Error Correction and Detection, a Geometric Approach

R. K. Ward and M. Tabandeh
Department of Electrical Engineering, University of British Columbia, Vancouver, B.C. Canada, V6T 1W5

This paper is basically a tutorial on error detection and correction. The presentation though is via a new approach. This approach has an inherent characteristic leading to parallel hardware implementation and is thus better suited for computer and parallel processing applications. Codes with the desired specifications are constructed by the computer. The design of the codes is within the geometric framework of the Karnaugh map. The mathematics involved is simpler and more intuitive than the traditionally highly mathematical approach. Original results on error detection and conditions for optimal codes are obtained, optimal in the sense of leading to minimum hardware time delays.

1. INTRODUCTION

Error detection and correction is traditionally a highly mathematical subject. The elegant and comprehensive literature on the subject, though, usually overwhelms many engineers and computer scientists who want to use the subject for the design of their specific applications. Luckily for parallel processing, the main applications in computer systems, this subject could be introduced in an intuitive geometrical fashion. The main tool is the Karnaugh map, instead of polynomial rings and Galois fields.

In this paper most of the basic and relevant theory will be reintroduced via this new approach. Reintroduction of the subject as such will also lead into further novel results. Results known for SEC-DED (single-error correcting double-error detecting) codes are found to be easily generalized to higher order t-error correcting $(t+1)$-error detecting codes. Also conditions which these higher order codes must satisfy, so that they result in minimum time delay, in hardware implementation, are obtained.

Error codes in use at present in computer systems have been adapted from codes developed in the past for communication systems. The constraints in many computer applications though, differ from those in communications. Generally, information processing should be handled in parallel instead of serially (as is done in communication systems), because the time allowed for coding and decoding is more crucial. A new approach which constructs codes with the above constraints as part of the design uses the Karnaugh map. This novel and simple approach extends to t-error correcting and detecting codes for all t. Results comparable in size to the best binary linear known codes are obtainable.

This approach has the following merits: first, the theory does not involve more than the knowledge of the simple Karnaugh maps and binary sums, yet it extends to any t-error correcting and detecting codes. Secondly, all the coding is done in parallel, i.e. each parity check bit is a function of the data bits only and not of any other check bit. This means that the hardware implementation of parallel coding is faster than that of serial coding.[1,2] Speed might not be a crucial matter in the communications area but it is of vital importance in computer applications.[3]

Thirdly, every code is represented in a Karnaugh map. This constitutes a further merit for the hardware engineer who can use the Karnaugh map as a direct implementation tool. Codes are traditionally represented via the parity-check matrix. The latter could be mapped on a Karnaugh map or vice versa. However, working with the Karnaugh map is easier since it is visual and one can use human intuition. The following section illustrates this point via an example of Hamming code represented in a Karnaugh map.

2. THE KARNAUGH MAP OF THE HAMMING CODE

Assume as an example a string of 8 bits ($x1, x2, \ldots x8$) which is to be transmitted or stored in a medium where errors could occur, i.e. xi could become 0 or vice versa. Before transmission or storage, one parity check bit p is appended to the word so that the value of p is the binary sum of $x1 + x2 + \cdots + x8$. Upon retrieval the same binary sum is performed and the value of the sum is compared to p. If the two values differ then an odd number of bits (including p) are wrong. In this example one is only 'detecting' whether an odd number of errors have occurred. If error correction is also desired then one has to add more parity check bits. The question is how many more bits should be added and what should their values be.

For the above string assume that 4 parity check bits are appended to form the codeword ($x1, x2, \ldots x8, p1, p2, p3, p4$) so that each pi is the binary sum of a subset of the xs. Thus there will be 4 sets $S1, S2, S3, S4$. Each set Si contains the parity bit pi and subset of the xs.

Let us deal first with the case where it is assumed that only 'one single error' could occur in the codeword. The question is which of the xs should belong to each set Si so that if one single error occurred, say, in one of the xs or the ps, then the code could point to the erroneous bit. This pointer in the literature is called the 'syndrome'. Each of the bits which could go wrong should be marked by a unique marker or pointer.

CCC-0010-4620/84/0027-0246$04.00

Before storage, the value of each pi is made equal to the sum (mod 2) of the xs in the set Si. Upon retrieval the same sums are performed again and compared to the retrieved $p1$, $p2$, $p3$, $p4$. If they differ in say $p1$ and $p4$ then the syndrome is 1001. Since it is assumed that one error only could occur, this syndrome tells us that the bit which belongs to sets $S1$ and $S4$ (and which does not belong to $S2$ and $S3$) went wrong. Now the erroneous bit must be found and corrected. On the other hand, if the syndrome was 0100 then the bit which belongs to $\overline{S1}$, $S2$, $\overline{S3}$, $\overline{S4}$ is erroneous. This is bit $p2$; therefore, the value of $p2$ must be negated.

