, each of which uses \(K = k_1 k_2\) cells for data and \(N - K\) cells for parity. The total user capacity of the RAM is therefore \(mK\) bits, which implies \(\log_2 m + \log_2 K\) address bits. These address bits are decoded by a row encoder and a column encoder.

The row decoder uses the \(\log_2 m\) row address bits to enable exactly one of the \(m\) \(N\)-bit word lines. Each read and refresh cycle operates in such a way that all the cells in one complete word line (and hence one complete sum code block) are read out to the sense amps simultaneously.

The column decoder then uses the \(\log_2 K\) column address bits to specify the particular bit being addressed. In doing so, it breaks up these \(\log_2 K\) bits into a group of \(\log_2 k_1\) bits and a group of \(\log_2 k_2\) bits. The \(\log_2 k_1\) bits specify a row in the sum codeword and the \(\log_2 k_2\) bits specify a column. The intersection of these two addresses specifies the one data cell in the word line that is being addressed, and the data selector/multiplexer selects and outputs this bit to the ECC unit.

The shaded portions of Fig. 2 show the areas that represent the ECC overhead. There are essentially only three areas of interest. These are the extra memory cells needed to store the parity checks, the selector block needed to “pull out” the vertical and horizontal component codewords associated with the bit being addressed, (i.e., the row codeword addressed by the \(\log_2 k_1\) bits and the column codeword associated with the \(\log_2 k_2\) address bits). These codewords, together with the addressed data bit are fed to the ECC unit which performs any necessary correction and outputs the corrected bit to the user.

B. Selector Block Complexity

Fig. 3 models the circuitry necessary to extract the component horizontal (row) and vertical (column) codewords of the sum code, in terms of multiplexers. It can be seen that \(n_1 k_2\)-input multiplexers are required to extract the horizontal codeword in which the addressed data bit lies, and \(n_2 k_1\)-input multiplexers are required to extract the vertical codeword. In practice, a multiplexer is implemented very simply by a set of two-input AND gates (one for each MUX input) with their outputs wire-ored together. One input of the AND gate is the data input and the other is the controlling input. This input is normally driven via a decoder from the multiplexer select or address lines. However, in our case, the select lines are simply the RAM column address lines, and these have already been decoded into 1-out-of-\(k_1\) and 1-out-of-\(k_2\) by the column decoders shown in Fig. 2. The selector block is, therefore, simply a collection of two input AND gates which together with the column decoder lines acts as the required multiplexer. It therefore follows that a total of \(n_1 k_2 + n_2 k_1\) two-input AND gates are required to extract the horizontal and vertical component codewords. It also follows that the data selector multiplexer requires \(k_1 k_2 = K\) AND gates which are needed in the uncoded case as well. We can, therefore, ignore this complexity component when comparing different coding schemes for a given value of \(K\).

C. ECC Complexity

1) Syndrome Complexity: The first task the ECC hardware must perform is to compute the horizontal and vertical component code syndromes. This is illustrated in Fig. 4. Each syndrome bit is the XOR sum of various of the selected
In this case, each syndrome bit is the XOR sum of three data bits and one parity bit, indicating that three four-input gates are needed. This in turn implies that each syndrome bit can be computed with three two-input gates. In general, the number of two-input gates needed to compute the syndrome of an \((n, k)\) block code is given by

\[ \|H\| - (n - k) \]

where \(\|H\|\) is the number of 1's in the parity check matrix of the code. The total number of two-input XOR gates needed to implement syndrome computation for an \((n_1, k_2, n_2, k_2)\) linear sum code is therefore

\[ \|H_1\| + \|H_2\| - (n_1 - k_1) - (n_2 - k_2) \]

where \(H_1\) and \(H_2\) are the parity check matrices for the constituent codes.

The three block codes which we used as constituent codes for the class of sum codes discussed in Section III are the SED, SEC, and SEC-DED codes; Hsiao [11] has demonstrated a procedure for constructing such codes such that the number of 1's in the parity check matrices is minimized. Using this procedure, one can arrive at the following values:

**SED:** \(\|H\| = k + 1 = n\)

**SEC:** \(\|H\| = \sum_{i=1}^{k} \left( \begin{array}{c} n-k \ni \end{array} \right) i + \sum_{j=1}^{k} \left( \begin{array}{c} n-k \nj \end{array} \right) j \)

...
to double-error detection. Therefore, in terms of two-input gates, as the single-bit syndrome provides all the needed.

