SU-SEL-72-031

Design of a Parallel Encoder/Decoder for the Hamming Code, Using ROM

by

H. Mitarai and E. J. McCluskey

June 1972

Technical Report No. 36

This work was supported by the
National Science Foundation under
Grant GJ-27527

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY • STANFORD, CALIFORNIA

SEL-72-030

DESIGN OF A PARALLEL ENCODER/DECODER FOR THE HAMMING CODE, USING ROM

by

H. Mitarai and E. J. McCluskey

June 1972

Technical Report No. 36

DIGITAL SYSTEMS LABORATORY
Department of Electrical Engineering Department of Computer Science
Stanford University
Stanford, California

This work was supported by the National Science Foundation under grant GJ-27527.

Design of a parallel encoder/decoder for the Hamming code, using ROM

by

H. Mitarai

and

E. J. McCluskey
Digital Systems Laboratory
Department of Electrical Engineering
Stanford University

ABSTRACT

ROM implementation of logic circuits which have a large number of inputs is generally considered unwise. However, in the design of an encoder/decoder for the Hamming code, ROM implementation is found to yield many advantages over SSI and MSI implementations. There is a one-to-one correspondence between the partition of H matrix into submatrices and the partition of the set of the inputs to the encoder into subsets of the inputs to the ROM modules. Hence, several methods of partitioning the H matrix for the Hamming code are devised. The resulting ROM implementation is shown to save package count compared with other implementations. However, at the present state of technology, there is a trade-off between speed and package count. In the applications where speed is of the utmost importance, the SSI implementation using ECL logic is the most attractive. The disadvantage of ROM in speed should diminish in the near future when semiconductor memory technology will progress to the point where the slow DTL/TTL gates in the input buffer, the address decoder, and the output buffer of ROM, can be replaced by faster gates.

TABLE OF CONTENTS

LIST OF FIGURES

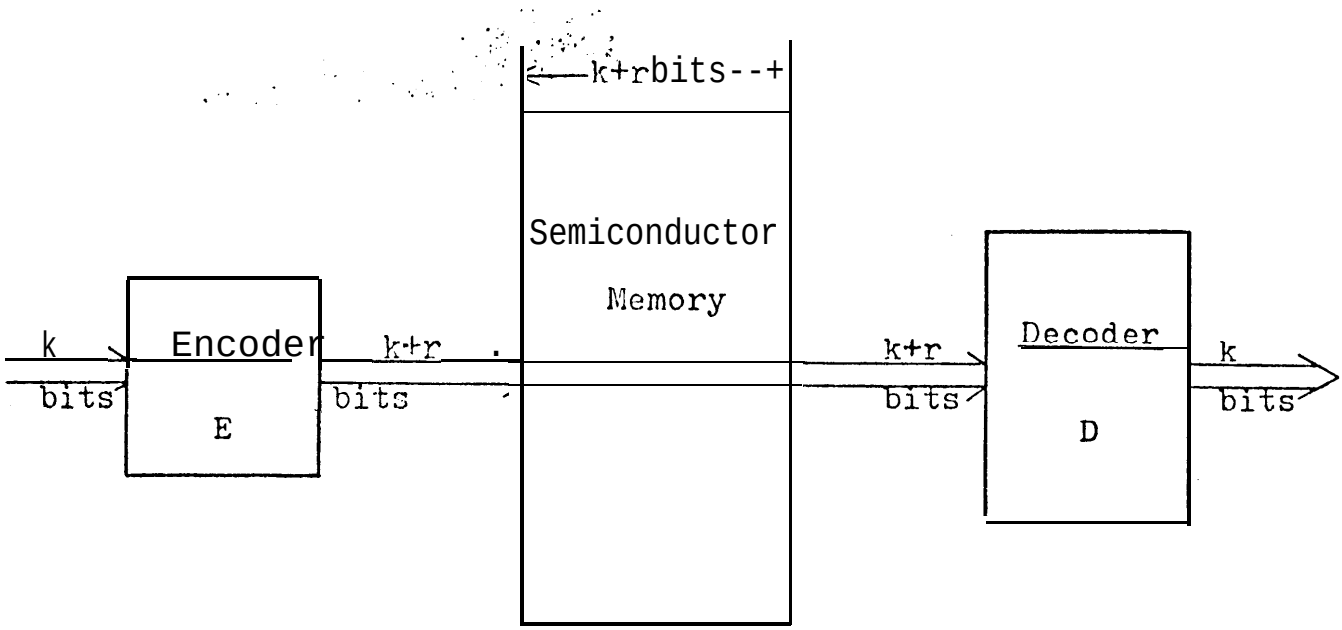
Introduction

The Hamming code is used to increase the reliability of a new semiconductor memory in the IBM 370/145. The semiconductor memory is organized in so-called, "single bit per module organization." In other words, even though a memory word of, say, 72 bits is accessed at each memory cycle, each bit of the word is stored in a physically distinct module. Any kind of component fault, single or multiple, as long as its effect is contained in a module, results in a single bit error. Hence, the Hamming code which is capable of correcting a single bit error, can increase the reliability of the semiconductor memory system. Actually, this idea of single bit per module organization can be extended up to the memory card level. So, even if one memory card is removed for repair, the memory system still works.