To find unique syndromes for each bit, the Karnaugh map is used. The 4 sets Si could be represented as an 'empty' Karnaugh map. For example in Fig. 1(a), the set $S1$ consists of a total 8 squares, those in the second and 3rd rows. The set $S4$ contains all the squares in the 3rd and 4th columns, etc.

Since $p1$ is to belong to set $S1$ only, it will be placed in the corresponding square $S1$, $\overline{S2}$, $\overline{S3}$, $\overline{S4}$. Similarly, so will $p2$, $p3$ and $p4$. The null square which does not belong to any set is named N.

The data bits, the xs, are then placed in any fashion in the squares of the map so that each bit uniquely occupies one square, as in Fig. 1(a), which now represents a 'single error correcting' code. This code has $p1$, $x8$, $x5$, $x7$, $x1$ and $x3$ in set $S1$ and $p2$, $x6$, $x8$, $x5$, $x1$ and $x2$ in $S2$, and so on.

If $x5$ went wrong, then since $x5$ belongs to $S1$, $S2$, $\overline{S3}$, $S4$, upon decoding at the receiver end the syndrome formed will be 1101. This syndrome points to $x5$, therefore $x5$ has to be negated. If the syndrome is 0, 0, 0, 0 then no error has occurred. This is the null square N.

The code of Fig. 1(a) is represented in the literature by the following Hamming parity-check matrix:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Notice that each column of the matrix corresponds to a syndrome or a square.

Figure 1(a) shows that, with 4 parity bits, up to 11 data bits could be corrected. The three empty squares could be filled with $x9$, $x10$ and $x11$ as shown in Fig. 1(b). Thus the maximum number of data bits that a code with 4 parity check bits can have is 11; For n parity bits it would be $2^n - 1 - n$.

The code of Fig. 1 is a 'single-error' correcting code. For higher order error correction and detection more parity check bits will be needed. It is easily seen that if c is a codeword generated as explained above then $Hc = 0$.

The ideas from the above example for single-error correction could be extended to higher order error correction and detection. For a binary word, if certain occurrences of errors are possible, say, in only ($x1$ $x5$) and ($x2$ $x3$), then to be able to correct these errors, each of these possible error occurrences should have its own unique syndrome or its own unique square in the map. This is for error correction. For error detection more than one possible error combination could share the same square. Thus the problem is that of finding or allocating the right square for each error possibility so that these errors are detected and or corrected.

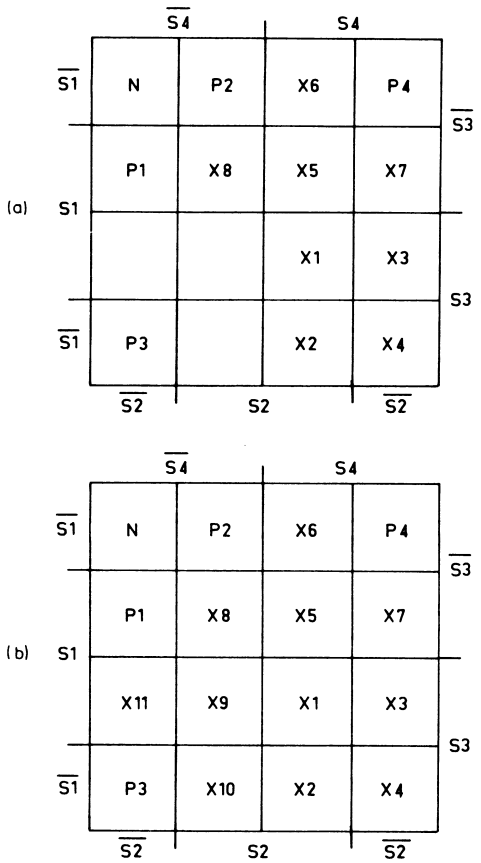

Figure 1. Karnaugh map of a Hamming code.

If two bits B_u and B_v are allocated to the squares whose addresses are u and v, respectively, then the square for the double error occurrence of B_u and B_v will have the address $(u + v)$. This will be formulated as Theorem 1 and it will be needed for the extension of earlier results on single error correction to results on higher order error detection and correction.

Theorem 1

Assume two bits each having its own single error syndrome. The syndrome of the double error occurrence of the two bits is the sum (mod 2) of their single syndromes.