The syndrome indicates an error in a particular bit AND that bit is being addressed. Also, an indication of the all-zero syndrome is output. The lookup table that converts syndromes into error locations is essentially being stored in the interconnections to the AND gates. The only other time any kind of encoding process is required is during a refresh cycle. During refresh, all of the data in the codeword are rewritten, "scrubbing" the data and making it easy to simply recompute all of the parities.

3) Encoder Complexity: Finally, consider the circuits needed to perform a write operation; this part of the RAM must not only write data into memory, but it must also alter the row and column parities so that the word line remains a valid sum codeword. We assume that all write cycles—and indeed all read cycles that reveal an error in the addressed bit—are in fact read-modify-write cycles. This is done so that errors in memory are cleaned out as they are found and not allowed to accumulate.

As mentioned above, the write operation consists of two parts—writing the new data into the addressed location, and making any necessary changes to the parities. This can be achieved as follows. First, the new data bit is compared to the corrected stored data bit. If they are the same, then no parity changes are necessary. If they are different, then all parity bits that check that particular location must be complemented.

Fig. 6 shows such an encoder for one constituent code. A set of \( n - k \) or gates takes its input from the already implemented \( k \) select lines. The interconnection matrix is specified by which data bits are included in which parity checksums; that is, each or gate is associated with a parity bit, and the output of an or gate goes high whenever the associated parity bit is affected by the addressed bit. If the new data are different from the old data, then \( n - k \) AND gates and \( n - k \) XOR gates selectively flip those parity bits that are specified by the outputs of the OR gates.

Thus, to implement the encoder described here requires \( (n_1 - k_1 + n_2 - k_2) \) two-input AND gates, the same number of two-input XOR gates, and \( [\|H_1\| + \|H_2\| - 2(n_1 - k_1) - 2(n_2 - k_2)] \) two-input OR gates.

One more minor point deserves mention. The encoding circuit described here is for writing new data onto a RAM chip. The only other thing any kind of encoding process is required is during a refresh cycle. During refresh, all of the data in the codeword are rewritten, "scrubbing" the data and making it easy to simply recompute all of the parities.

D. Overall Complexity

Having assessed the complexity of the three main overhead areas needed for sum code ECC in terms of AND gates and XOR gates, we can combine these into a single measure. In CMOS technology, the needed AND gates and OR gates can be implemented with four transistors, and the XOR gates with six. Expressing the complexity in terms of transistors then allows us to compare the various coding schemes with a single measure.

V. NUMERICAL RESULTS

In Fig. 7 we have graphed \( R(L) \) for the four codes studied in Section III. In addition, we have listed in Table I the complexity data for the optimal (i.e., best rate) configurations of each of these codes for several values of \( L \).

Several conclusions can be drawn from these data; among them are the following.

- Once the decision is made to place error control on the chip, relatively little is paid in terms of complexity to increase the power of the code; the difference in the number of transistors needed to implement the single-error-tolerating...
code and the double-error-tolerating codes is modest, and the difference between the two- and three-error-tolerating codes is essentially nonexistent.

- There is a penalty in terms of code rate which must be paid for increasing the power of the on-chip code. However, relatively high rates can be achieved with the multiple-error-tolerating codes for reasonable values of $L$; for instance, by placing 256 data bits on each word line, a two-error-tolerating SEC-DED/SED code can be implemented at a rate of 0.744. This rate can be considered acceptable, since there is currently a commercial RAM chip on the market that employs a shortened $(12, 8)$ Hamming code with a rate of 0.667 [8]. The increase in rate is, of course, directly attributable to the increased blocklength of the SEC-DED/SED code, and it demonstrates one of the key tenets of information theory—that it is “better” (in terms of both rate and performance) to build long codewords with strong error correcting capabilities than to build shorter, weaker codewords.