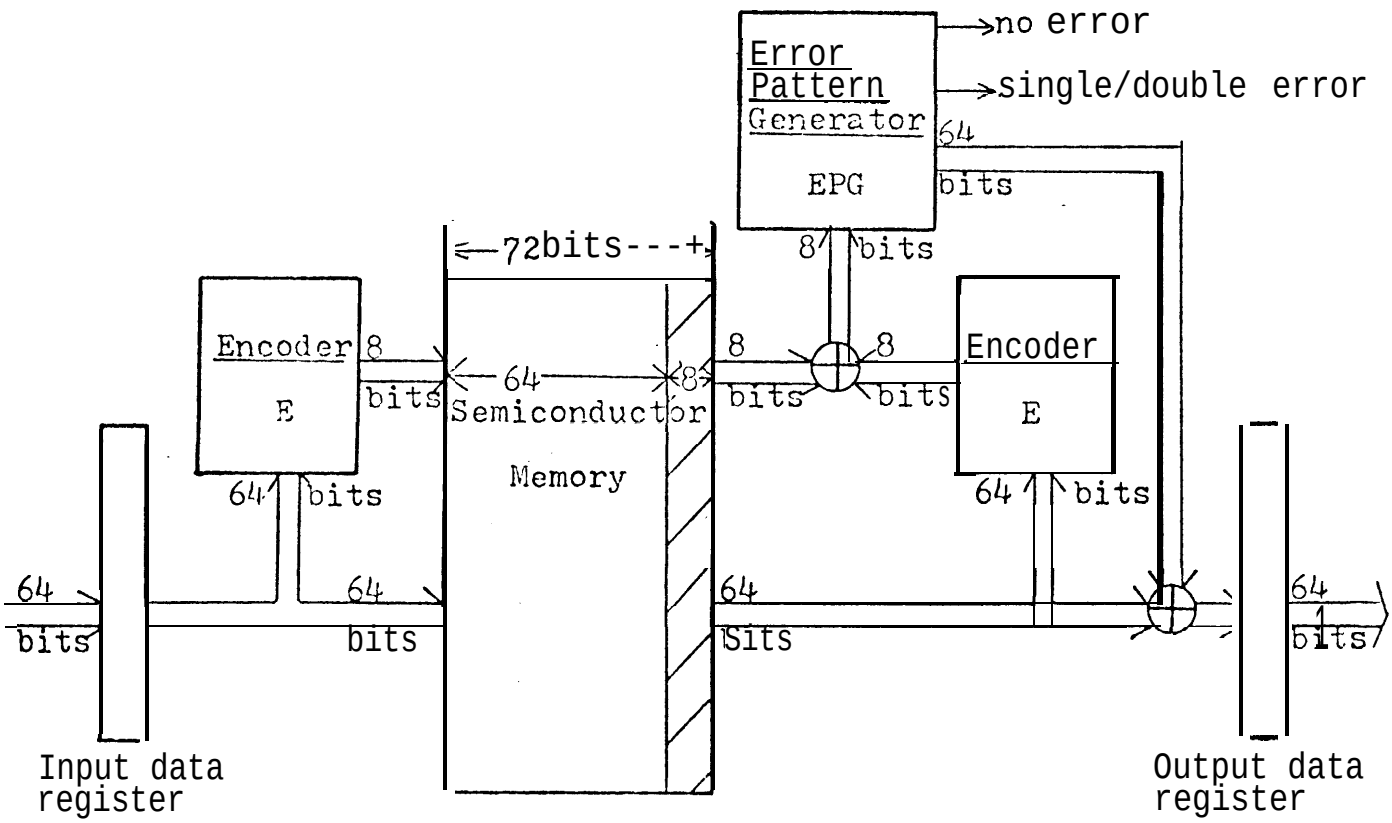
However, the success of this reliability scheme depends on how reliably the encoder/decoder is implemented. It turns out that the encoder alone requires more than 800 NAND gates for a word length of 64 bits. In a logic circuit of this complexity, the reliability problem becomes important, The probability of an encoder/decoder failure is reported to be comparable to the probability of a memory double-bit failure [1]. In this paper, as an effort to improve reliability, semiconductor ROM (Read-Only-Memory) is used as a logic element to minimize the package count of the encoder/decoder.

Implementation of an Encoder/Decoder by ROM

The memory system with an encoder/decoder for the Hamming code attached to it is shown in Fig. 1(a). The numbers of inputs to an encoder and a decoder are k and (k+r) respectively, where k is the word length and r is the number of check bits. The numbers, k and r, are related by the Hamming inequalities:



(a)



* $\oplus$ = exclusive-OR gates

(b)

Fig. 1: Hamming code encoder/decoder for semiconductor memory

$2^{r-1} \geq r+k$ for single error correcting (SEC) codes,

$2^{r-1} \geq r+k$ for single error correcting and double error detecting (SEC/DED) codes.

In Fig. 1(b), a decoder is further divided into two functional blocks: an E block which is identical to an encoder block, and an EPG block or error pattern generator block. It works this way: suppose the word length is 64 bits, then in writing a word into a memory, an encoder block produces 8 check bits and these 8 check bits are stored along with 64 information bits. So, the memory size is increased by 11% redundancy. In reading out a word, an E block encodes over the received word and produces new check bits. These new check bits are compared with old check bits. The disagreement vector, called the syndrome, is then processed by an EPG block either to detect double errors or to pinpoint an erroneous bit if there is any single error.

The previous design of an E block uses exclusive-OR gates [2]. In this design ROM is used to implement an E block. Since ROM has a binary decoder internally, if a single ROM is to implement the E block which has 64 bits at its input, the size of the ROM required is 2^{64} words or 10^{19} words, and prohibitively large. Hence, the set of the inputs to the E block has to be subdivided, so that multiple ROM's of reasonable size can be used.

The problem is a special case of functional decomposition. One approach to the problem, "the input partition method," decomposes the function, F_j, for the check bit, C_j, into functions of subsets of inputs as follows:

$$C_j = F_j(x_1, x_2, \ldots, x_k) = f_j(g_{1j}(x_1, x_2, \ldots, x_m), g_{2j}(x_{m+1} \ldots x_{2m}), \ldots,$$
$$g_{Lj}(x_{k-m+1} \quad \text{[illegible]}) \qquad \text{for } j = 1,2,3,\ldots r,$$

where x_i = i^{th} input variable for i = 1,2,3,...k,

m = number of inputs to ROM,

L = k/m,

g_{aj} = j^{th} output of multiple output function, g_a, for a = 1,2,...L,

f_j = "post-operation functions" (see Fig. 2).

Fortunately in this case, the function of the E block is concisely described by the parity check matrix, commonly called the H matrix. There is a one-to-one correspondence between the partition of H matrix into sub-matrices and the partition of the set of the inputs to the E block into subsets of the inputs to the ROM modules. Effective partitioning of the H matrix results in implementation of multiple output functions, $g_1, g_2, \cdots g_L$, by the identical ROM's without complicating the logic circuits represented by the "post-operation functions," f_j.

Partitioning of the H Matrix

Several check bits are used for the Hamming code. Each check bit "parity-checks" one of the groups of information bits, which are not mutually exclusive to each other. It is then possible to locate an erroneous bit by which combination of groups are indicating parity disagreement. The method of grouping is summarized in the H matrix. For example, in a simple H matrix given in Fig. 3, the first row represents group A in which the check bit #1 parity-checks information bits 5, 6 and 7. In group B, the check bit #2 parity-checks information bits 4, 6 and 7. In group C, the check bit #3 parity-checks information bits 4, 5 and 7. If A and B groups indicate parity disagreement, a corresponding Venn diagram indicates that the bit #6 is in error.