Proof. Assume two bits $B1$ and $B2$. B could be x or p. $B1$ belongs to the sets $(Si_1, Si_2, \ldots, Si_N)$ and $B2$ belongs to the sets $(Sj_1, Sj_2, \ldots, Sj_M)$. If $B1$ is the only wrong bit then at the decoder (the receiving end) the calculated value for $(pi_1, pi_2, \ldots, pi_N)$ would be changed. Similarly, if only $B2$ went wrong, then $(pj_1, pj_2, \ldots, Pj_M)$ would be changed. If both $B1$ and $B2$ went wrong then $(pi_1, pi_2, \ldots, pi_n, pj_1, pj_2, \ldots, pj_M)$ would be changed if the pi and the pj subsets do not overlap. If these subsets overlap then for those ps occurring in both subset the 2 changes would cancel out. The result would be the (mod 2) sum of both subsets which is the (mod 2) sum of the single syndromes. Q.E.D.

Definition

The (Hamming) distance between any two words of equal length is defined as the number of bits by which they differ.

Definition

The (Hamming) weight of a binary word is the number of ls in the word.

Definition

If two sqaures differ by only the one set Si then one square is said to be the image of the other with respect to the Si axis.

Notice that in the single error correcting map of Fig. 1 the squares of any two bits have a minimum distance of one.

The following Sections 3 and 4 are independent, i.e. the reader could choose to read either first.

3. ERROR DETECTION

Hsiao[4] has introduced a class of optimal single-error correcting double-error detecting codes which provide faster and simpler hardware implementation. In what follows these codes will first be deduced via the Karnaugh map. These then will be generalized for t-error correction $(t+1)$-error detection. Conditions for fast hardware implementation are discussed in Section 4.

Consider any single error correcting code, e.g. that of Fig. 1. By adding another parity bit, $p5$, s.t. $p5 = x1 + \cdots + x11 + p1 + \cdots + p4$ the code becomes SEC-DED (single-error correcting and double-error detecting). In the Hamming matrix[5,6] another row of 1s is added and also another column $[00001]^T$. Now the set $S5$ contains all the xs and all the ps. Since the newly added $p5$ is a function of the other parity-check bits, this is serial coding, and the circuit has to wait for $p1, p2, p3, p4$ to be computed before it computes $p5$. It is desired that each parity bit, including $p5$, be a function of the data bits only so as to minimize encoding and decoding time. It is also desired that the maximum time taken to obtain the pis is minimal.

The Karnaugh map of the extended Hamming code with the five parity check bits is represented in Fig. 2. This is the image of Fig. 1(b) with respect to the $S5$ axis.

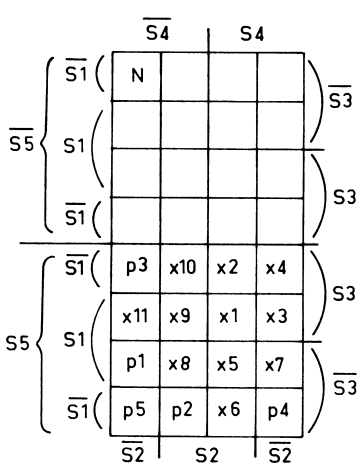

Figure 2. Single-error correction double-error detection by extending the Hamming code.

If the syndrome lies in $S5$, the single bit corresponding to that square is corrected; and if it lies in $\overline{S5}$ there is a double error somewhere, unless it lies in the N square, then no error is present.

If each of the parity bits is to be a function of the data bits only, then one could see that $p1, p2, p3$ and $p4$ should be placed back in their positions as in Fig. 1, that is, pi lies in the intersection of Si and $\overline{Sk}$ for all $k \neq i$.

A method obtained for correcting t-errors and detecting any $(t+1)$ errors from a t-error correcting code that will minimize circuit time delay is presented below. Here (by adding one extra parity check bit), any given t-error correcting code will become a t-error correcting and a $(t+1)$-error detecting code while keeping the property that each parity bit is a sum of data bits only. Before proving the general theorem some definitions and preliminary theorems are due.

Definition

A binary word will be defined as even if its Hamming weight (the sum of the non-zero bits) is even, and odd if the Hamming weight is odd.

All sums in this paper refer to exclusive-or operations or (mod 2) sums.

Theorem 2

The sum of any two odd words is an even word.

Proof. Suppose the two odd words are a and b, where a has the odd weight r and b has the odd weight s. b is the sum of canonical vectors, b_js$(j = 1, s)$, where each vector b_j has only one non-zero entry (and the sum of all the b_js is b). From this we get the following:

$$a + b = a + b_1 + b_2 + \cdots + b_s$$

$a + b_1$ is an even vector, since a has an odd number of 1s and b_1 has only 1. The even vector $a + b_1$ added to the odd vector b_2 results in an odd vector, etc. Since s is an odd number, the resulting vector $a + b$ is even. Q.E.D.