- In comparing the two double-error-tolerating codes, the SEC-DED/SED configuration seems to be preferable to the SEC/SEC implementation. The optimal rates for the SEC-DED/SED configurations range from 10 to 20 percent better than those of the SEC/SEC codes over the most reasonable values of $L$ (i.e., $8 \leq L \leq 12$). The complexity of implementing these codes is essentially the same whether one chooses SEC-DED/SED or SEC/SEC.

VI. SUMMARY

In this paper, the following points were made.

1) A class of error control codes called linear sum codes was defined; these codes are a generalization of the bidirectional parity check code implemented by Nippon Telephone and Telegraph, and their intended use is for error control on the word lines of random access memories. They are similar to product codes in that their codewords are two-dimensional arrays constructed from two constituent codes. They differ from product codes by a constraint put on the decoding of LSC's; this constraint involves taking advantage of the addressing scheme used on RAM’s to minimize the overhead of codeword selection.

2) One particularly simple class of LSC's—those which can be constructed from parity checks, Hamming codes, and extended Hamming codes—were examined in detail. It was shown that codes capable of tolerating one, two, and three errors can be found in this class of codes. Furthermore, simple decoding algorithms were given, and the best achievable code rates were found for these codes.

3) The complexity of actually implementing these codes was examined; this was given in terms of the number of gates required to perform the different selection and computation functions required for error correction. It was shown how the


**Table 1**

| Code | Error Toler. | L | \( R(L) \) | \( n_1 \) | \( n_2 \) | \( n_3 \) | Select. AND's | Syndrome XOR's | Correct. AND's | Encoder XOR's | Encoder OR's | Overall |
|------|--------------|---|-----------|------|------|------|------------|--------------|--------------|-------------|-------------|-------------|---------|
| **SEC-DED** | 1 | 4 | 0.667 | 5 | 4 | 4 | 40 | 16 | 0 | 2 | 2 | 6 | 252 |
| | 6 | 0.800 | 7 | 8 | 8 | 8 | 144 | 16 | 0 | 2 | 2 | 14 | 746 |
| | 8 | 0.889 | 17 | 16 | 16 | 16 | 544 | 32 | 0 | 2 | 2 | 30 | 2920 |
| | 10 | 0.941 | 33 | 32 | 32 | 32 | 2112 | 64 | 0 | 2 | 2 | 62 | 9100 |
| | 12 | 0.970 | 65 | 64 | 64 | 64 | 8320 | 128 | 0 | 2 | 2 | 126 | 134572 |
| | 14 | 0.985 | 129 | 128 | 128 | 128 | 33024 | 256 | 0 | 2 | 2 | 254 | 134668 |
| **SEC-SED** | 2 | 4 | 0.460 | 7 | 4 | 7 | 4 | 56 | 18 | 30 | 6 | 12 | 96 |
| | 6 | 0.500 | 12 | 8 | 12 | 12 | 192 | 36 | 72 | 8 | 8 | 28 | 1664 |
| | 8 | 0.615 | 21 | 16 | 21 | 16 | 672 | 76 | 170 | 10 | 10 | 10 | 4168 |
| | 10 | 0.727 | 39 | 32 | 39 | 32 | 2432 | 40 | 80 | 10 | 10 | 10 | 13004 |
| | 12 | 0.821 | 71 | 64 | 71 | 64 | 9088 | 352 | 704 | 14 | 14 | 14 | 45656 |
| | 14 | 0.889 | 136 | 128 | 136 | 128 | 34616 | 600 | 2064 | 16 | 16 | 16 | 156516 |

*Note: Since \( L = 12 \) is of the form \( 2^n - n \), there are two SEC-DED/SED configurations which achieve \( R^*(L) \) (see Section III-B-2-b); we have included the configuration which minimizes the complexity.*

Complexity varies with the dimensions of the codewords and with the parity check matrices of the constituent codes.

4) Finally, several numerical examples were given for specific codes; this showed that multiple-error-correcting LSC's could be implemented at rates favorable to those of single-error-correcting codes already used in some RAM designs, and that the complexity penalty paid for doing so is not significant.