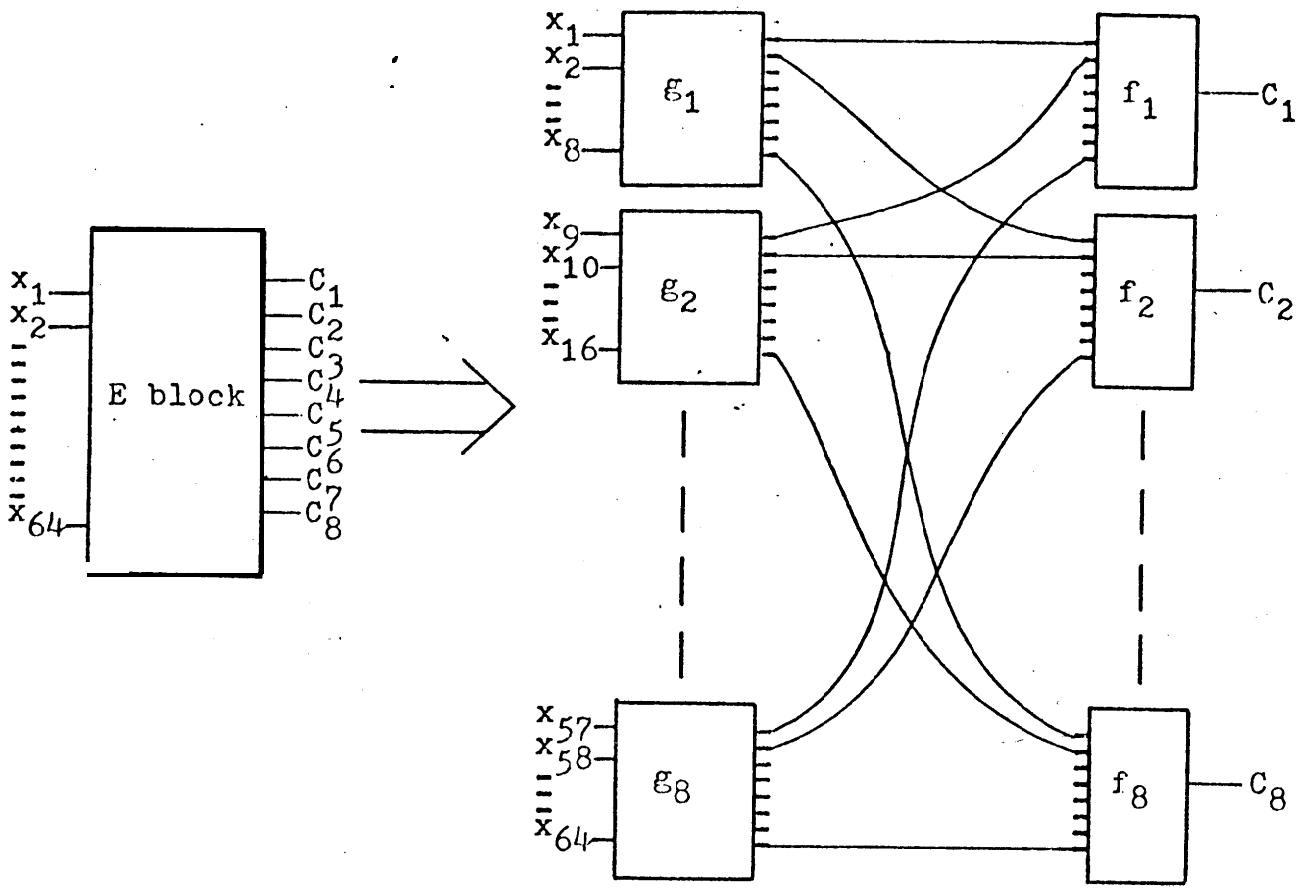


Fig. 2: Functional decomposition by the input partition method, assuming that m=8.

$$H = \begin{array}{c} \\ A \\ B \\ C \\ \\ \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{array}{c} \\ \\ \\ \\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \end{array}$$

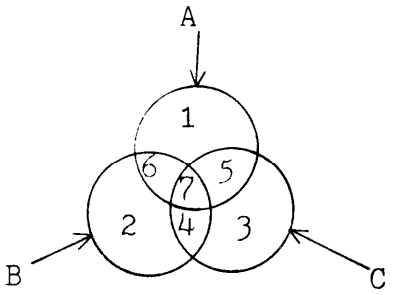


Fig. 3: Simple H matrix and a corresponding Venn diagram

The characteristics of the H matrix are simply summarized in the following rules. It is necessary for SEC codes that:

(1) no column of the H matrix should be all zeros, and

(2) the columns of the H matrix should be distinct.

It is sufficient for SEC/DED codes that, in addition to the above two rules,

(3) the columns of the H matrix should contain only odd numbers of 1's.

Next, a definition is given to show how the H matrix looks after partitioning:

DEFINITION: The partitioned H matrix.

$$H = [I \mid h_1 \mid h_2 \mid h_3 \mid \text{--------} \mid h_L]$$

where the size of H matrix is r by $(r+k)$,

r is the number of check bits,

k is the number of information bits,

I is an r by r identity submatrix,

the size of all other submatrices, $[h_i]$, is r by m,

m is the number of inputs to ROM.

It is assumed in this paper that the number of inputs m to the ROM, is limited to 8 from a practical point of view. An 8 input ROM is now available as a standard product. And also, if number 8 is used, the number of submatrices, L, becomes the word length in bytes. Of course, any number larger than 8 can be used with a slight modification to the following partition methods.

With the above definition, the parity check bits in vector form, $[c_1 c_2 \cdots c_r]^T$, can be expressed in the following matrix form:

$$[c_1 c_2 \cdots c_r]^T = [h_1 \mid h_2 \mid \cdots \mid h_L][x_1 x_2 \cdots x_k]^T$$

$$= [h_1][x_1 x_2 \cdots x_m]^T \oplus [h_2][x_{m+1} \cdots x_{2m}]^T \cdots$$

$$\oplus [h_L][x_{k-m+1} \cdots x_k]^T$$

where all the additions are done in modulo 2,

$[x_1 x_2 \cdots x_k]^T$ is the vector representation of input variables.