Corollary 1

The sum of an odd word and an even word is an odd word.

Corollary 2

The sum of any two even words is an even word.

The proofs for corollaries 1 and 2 are very similar to that of Theorem 2 and thus will be omitted.

In a Karnaugh map each square is represented by a binary vector. The square is defined as even if its binary word representation (addresses) is even, and odd otherwise. For example, in Fig. 1(b), the square where $x7$ lies is $[1, 0, 0, 1]$ corresponding to $\{S1, \overline{S2}, \overline{S3}, S4\}$; thus this square is an even square. In any Karnaugh map exactly half the squares are even and the other half are odd. Figure 3 shows the odd and even squares in a Karnaugh map of order 5, the shaded squares being the even ones.

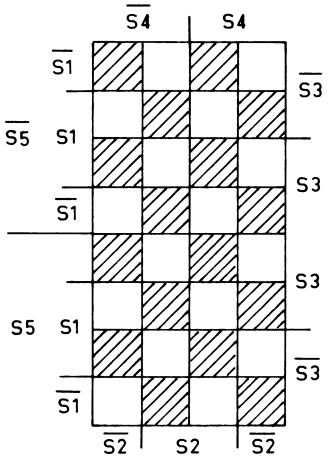

Figure 3. Shaded squares are the even squares.

Now, assuming that one starts with any single-error correcting code, e.g. the Hamming code shown in Fig. 1, and adds one more parity bit, the number of squares in the map becomes twice its original size. To make the above code double-error detecting as well, one proceeds as follows. Move bit positions which originally lay in an even square to their images around the $S5$ axis. That is, such bits will belong to $S5$ as well as to their original sets. Thus all bits now lie in odd squares. Since every bit still lies in a square not shared by any other bit, the code is still single-error correcting. If two errors occurred, then the syndrome would be even, since from Theorems 1 and 2 the sum of any two odd syndromes is even. Thus, if the syndrome corresponds to an even square, a double error has occurred.

The parity-check matrix of Fig. 4 is as follows:

$$\begin{bmatrix} 1&0&1&0&1&0&1&1&1&0&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&1&1&1&0&0&1&0&0&0 \\ 1&1&1&1&0&0&0&0&1&1&1&0&0&1&0&0 \\ 1&1&1&1&1&1&1&0&0&0&0&0&0&0&1&0 \\ 1&0&0&1&0&1&1&1&0&1&1&0&0&0&0&1 \end{bmatrix}$$

It will be shown in Section 5 that this code is optimal in terms of time; i.e. it results in minimum encoding and decoding time. Observe that the single-error correcting code has been extended to an SEC-DED code with no addition in delay execution time. That is, the delay time taken in encoding or decoding the new code (of Fig. 4) is exactly the same as that of the original single-error correcting code from which the new code has been obtained. The delay time is governed by the time taken to add (exclusive-or) the maximum number of bits in any set. In both cases (Figs 1 and 4) it is 8 bits. The maximum number of bits belonging to any set is the same as the maximum number of 1s in any row in the matrix.

S5S3S1 \ S4S2	00	01	11	10
000	N	P2		P4
001	P1		X5	
011		X9		X3
010	P3		X2	
110		X10		X4
111	X11		X1	
101		X8		X7
100	P5		X6	

Figure 4. Single-error correction double-error detection with independent parity bits.

For the general case, it could be shown (Theorem 5) that to extend a t-error correcting code to a t-error correcting and $(t+1)$-error detecting code with no additions in delay time the following sufficient condition should be satisfied: for the original t-error correcting code the total number of even data bits should be less than the maximum number of bits in any set of the code.

The above extension of the code to double error detection could be reversed. That is, given an (SEC-DED) code, constructed by adding another parity bit as explained above, then by eliminating the last parity bit, one gets back again a single-error correcting code. Actually, this case is of a more general nature, for it could be shown that by discarding any of the parity bits (not necessarily the last one), the resulting code becomes single-error correcting. This is generalized in Theorem 4. One can see this fact directly from Fig. 4. For example, assume that $S2$ is removed, then $p2$ is discarded: $x5$ will fall in the square between $p4$ and $x3$, which is the image of $x5$ with respect to the $S2$ axis, similarly, $x9$ will fall on its $S2$ image, which is the square between $p1$ and $p3$, and $x2$ will fall on its $S2$ image, which is between $x3$ and $x4$, etc. The resulting map will be single-error correcting since each square is occupied by only one position for each syndrome. In the corresponding parity-check matrix, this means discarding any one of the rows and the remaining zero column.

The reversal of the above extension of the single-error correcting to SEC-DED could be generalized to any t-error correcting codes.