**Appendix**

Proof of Lemma 2.1: Let \( C_j \), \( 0 \leq j \leq k_1 \), represent a partition of the row code by the number of nonzero symbols in the information symbols; that is, \( C_j \) is a set containing all those codewords in the row code with exactly \( j \) nonzero symbols in the information segment. Similarly, let \( C_i \), \( 0 \leq j \leq k_2 \), be a partition of the column code. Finally, let \( p_1(j) \) and \( p_2(j) \) be the Hamming weight of the "lightest" codewords in \( C_j \) and \( C_i \), respectively.

First, for arbitrary row and column codes, consider any nonzero codeword \( c \). We argue that the Hamming weight \( \| c \| \) is at least \( \mu_1(m) + \mu_2(n) - 1 \). Let \( (i, j) \) be the coordinates of a nonzero element in the information array of \( c \), and let \( m \) and \( n \) be, respectively, the number of nonzero elements in the \( i \)th row and the \( j \)th column of the information array. Then the weight of this codeword \( \| c \| \) is no less than \( p_1(m) + p_2(n) - 1 \). Thus, from the definition of \( \mu_1(m) \) and \( \mu_2(n) \), this is equivalent to assuming that \( p_1(m) = \mu_1(m) \) and \( p_2(n) = \mu_2(n) \). Choose \( c_1 \) and \( c_2 \) to be minimum weight codewords from the row code and column code, respectively, such that \( c_1 \) and \( c_2 \) each contain only a single nonzero symbol in their information parts. That is,

\[
\begin{align*}
&c_1 \in C_{j_1}, \quad \| c_1 \| = \mu_1(m) = \mu_1(m) \\
&c_2 \in C_{j_2}, \quad \| c_2 \| = \mu_2(n) = \mu_2(n)
\end{align*}
\]

Furthermore, because the codes are linear, we can assume that both of these nonzero information symbols are the symbol "1".

Let \( j \) and \( i \) be, respectively, the location of the nonzero entries in the information parts of \( c_1 \) and \( c_2 \). Then by placing a single "1" at the \( (i, j) \)th coordinate and zeros everywhere else in the information array, exactly \( d_{\text{min}} - 1 \) nonzero entries are induced in the parities of the \( i \)th row and \( d_{\text{min}} - 1 \) nonzero entries are induced in the parities of the \( j \)th column. All other elements in the codeword are zero. Thus, the Hamming weight of this codeword is \( d_{\text{min}} + d_{\text{min}} - 1 \), and so

\[
d_{\text{min}} \leq d_{\text{min}} + d_{\text{min}} - 1.
\]

Thus, equality holds.

Proof of Lemma 3.2: From simple calculus,

\[
2^{-l/2}M(L)(x) = 2^{-x} \left[ 1 - \ln \left( \frac{L}{2} + x + 1 \right) \right]^{2^{-l/2}} + x \left[ \ln \left( \frac{L}{2} + x + 1 \right) \right]^{-1}
\]
and
\[ 2^{-L/2}M_t(x) = \ln(2) \left( \frac{L/2 + 1}{2} \ln 2 - 2 \right) + 2^x \left( \frac{L/2 - 1}{2} \ln 2 - 2 \right) \]
\[ \geq \ln(2)(2^{x+2} - 2) \]
for \( x \in [2 - (L/2), (L/2) - 2] \). Thus, \( M_t(x) \) is positive and so \( M_t(x) \) is convex on \([2 - (L/2), (L/2) - 2]\). To see that there is a unique minimum at \( x = 0 \), we just note that \( M_t'(0) = 0 \).

**Proof of Lemma 3.4:** Define \( j = L - (2^n - m) \), so that \( L = 2^m - m - j \) and \( 0 \leq j \leq 2^m - 2 \). Make the change of variable \( t = 2^m - 1 + (j/2) - x \); then we can restate the lemma as follows: Define \( P_t : R \to R \) as
\[ P_t(t) = 2^{2m-1} - t + 2^{m-1} - m - t \]
\[ \cdot \left( 2^{-1} - t + 2 \right) \]
Then \( P_t(t - 1) \leq P_t(t) \) for \( t \in [m - 2^m - 1 - (j/2) + 1, 0] \) and \( P_t(t + 1) \geq P_t(t) \) for \( t \in [0, 2^m - 1 + (j/2) - 3] \).