Effective partitioning of H matrix is made through the first optimization step.

FIRST OPTIMIZATION STEP: Form "the primary submatrix," $[h_G]$, which can generate other submatrices by the permutation of its rows. In matrix notation, $[h_i] = p_i[h_G]$ for $i = 1, 2, \cdots L$, where p_i is a row-permuting operator. Note that once the H matrix is optimally partitioned, any submatrix can be called a primary submatrix.

Now the parity check equation can be rewritten as

$$[c_1 c_2 \cdots c_r]^T = [h_G][x_1 x_2 \cdots x_m]^T \oplus p_2[h_G][x_{m+1} \cdots x_{2m}]^T \cdots$$

$$\oplus p_L[h_G][x_{k-m+1} \cdots x_k]^T$$

Comparing this equation with that of the functionally decomposed check bit function, C_j, it should be clear that $g_1(x_1,x_2,---,x_m)$ is represented by $[h_G][x_1x_2---x_m]^T$, $g_2(x_{m+1},---x_{2m})$ by $p_2[h_G][x_{m+1},---x_{2m}]^T$, etc. Therefore, as a result of the first optimization step, multiple-output functions, $g_1,g_2,---,g_L$, can be implemented by the identical ROM's, if the row permutation of a submatrix is simply reflected in the permutation of ROM outputs.

It is possible to take the first optimization step, because the number of the columns in the H matrix, $(r+k)$, is usually far less than its maximum value consistent with the Hamming inequality. For example, for the word length of 64 bits $(k=64)$, in order to satisfy the Hamming inequality for SEC/DED codes, r must be at least 8, but then $(k+r)$ could be as large as 128. Thus by proper choice and arrangement of 72 out of 128 possible columns, the H matrix is constructed in a form which can be optimally partitioned. Now the partition problem becomes the problem of finding the primary submatrix and the permutation method.

Methods of Forming a Primary Submatrix

Since a primary submatrix is small in size and not unique, it is easily formed by trial and error. First, try to form more than 8 groups of columns within which each column is related to the other columns by a trial permutation method. The column length, r, should be the minimum value consistent with the Hamming inequality, if possible. The choice of 8 columns in forming a primary submatrix is made through the following optimization.

SECOND OPTIMIZATION STEP: In order to reduce the number of ROM outputs, a primary submatrix should contain

(1) as many rows of all 0's as possible,

(2) as many rows of only one 1 as possible,

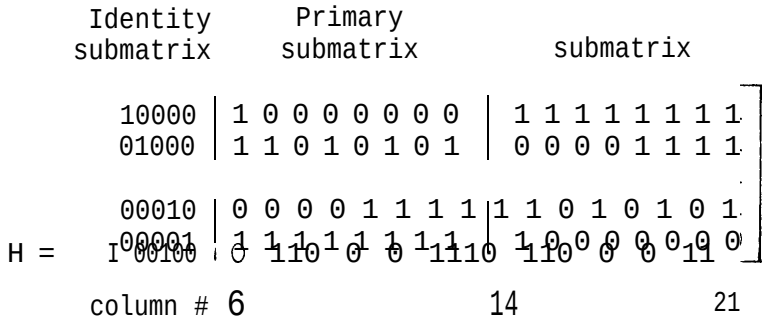
(3) as many identical rows as possible.

In some applications where the parity of 8 bits (or byte) is useful, a primary submatrix should contain a row of all 1's.

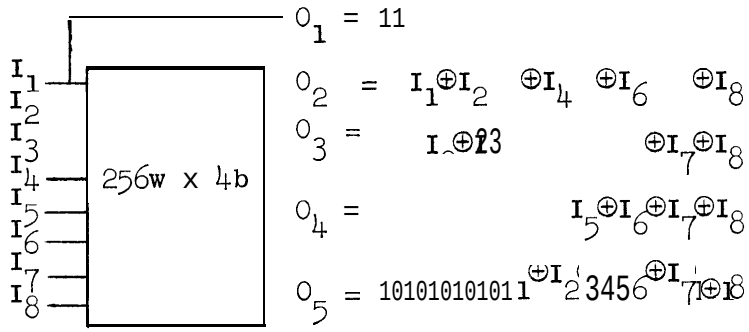
In order to illustrate the two optimization steps, it is probably best to consider the cases for the word length of 16, 32 and 64 bits.

Case (1) The SEC Code for the Half Word Length (k = 16, r = 5):

It is easy to form the 8 columns of a primary submatrix which are not self-symmetric with respect to an (upside-down) inversion operation. For example, $[11011]^T$ is self symmetric, but $[10011]^T$ is not self-symmetric. The other submatrix is obtained simply by inverting the columns of the primary submatrix. As seen in Fig. 4(a), column #14 is the inversion of column #6 and so forth. The primary submatrix implies the input/output relations for the ROM given in Fig. 4(b). ROM may be considered a combinational network where each combination of input variables is an address and where outputs are stored as programmed bits at the address. Since the first row of only one 1 implies that 01 = 11, the number of ROM outputs is reduced by one. Note also that the fifth row of the primary submatrix is deliberately made with all l's so that the corresponding ROM output, 05, generates the byte parity bit.



(a) Partitioned H matrix



(b) Input/output equations for ROM

Fig. 4: Partition of H matrix for the half word length

Case (2) The SEC Code for the Full Word Length (k = 32, r = 6):

The primary submatrix in Fig. 5 generates other submatrices by cyclic permutation. In other words, the first row of the primary submatrix, $[h_G]$, is rotated to the sixth row for the submatrix $[h_2]$, and the second row of the primary submatrix is rotated to the first row for the submatrix $[h_2]$ and so forth for the other rows. Since a cyclic permutation method results in groups of r columns, as long as the number of check bits, r, is greater than the number of submatrices, L, as it is in this case, the primary submatrix shown in Fig. 5 can generate enough submatrices by the cyclic permutation of its rows. Note that the first row of the primary submatrix is deliberately made with all O's so that the number of ROM outputs is 5 instead of 6.

$$H = [I_I | h_G | h_2 | h_3 | h_4]$$

where

$$h_G = \begin{vmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&1&1&1&0&1 \\ 0&1&1&0&1&1&1&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1 \end{vmatrix} \qquad h_2 = \begin{bmatrix} 0&0&0&0&0&0&1&1 \\ 0&0&0&1&1&1&0&1 \\ 0&1&1&0&1&1&1&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

$$h_3 = \begin{vmatrix} 0&0&0&1&1&1&0&1 \\ 0&1&1&0&1&1&1&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \end{vmatrix} \qquad h_4 = \begin{vmatrix} 0&1&1&0&1&1&1&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&1&1&1&0&1 \end{vmatrix}$$