The general theorems for error detecting are as follows.

Theorem 3

With any t-error correcting code, by adding one extra parity bit and moving the positions of all single errors originally occupying an even square in the Karnaugh map to their image squares around the last added axis, the code resulting is a t-error correcting $(t+1)$-error detecting code.

In matrix notation, this means adding another row to the parity-check matrix and another column. The entries of the row to be added would be 1 if the column above it is even and 0 otherwise. The column would have 1 in the $(t+1)$th row, and 0 otherwise.

Corollary

For a t-error correcting code, if all the data bits occupy odd squares, then the code is t-error correcting and $(t+1)$-error detecting as well.

Theorem 4

With any t-error correcting $(t+1)$-error detecting parallel code, whose single error syndromes are odd, by discarding

any of its parity bits, the resulting code is a t-error correcting one.

In matrix notation this means that in the parity-check matrix, if any row is discarded (plus the remaining zero column), then the resulting matrix is a parity check matrix for a t-error correcting code.

4. t-ERROR CORRECTION

In Section 2, it is assumed that only a single error could occur in a codeword. If single as well as double errors could occur and are to be corrected, then we need a double-error correcting code. If a code is to correct double errors as well as single errors then every data bit $x_1 \cdot i$, every parity-check bit pi, and every double-error $xixj$, $xkpl$ and $pmpn$ should have a unique square location in the map. Figure 5 is such a map. The code of Fig. 5 corrects 11 bits of which only 4 are data bits and 7 are parity-check bits.

Unfortunately, there are many empty squares in the map. If another data bit $x5$ is to be placed, for the code to be double-error correcting, eleven empty squares for $x5\,xi (i = 1, \ldots, 4)$ and $x5\,pj (j = 1, \ldots, 7)$ must be found. This is not possible. Much research could still be conducted in determining the maximum efficiencies of codes, and in finding such codes. The following is an algorithm for the construction of double-error codes. This algorithm is easily extended to construct higher order error correcting codes.

Starting from an empty map, first place all the check bits pis in their squares, remembering that for parallel encoding each pi lies in the corresponding Si only. Then place all the double occurrences of $pipj$, $\forall i, j$. The map is now as shown in Fig. 6. Manually, to place double occurrences e.g. $pipj$ (which is, from Theorrem 1, equal to $pi + pj$), one would check the sets that pi lies in and change pj by those sets. For example, in Fig. 6, $p1$ lies in $S1$ only, so to get $p1p2$ one would change $p2$ by $S1$ only (i.e. take the image of $p2$ around the $S1$ axis thus resulting in $p1p2$ as shown in Fig. 6).

Now the map is ready for placing data bits in squares. To do so, search for an empty square for the first data bit

S7S5S3S1 \ S6S4S2	000	001	011	010	110	111	101	100
0000	N	P2	P2P4	P4	P4P6	X4P7	P2P6	P6
0001	P1	P1P2	X1P3	P1P4	X2X3			P1P6
0011	P1P3	X1P4	(X1)	X1P2		X1P6		
0010	P3	P2P3	X1P1	P3P4				P3P6
0110	P3P5		X3X4					X3P7
0111			X1P5	X2P7			X2X4	
0101	P1P5							
0100	P5	P2P5		P4P5				P5P6
1100	P5P7	X1X2				X4P5		X3P3
1101				X2P3		X1X3		
1111	X2P4		X2P2	(X2)	X2P6			X3P1
1110	X3P6			X2P1	X3P4		X3P2	(X3)
1010	P3P7					X4P3		X3P5
1011			X1P7	X2P5				X1X4
1001	P1P7					X4P1		
1000	P7	P2P7	X4P6	P4P7	X4P2	(X4)	X4P4	P6P7

Figure 5. Karnaugh map for a double-error correcting code.

S7S5S3S1 \ S6S4S2	000	001	011	010	110	111	101	100
0000	N	P2	P2 P4	P4	P4P6	_	P2P6	P6
0001	P1	P1P2	_	P1P4	_		_	P1P6
0011	P1P3	_		_				_
0010	P3	P2P3	_	P3P4	_		_	P3P6
0110	P3P5	_		_				_
0111	_							
0101	P1P5	_		_				_
0100	P5	P2P5	_	P4P5	_		_	P5P6
1100	P5P7	_		_				_
1101	_							
1111								
1110	_							
1010	P3P7	_		_				_
1011	_							
1001	P1P7	_		_				_
1000	P7	P2P7	_	P4P7	_		_	P6P7

Figure 6. Starting map for any double-error correcting code. Map is ready for placing data bits.