First take \( t \in [0, 2^m - 1 + (j/2) - 3] \). Then
\[ 2^{-L/2}(P_t(t + 1) - P_t(t)) \]
\[ = 2^{-t} - 2^{m+1} \left( 2^{-m+1} + \frac{j}{2} - t + 1 \right) \]
\[ - 2^{-t} - 2^{m-1} \left( 2^{-m+1} + \frac{j}{2} - t + 2 \right) \]
\[ = 2^{-t} + 2^{m-1} \left( 2^{-m+1} + \frac{j}{2} - t + 1 \right) \]
\[ \geq 0 \]
if and only if \( g(t) = 2^{2m-1} \left( 2^{m-1} + \frac{j}{2} - t + 1 \right) \) is monotone increasing on \([0, 2^m - 1 - (j/2) + 1, 0] \), and so \( h(t) \leq h \) on that interval; therefore, \( P_t(t - 1) \) for \( t \in [m - 2^m - 1 - (j/2) + 1, 0] \). QED

**References**


[6] J. Yamada et al., "A submicron 1 Mbit dynamic RAM with a 4-bit-at-


[10] J. Yamada et al., "A submicron 1 Mbit dynamic RAM with a 4-bit-at-


Maryland, College Park, MD, as an Assistant Professor of Electrical Engineering. He teaches courses in communication systems and error control coding. His research interests include information and coding theory, with particular emphasis on applications regarding computer memories.

Dr. Fuja was an AT&T Bell Laboratories Scholar while at Cornell. He is a member of Tau Beta Pi and Eta Kappa Nu.

Chris Heegard (S'75-M'81) was born in Pasadena, CA, on October 4, 1953. He received the B.S. and the M.S. degrees in electrical and computer engineering from the University of Massachusetts, Amherst, in 1975 and 1976, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1981.

From 1976 to 1978, he was an R & D Engineer at Linkabit Corp., San Diego, CA, where he worked on the development of a packet-switched satellite modem and several sequential decoders for the decoding of convolutional codes. In 1981, he joined the faculty of the School of Electrical Engineering at Cornell University in Ithaca, NY, where he is currently Associate Professor. At Cornell he teaches courses in digital communications, error control codes, information theory, and introduction to digital systems. His current research interests include information and communication theory, algorithms for digital communications, coding for computer memory systems with applications to VLSI memory architectures, and signal processing and error control in optical and magnetic recording systems.

In 1984, Dr. Heegard received the Presidential Young Investigator Award from the National Science Foundation and the IBM Faculty Development Award. He was elected to the Board of Governors of the Information Theory Group of the IEEE in 1986. He is a member of Eta Kappa Nu.

Rod Goodman (M'85) was born in London, England on February 22, 1947. He received the B.Sc. degree in electrical engineering from Leeds University, Yorkshire, England, in 1968. He then spent four years studying for the Ph.D. degree in Electronics at the University of Kent at Canterbury, England.

From 1972 to 1975 he was a lecturer in the Department of Electrical Engineering at Kingston Polytechnic, Surrey, England and he also ran a small electronics company in Canterbury. In 1975 he was awarded the Ph.D. and joined the faculty of the University of Hull as Lecturer. In 1979, he was granted tenure and in 1982 was promoted to Senior Lecturer in Electrical Engineering. In 1985, he joined the faculty of the Department of Electrical Engineering at the California Institute of Technology as Associate Professor. His teaching specialties are digital communications, computer engineering, and VLSI. His research has spanned error control coding, cryptography, medical electronics, information engineering, and expert systems. In addition, he has founded two high technology research and development companies in the U.K. They are Electronic Automation Ltd., a company developing robot vision systems, and Metaforth Ltd., a developer of high-speed computer architectures for real-time AI applications. He has consulted for a wide variety of government and commercial organizations; currently, he is a consultant for the Jet Propulsion Laboratories and Pacific Bell.