Fig. 5: Partition of the H matrix for the full word length

Case (3) The SEC/DED Code for the Double Word Length ($k = 64$, $r = 8$):

As the number of submatrices exceeds the number of check bits, a row permutation other than cyclic permutation is needed. One way of devising complex permutation methods is to divide the rows of the primary submatrix into two kinds: permuting rows and non-permuting rows. For example, two primary submatrices formed for this particular case are given in Fig. 6. Note that all the columns now have an odd number of l's, because it is necessary to add double-error detection capability to the Hamming code as the word length increases. The first primary submatrix is made up of five permuting rows and three non-permuting rows. The five permuting rows assure that this primary submatrix can generate $\binom{5}{3} = 10$ submatrices; that is, more than 8 submatrices are needed here, and the three non-permuting rows assure that 8 columns within each submatrix are distinct. Note that the first three rows of the primary submatrix are identical. Therefore, one ROM output can fan-out to make up the corresponding 3 outputs. Similarly, the identical 3rd and 4th rows result in saving ROM outputs by 1.

$h_G =$

1 1 1 1 0 0 0 0	5 permuting rows
1 1 1 1 0 0 0 0	
1 1 1 1 0 0 0 0	
0 0 0 0 1 1 1 1	
0 0 0 0 1 1 1 1	
0 0 1 1 0 0 1 1	3 non-permuting rows
0 1 0 1 0 1 0 1	
0 1 1 0 1 0 0 1	

(a)

$h_G =$

1 1 1 1 1 1 1 1	4 permuting rows
1 1 1 1 1 1 1 1	
0 0 0 0 0 0 0 0	
0 0 0 0 0 0 0 0	
1 0 0 1 0 1 1 0	4 non-permuting rows
0 0 0 0 1 1 1 1	
0 0 1 1 0 0 1 1	
0 1 0 1 0 1 0 1	

(b)

Fig. 6: Two examples of the primary submatrices for the double word length

The second primary submatrix consists of 4 permuting rows and 4 non-permuting rows. The 4 non-permuting rows assure that the 8 columns within each submatrix are distinct and of odd weight. The 4 permuting rows are made of identical columns of even weight. This primary matrix can generate $\binom{4}{2} = 6$ submatrices; that is, less than 8 submatrices are needed here. However, all the columns within these 6 submatrices are obviously not self-symmetric with respect to the inversion permutation considered in case (1). Hence 2 x 6 = 12 submatrices can be generated by this primary submatrix, if the permutation method composed of two simple permutations is employed.

The partitioned H matrix corresponding to the second primary submatrix is shown in Fig. 7. Note that the columns of the H matrix satisfy the three rules given previously. The submatrices $[h_2],[h_3],[h_4]$, are generated by permutation of the permuting rows in the primary submatrix, $[h_G]$. The other submatrices $[h_5],[h_6],[h_7]$, and $[h_8]$ are generated by inversion permutation of the submatrices $[h_3],[h_4],[h_G]$, and $[h_2]$ respectively. Cyclic permutation of the non-permuting rows among these submatrices is used not to generate other submatrices but in order to equalize the post-operation functions as will be shown in the next section. The fact that each column in any three out of four non-permuting rows is the simple encoding of its column position within its submatrix will be utilized in implementation of an EPG block.

Implementation of an Encoder Block by ROM

Fig. 8 shows the input/output equations of ROM corresponding to the primary submatrix in Fig. 7. The ROM output 01, which generates the byte parity, can fan out to make up the first two identical rows of the primary submatrix. No ROM output is needed for the next two rows of all 0's. So the number of ROM outputs is reduced from 8 to 5 by the second optimization step. The post-operation function, f_j, is a 6-input parity generator. Note that the row of all 0's in the primary submatrix saves not only the number of ROM outputs but also the number of inputs to the post-operation function. Actually the minimum number of ROM outputs is the number of linearly independent non-zero rows in the primary submatrix. In this particular case, the number of ROM outputs can be as small as 4, because the output O_5 is a linear combination of the other 4 outputs. But if the number of ROM outputs is reduced to 4, the required post-operation function

$$H = [I \mid h_G \mid h_2 \mid h_3 \mid h_4 \mid h_5 \mid h_6 \mid h_7 \mid h_8]$$

where

$$h_G = \begin{bmatrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&0&0&1&0&1&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ k&l&l&l&l&l&l&l \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}$$

$$h_3 = \begin{bmatrix} 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&0&1&0&1&1&0 \\ 0&0&0&0&1&1&1&1 \end{bmatrix} \qquad h_4 = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&0&1&0&1&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \end{bmatrix}$$

$$h_5 = \begin{bmatrix} 0&0&0&0&1&1&1&1 \\ 1&0&0&1&0&1&1&0 \\ 0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix} \qquad h_6 = \begin{bmatrix} 0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1 \\ 1&0&0&1&0&1&1&0 \\ 0&1&0&1&0&1&0&1 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

$$h_7 = \begin{bmatrix} 0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1 \\ 1&0&0&1&0&1&1&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \end{bmatrix} \qquad h_8 = \begin{bmatrix} 1&0&0&1&0&1&1&0 \\ 0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