$x1$ such that all its images with respect to all the set axes (i.e. all occurrences of $x1\,pi$, $\forall i$) are empty. Once such a square is found, assign $x1$ to that square and place all $x1\,pi$, $\forall i$ in their respective squares. For example, in Fig. 5, $x1$ and the images $x1\,p1$, $x1\,p2$, $x1\,p3$, $x1\,p4$, $x1\,p5$, $x1\,p6$, $x1\,p7$ are placed, the first four images happen to be surrounding $x1$ in this case. Now it is seen that the minimum distance of $x1$ from any other single (data or check) bits should be three sets on the map (corresponds to distance 5 in code words) or it is at least two sets from the double-error squares. Thus it is time saving to mark all squares whose distances from every placed (check or data) bit are less than three sets. These marked squares will never qualify to become a square for any other bit to be placed. The marking is more easily done from double-error squares since these marked to be squares are of distance one from the already existing double-error squares. These marks are shown as dashes in the lower right-hand corner of some squares in Fig. 6.

After placing any data bit, repeat the following: search for an eligible empty square for the next data bit xj such that the corresponding squares for xj and all the $xj\,xi$ for all previously assigned xis are empty. If dashing as explained above has not been done, then one should as well search for empty squares for $xj\,pi$, $\forall i$. If such a square is not found, stop. Otherwise place xj and all its corresponding double errors $xj\,pi$ and $xj\,xi$ in their respective squares and repeat.

The above algorithm finds a solution but not the best. That is, it does not place the maximum number of data bits in the map. However, the computer time used to reach and find such best known solutions (as in Ref. 7) for the double-error correction case was comparatively small. If one assumes that the code is $A(l, x)$ where l is the total number of bits and x is the number of data bits placed, then for $A(11, 4)$ the total time was 0.402 s, for $A(17, 9)$ it was 1.013 s and for $A(22, 13)$ it was 2.035 s. These were run on The University of British Columbia computer, the Amdahl 470 v6, Model II, using the time consuming, but convenient, WATFIV compiler.

Codes correcting burst-errors of specific length could also be obtained by modifying the basic algorithm. For example, for burst-errors of length 2, each data and parity-check bit and each consecutive error occurrence of all the $x(i-1)\,xi$, $\forall i$, all the $p(j-1)\,pj$, $\forall j$, and the last x with $p1$, should be allocated its own square. With an almost exhaustive computer search it was found that with six parity check bits, up to six data bits could be coded; and with seven parity check bits, up to eleven data bits were possible.

For higher order t-error correcting codes, the size of the problem and computations become high. However, there are methods to construct such codes where one would work with smaller maps; these will not be discussed in this paper.

5. MINIMUM TIME DELAY

Here it is assumed that data arrive in parallel in a data bus; all bits are accessed simultaneously and processed simultaneously so as to minimize time delays.

At the encoder, the value of each pi is obtained by exclusive-oring all the data bits lying in set Si. The values of the pis are obtained simultaneously (in parallel). Data bits are passed through inverted trees of exclusive-or gates simultaneously. The resulting outputs are the values of the pis. Assume that the parity check bit pj is the exclusive-or sum of the largest number of data bits, let this number be X. X could be written as $2^n + R$ if the exclusive-or gates have 2 inputs. For v-input exclusive-or gates $X = v^n + R$. Then the time delay is $n(=\log_v X)$ successive exclusive-or gates, if $R = 0$ and $n + 1$ if $R \neq 0$. Among codes of the same length it is desired to find those which result in minimum n. This will be discussed later.

The decoder is basically composed of two stages. The first stage obtains the syndrome S in the same fashion as the encoder, i.e. binary trees of exclusive-or gates. Exclusive-Nor could also be used, e.g. Motorola parity trees MC40008, MC4008.[8] The inputs to this stage include the parity bits as well as the data bits. Assume that the set Sj contains the largest number (X) of data bits, and if $X + 1 = v^n + R$, then the time delay is $n(=\log_v (X + 1))$, if $R = 0$; and is $n + 1$ if $R \neq 0$.

Once the syndrome is obtained it becomes the input to the second part of the decoder which eventually inverts the erroneous data bits. The implementation of such a stage could be done simply by programming the Karnaugh map code in a ROM or by having binary trees of AND gates in parallel. For the latter each binary tree corresponds to a syndrome Si and has as its first stage some corresponding inverting gates. Now, if the incoming signal is, say, 1001 then the only AND tree having inverting gates at its first and fourth inputs will show a 1 as output. The time delay of this part is governed by the number of parity check bits. If the code has p parity bits such that $p = v^m + R$, then the time delay is $m(=\log_v p)$ successive AND gates, if $R = 0$; and $m = 1$ if $R \neq 0$.