Fig. 7: Partition of the H matrix for the double word length

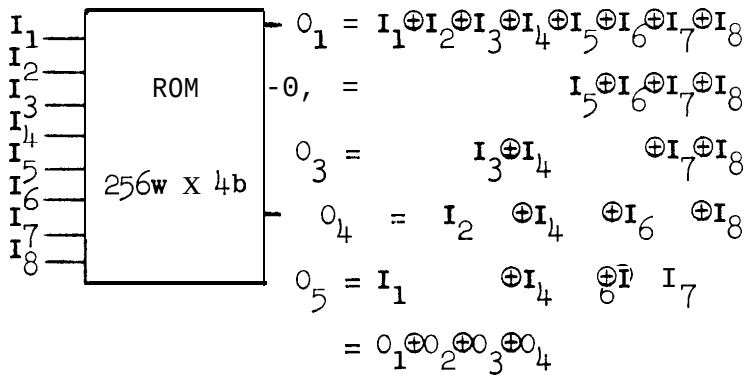
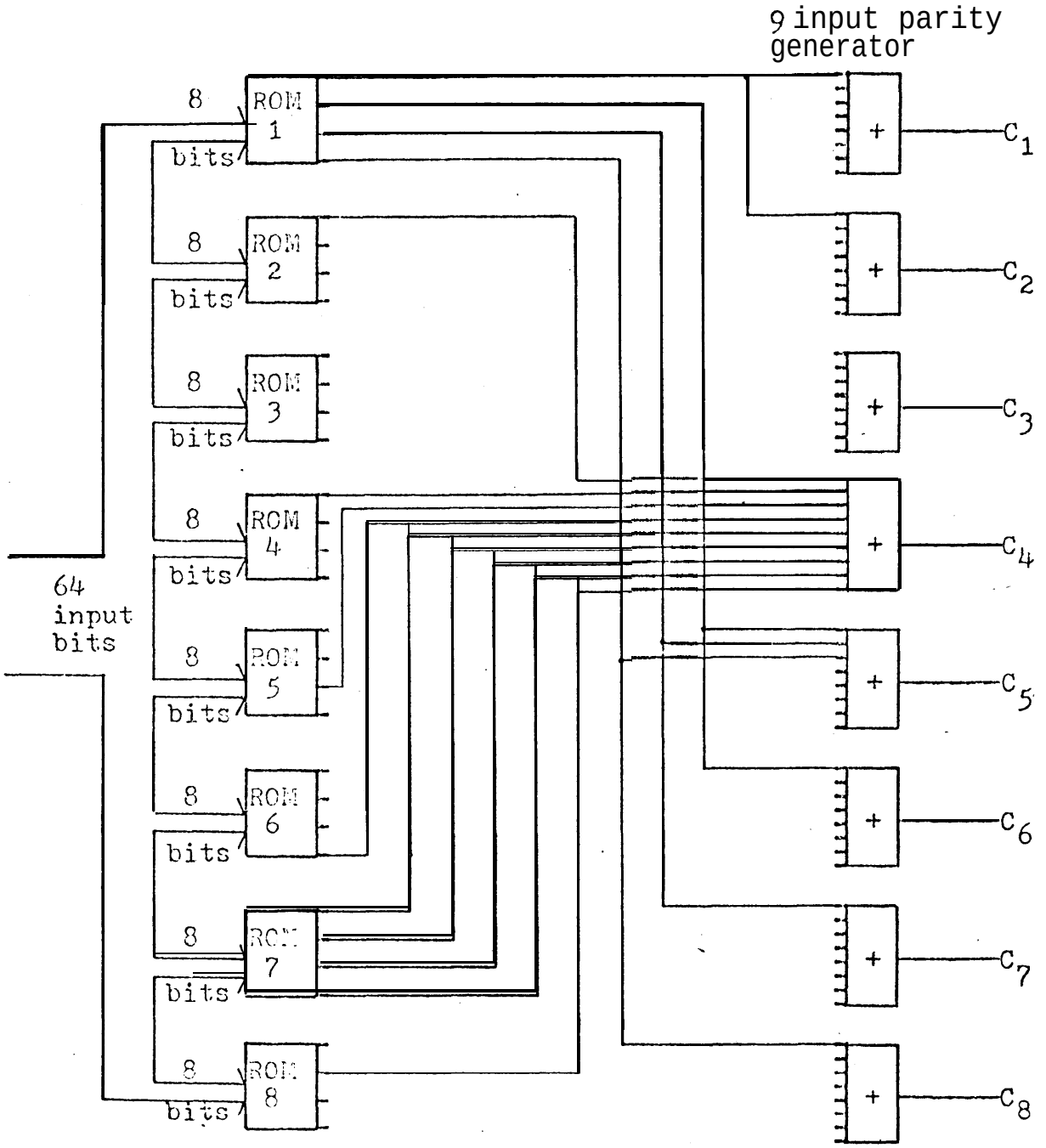


Fig. 8: Input/output equations for ROM, corresponding to the submatrices in Fig. 7

would be a p-input parity generator, and the number of interconnection leads would increase. The final choice of the number of ROM outputs depends on such practical considerations as the size of the available ROM, and the availability of the post-operation function in MSI.

Fig. 9 shows the encoder block corresponding to the partitioned H matrix in Fig. 7. It consists of eight identical ROM's and eight of the p-input parity generators in MSI packages. Not all the wiring is shown, but enough is drawn to show that the row permutation of a primary sub-matrix is reflected in the permutation of ROM output wires.

In Table 1, the results of ROM implementation are summarized and compared with other implementations. In this design, the first objective in using ROM is to minimize package count. Indeed, ROM implementation drastically saves package count as seen here. It is an order of magnitude improvement over small scale integration. Even compared with medium scale integration, ROM cuts package count into half. The small number of package count together with the small number of input leads should improve the reliability of the encoder/decoder for the Hamming code. This



* All ROM's are identical ones of the size 256w X 4b
** Only ¼ of interconnecting lines are shown

Fig. 9: ROM implementation of an Encoder block

advantage is obtained without sacrificing other performances, especially speed. Speed is important, because the delays through the encoder/decoder add directly to memory cycle time.

	ROM	MSI	SSI
	256w x 4b	p-input parity generator	Quadruple 2-input NAND gate
Signetics product number	8226	N74180	N7400
Package count	8ROM+8 MSI	32	208
Number of input leads	136	248	416
Speed (max.)	128 nsec	136 nsec	330 nsec
Power per byte (max.)	1144 mw	1176 mw	1092 mw
Cost per byte (approx.)	$20	$20	$13

Table 1: Comparison of different implementations of an encoder block

In Table 1, speed is calculated under the assumption of using medium speed TTL logic, because ROM also contains similar TTL gates. In the applications where speed is of the utmost importance, speed can be improved up to 45 nsec by using ECL logic of 3 nsec delay per gate. However, most ECL logic circuits are in small scale integration. Therefore, at the present state of technology, there is a trade-off between speed and package

count. The speed of the ROM will certainly be improved because the maximum access time of 60 nsec is accounted for mostly by the delay through 3 levels of slow DTL/TTL gates in the input buffer, the address decoder and the output buffer of the ROM. These gates could be replaced by faster gates with the advent of tri-state logic [3] and with the solution of power dissipation problems in LSI in general. Already the MSI circuit in ECL logic, e.g. a p-input parity circuit (Signetics 10170), is being announced. And an ROM which contains faster gates than the medium speed TTL gates should appear in the near future.

Implementation of an EPG Block

The implementation of an EPG block by a CAM (content-addressable-memory) is conceptually interesting, because what the EPG block does is to locate the column of the H matrix which matches the syndrome. However, the price ratio between ROM and CAM of 1 to 100 makes the use of CAM impractical for the present moment.

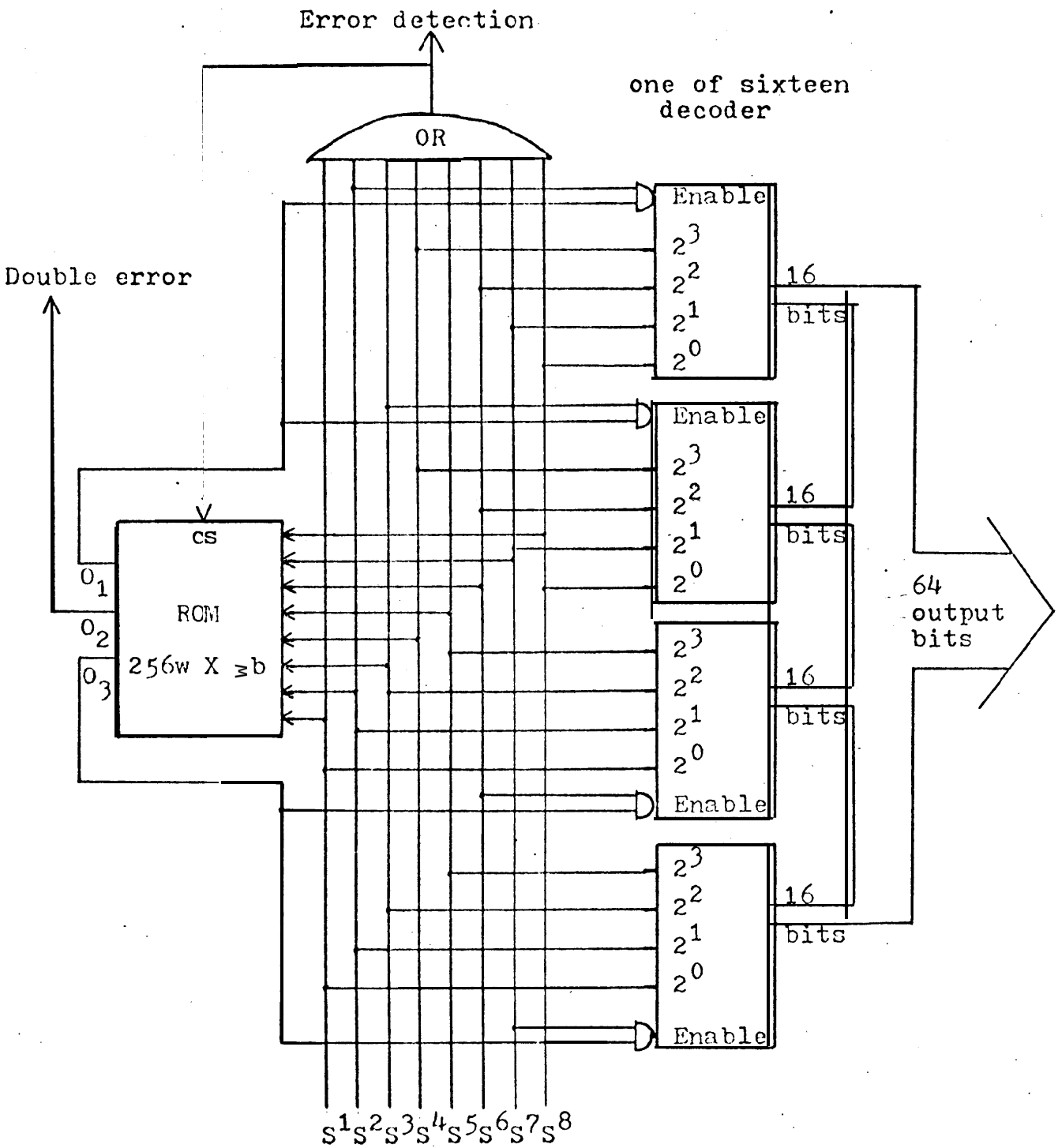
The direct implementation of an EPG block by a single ROM would not be possible, if the number of EPG outputs exceeds the limit of ROM outputs, say, 8. For example, when the word length is 64 bits, as in Fig. l(b), the number of ROM inputs, 8, is within the limit, but at least 8 different ROM's are required to accommodate 64 outputs of an EPG block. Using ROM's without any additional MSI chips for the EPG block does not result in reducing the package count. One possible implementation uses a ROM for reducing 8 syndrome bits to the error position encoded in 6 binary bits. Then the ROM outputs can be decoded by four "one of sixteen decoders" in MSI packages. Since each column of the H matrix is the simple encoding of its column position (see Fig. 7), if the corresponding syndrome bits are

directly decoded to locate an erroneous bit, the size of the ROM would be considerably reduced, as shown in Fig. 10. The ROM output, 0_2, indicates double errors, when the syndrome has an even number of 1's. The other two ROM outputs are used to locate the error position either within the first 32 bits or within the last 32 bits, if there is a single error among the 64 information bits. This is another example where the efficient use of ROM together with MSI circuits results in saving the package count and the number of interconnecting leads. Another reason for the use of ROM in an EPG block is that a pair of ROM's in the E block and the EPG block can be modified accordingly, if a different H matrix is employed.

Error Detection Methods for ROM

The reliability of the encoder/decoder should be improved by the ROM implementation which results in the small number of package count together with the small number of input leads. However, it is still desirable for an error in the encoder/decoder to be detected in order not to transmit incorrect memory words. The duplication-comparison method detects all kinds of errors exclusively confined within either one of two encoder/decoders. When hardware complexity is measured by package count, the duplication-comparison method (which requires a little more than 100% of extra hardware) may become feasible for the ROM implementation.