Using a ROM for this stage is more convenient but might be slower than using the parallel AND binary trees. The code, as it is in the Karnaugh map, is programmed in the ROM; thus a syndrome becomes the address of the memory location. So if the address points to the memory location whose content is say $xi\,xj$, then the corresponding wires to these data bits are made high and thus inverted.

For codes with a certain number of parity bits (and a certain minimum distance) one would like to find those codes which minimize the time delay of the first stage of the decoder (or encoder). The second stage is governed by the given number of parity bits. The rest of this section is devoted to the first stage.

In Section 3 the sufficient condition for a t-error correcting code to have the same delay execution time as a t-error correcting $(t + 1)$-error detecting code was stated. The delay time is determined by the number of binary bits to be exclusive-ored.

This will be stated as the following theorem.

Theorem 5

It is possible, by adding one extra parity bit, to extend a t-error correcting code to a t-error correcting $(t + 1)$-error detecting code, with no addition in the maximum number of bits to be summed (exclusive-ored), if the total number of even data bits in the t-error correcting code is less than the maximum number of bits in any of its sets.

As an example, consider the double error correcting code shown in Fig. 5 whose parity check matrix is

$$H = \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & \\ 1 & 0 & 0 & 1 & \\ 1 & 1 & 1 & 0 & \\ 1 & 1 & 0 & 1 & I \\ 0 & 1 & 1 & 0 & \\ 0 & 0 & 1 & 1 & \\ 0 & 1 & 1 & 1 & \end{array}\right]$$

There are three even data bits $x1$, $x3$ and $x4$ (corresponding to the first, third and fourth columns). This is less than the maximum number of bits in sets 3, 4 and 7 which is 4. By extending this code to triple error detecting, the eighth row to be added will be [1011, 00000001] which contains four 1s and thus the delay time will not be increased. If $v = 2$, this code will have a delay time of 2 (since there are three exclusive-ors for encoding and four for decoding).

But if one considers the following double error correcting code which has the same length as the one above

$$H_1 = \left[\begin{array}{cccc|c} 1 & 1 & 0 & 0 & \\ 1 & 0 & 0 & 1 & \\ 1 & 0 & 1 & 0 & \\ 1 & 1 & 0 & 1 & I \\ 0 & 1 & 1 & 0 & \\ 0 & 1 & 1 & 1 & \\ 0 & 0 & 1 & 1 & \end{array}\right]$$

and extends this code to a double-error correcting triple-error detecting code, the time lag will be increased from two to three successive exclusive-or gates (50% increase).

For t-error correcting codes a lower bound for the maximum number of bits to be exclusive-ored could be obtained. This is stated in the following theorem.

Theorem 6

For a t-error correcting code with n data bits and p parity check bits, the lower bound on the maximum number of bits to be decoded in obtaining any one bit of the syndrome is

$$= \frac{(n \times 2t)}{p} + 1, \quad \text{if } \frac{n \times 2t}{p} \text{ has no remainder}$$

$$= \frac{n \times 2t}{p} + 2, \quad \text{if } \frac{n \times 2t}{p} \text{ has a remainder}$$

Proof. For a t-error correction code the syndrome of any data bit should contain at least $2t$ non-zero elements. This is because the square for $xip1\,p2 \ldots p(t-1)$ should be distinct from the square for $ptp(t+1) \ldots p(2t-1)$ (e.g. for a double error correcting code the syndrome for the $xip1$ error should be different from the syndrome for the $p2p3$ error). And since there are n data bits altogether, the number of 1s in all the syndromes of the data bits must be at least $(n \times 2t)$. That is if the parity check matrix H is written as $H = [h1 : I]$, then $h1$ must contain at least $n \times 2t$ non-zero elements. The $h1$ submatrix is a $[p \times n]$ matrix. Therefore, at most, $(n \times 2t)/p$ columns could be completely filled by 1s if $(n \times 2t)$ is divisible by p; or $(n \times 2t)/p + 1$ otherwise.

Now to minimize the maximum number of 1s in any row of $h1$, the best that can be done is to fill $(n \times 2t)/p$ ($+1$ if remainder $\neq 0$) columns of $h1$ by 1s. Thus $(n \times 2t)/p + 1$ $(+1)$ is the minimal largest number of 1s any row of H could possibly have. Q.E.D.