Under the single fault assumption, it is possible to check an E block and an EPG block by schemes which require much less than 100% hardware redundancy [1]. The error detection capability of the schemes can be enhanced by an additional error check on the ROM. Another advantage in designing an encoder/decoder by ROM is the fact that ROM can be easily checked by the following error detection methods:



* cs= Chip select input

Fig. 1=: ROM implementation of a EPG block

(1) Single Bit Error Check

There are two types of errors associated with ROM under the single fault assumption. One is a single bit error due to cell defect, open bit line, output gate failure, and so forth. Another type is a single module error due to an access error. The single bit error in the ROM output can be detected by adding a parity check bit over the entire output bits.

(2) Access Error Check

A single fault in the address decoder of ROM causes a multiple bit failure which cannot be detected by a parity check'over the outputs alone. A simple method of checking the access unit is to include the address in the output. In order to reduce the number of redundant bits, coding can be applied over the address [4]. The residue code or the modulo M check can be used to check against addressing a wrong word [5]. If M = 2, this is a simple parity check over the address. Higher modulo codes can detect a larger class of failures, but require more check bits and a more complex decoder. To check against addressing multiple words, the address check bits can be further encoded. For example, a simple parity check over the address can be expressed as either (01) or (10). The illegal check bits (00) or (11) indicate that multiple words were read out. In order to minimize the number of interconnecting leads, wired-OR and wired-AND functions are extensively used in ROM. Valid check bits are ANDed or ORed to result in illegal check bits if multiple words are accessed.

(3) Functional Check

The outputs of ROM are often interrelated. In other words, the output word is not encoded in maximum efficiency and so contains redundancy. This fact can help check against a certain class of failures associated with both single bit error and access error.

ROM's in the encoder/decoder are especially easy to check, because one of the ROM outputs is either an even or an odd parity of the address. For example, the ROM in the EPG block has three outputs, 0_2, indicating the even parity of the address, 0_1, enabling the first two decoders, and 0_3, enabling the last two decoders. The only valid output patterns $(0_1,0_2,0_3)$, are $(0,0,0),(1,0,0),(0,1,0)$ and $(0,0,1)$, when ROM is enabled by the chip select input. Hence certain failures are checked functionally or by observing the output pattern. If a simple parity over the outputs is added at the output 0_4 which is available "free" for a 256w x 4b ROM anyway, only the threshold gate which detects the presence of all O's or more than two l's at the ROM output is needed to check against a single bit failure and most of the access failures due to multiple addressing. The output 0_2 should be compared with even parity of the address in order to check against access failures due to wrong addressing.

Conclusion

The H matrix for the Hamming code is effectively partitioned for the ROM implementation of an encoder/decoder. The first optimization step results in the use of only one kind of ROM in an encoder no matter what the word length is. The second optimization step results not only in the same number of ROM's per byte but also in almost the same number of ROM outputs as the one byte case. The resulting ROM implementation is shown to save package count compared with other implementations. However, there is a trade-off between speed and package count. The disadvantage of ROM in speed should diminish in the near future when semiconductor memory technology will progress to the point where the slow DTL/TTL gates in the input buffer, the address decoder, and the output buffer of ROM, can be replaced by faster gates.

The example of ROM implementation just given is rather exceptional. Usually ROM implementation of logic ends up without gaining much advantage over MSI implementation. Successful applications are found only where ROM replaces random logic, as in microprogramming and code conversion. Most logic circuits in a computer are not that random. As a matter of fact, the first impression one gets from reading the literature on ROM implementation is a rather pessimistic one. Since around 1967, the semiconductor people have been advocating ROM for logic [5-11]. And after a few years delay, the response from systems people is starting to appear in computer journals [12-16]. But, they seem to be saying that ROM is difficult to use directly, and suggesting modification to ROM, for example, the Read-Only-Associative Memory. However, it is still interesting to find ways to use ROM as it is, perhaps by modifying computer organization or by using ROM in a special computer where the error detection capability of ROM is fully utilized.

REFERENCES

[1] Carter, W. C., D. C. Jessep and A. Wadia, "Error-free decoding for failure-tolerant memories," Proc. of the 1970 IEEE International Computer Group Conference, pp. 229-239.

[2] Hsiao, M. Y., "A class of optimal minimum odd-weight-column SEC/DED codes," IBM J. Res. and Dev., vol. 14, July 1970, pp. 395-401.

[3] Femling, D., "Enhancement of modular design capability by use of tri-state logic," Computer Design, vol. 10, June 1971, pp. 59-64.

[4] Szygenda, S. A. and M. J. Flynn, "Coding techniques for failure recovery in a distributive modular memory organization," Proc. of the 1971 Spring Joint Computer Conf., pp. 459-466.

[5] Beuscher, H. J. and W. N. Toy, "Check schemes for integrated micro-programmed control and data transfer circuitry," IEEE Trans. on Computers, vol. C-19, Dec. 1970, pp. 1153-1159.

[6] Nichols, J. L., "A logical next step for read-only memories," Electronics, vol. 40, June 12, 1967, pp. 111-113.

[7] Kvamme, F., "Standard read-only memories simplify complex logic design," Electronics, vol. 43, Jan. 5, 1970, pp. 88-95.

[8] Linford, J., "ROM at the top," The Electronic Engineer, vol. 28, May 1969, pp. 64-71.

[9] Hemel, A., "Making small ROM's do math quickly, cheaply and easily," Electronics, vol. 43, May 11, 1970, pp. 104-111.

[10] Uimari , D. C., "Field-programmable read-only memories and applications," Computer Design, vol. 9, Dec. 1970, pp. 49-54.

[11] Leininger, J. C., "The use of read-only storage modules to perform complex logic functions," Proc. of the 1970 IEEE Int. Computer Group Conf., PP. 307-313.

[12] Henle, R. A., I. T. Ho, G. A. Maley and R. Waxman, "Structured logic," Proc. of the 1969 Fall Joint Computer Conf., pp. 61-67.

[13] Wyland, D. C., "Associative read-only memory technique: a key to LSI control logic," Computer Design, vol. 10, Sept. 1971, pp. 61-68.

[14] Flinders, M., P. L. Gardner, R. L. Llewelyn and J. F. Minshull, "Functional memory as a general purpose systems technology," Proc. of the 1970 IEEE Int. Computer Group Conf., pp. 314-324.

[15] Gardner, P. L., "Functional memory and its microprogramming implications," IEEE Trans. on Computers, vol. C-20, July 1971, pp. 764-775.

[16] Thurber, K. J., and R. 0. Berg, "Universal logic modules implemented using LSI memory techniques," Proc. of the 1971 Fall Joint Computer Conf., pp. 177-194.