Corollary

For a t-error correcting $(t+1)$-error detecting code with n data bits and p parity check bits, the lower bound on the maximum number of bits to be decoded in obtaining any bit of the syndrome is

$$= \frac{n(2t+1)}{p} + 1 \quad \text{if } n(2t+1) \text{ is divisible by } p$$

$$= \frac{n(2t+1)}{p} + 2 \quad \text{otherwise}$$

As an application example to the above corollary one could see that the code of the single-error correcting double-error detecting code of Fig. 4 (with 11 data bits and five parity check bits), would lead to a minimum delay time execution. This is because there are eight bits to be exclusive-ored in every one of the five binary trees,

S7S5S3S1 \ S6S4S2	000	001	011	010	110	111	101	100
0000	N	P2	P2P4	P4	P4P6		P2P6	P6
0001	P1	P1P2	X3P5	P1P4	X2P3			P1P6
0011	P1P3	X1P7		X2P6	(X2)	X2P2		X2P4
0010	P3	P2P3		P3P4	X2P1			P3P6
0110	P3P5						X2X3	
0111			X3P3		X2P5	X4P7		
0101	P1P5	X3P4	(X3)	X3P2		X3P6		
0100	P5	P2P5	X3P1	P4P5	X1X4			P5P6
1100	P5P7	X2X4						
1101			X3P7			X4P3		
1111		X1P5	X4P6		X4P2	(X4)	X4P4	
1110				X1X3		X4P1		
1010	P3P7	X1P1						X3X4
1011	X1P2	(X1)	X1P4		X2P7	X4P5	X1P6	
1001	P1P7	X1P3						
1000	P7	P2P7		P4P7		X1X2		P6P7

Figure 7

and this coincides with the lower bound which is 8 (the integer part of $[(11 \times 3)/5] + 2$).

As an application example of Theorem 6 it will be seen that for a double-error correcting code with $n = 4$ and $p = 7$ the lower bound on the delay time is the time taken to exclusive-or $[(4 \times 4)/7 + 2] = 4$ bits. The codes written above with the parity check matrices H and H_1 each have a delay time of 4 exclusive-ors at the decoder and as such are optimal codes. But the equivalent double-error correcting code of Fig. 7 has a delay time of five exclusive-ors and, as such is non-optimal. The parity check bit matrix of Fig. 7 is

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & \\ 0 & 0 & 1 & 1 & \\ 1 & 1 & 0 & 1 & \\ 0 & 1 & 1 & 1 & I \\ 0 & 0 & 1 & 1 & \\ 0 & 1 & 0 & 1 & \\ 1 & 0 & 0 & 1 & \end{bmatrix}$$

As mentioned in Section 4 the computer program generates many different possibilities of codes with a certain number of parity check bits. When choosing among the different largest codes of a given length and minimum distance, the computer chooses the ones with the minimum delay time.

6. CONCLUSION

The above represents an approach to error detection and error correction using the Karnaugh map. This approach is basically a geometrical extension of the Hamming code (most of the other existing methods are basically algebraic extensions).[9] The main advantage of this approach is its simplicity. Knowledge of advanced mathematics is not needed for constructing codes or for using them. The theory involved, though simple, extends to t-error correction and detection for all t.

Coding is done simply by placing data bits in a Karnaugh map of S variable sets, S being the number of parity-check bits. Then bits representing parity check bits and data bits are placed in the map so that they vary by a certain number of map sets depending on t, t being the order of error correction or detection. The codes obtained (or obtainable) are comparable to those best known.

For computer systems speed is a crucial factor. Therefore, encoding and decoding should be processed in parallel where possible. The codes produced with this approach are directly designed for parallel processing. Error correction and or detection have been introduced. Original results on detection were also found. It was also shown that burst codes for parallel processing could be constructed. (However, this approach is more suited to design codes dealing with certain specific bursts.) Conditions and lower bounds on time delays in the hardware implementation were found. These help find optimal codes which lead to minimum time delay implementation.

REFERENCES

1. J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applicators*, North-Holland, New York, Computer Design and Architecture Series, Chapter 9 (1978).
2. W. C. Carla, K. A. Duke and D. C. Jessep, Lookaside technique for minimum circuit memory translators. *IEEE Transactions on Computers* **C-22**(3), 283–289 (1973).
3. D. K. Pradham and J. J. Stiffer, Error-correcting codes and self-checking circuits. *IEEE, COMPUTER*, March (1980).
4. M. Y. Hsiao, A class of optimal minimum odd-weight-column SEC-DED codes. *IBM Journal of Research and Development* **14**(5), 395–401 (1970).
5. R. W. Hamming, Error detecting and error correcting codes. *Bell System Technical Journal* **29**, 147–160 (1950).
6. I. J. MacWilliams and N. J. A. Sloan, *The Theory of Error-Coding Codes*, Part I, North-Holland Publishing Co. (1977).
7. I. J. MacWilliams and N. J. A. Sloan, *The Theory of Error-Coding Codes*, Part II, p. 674, North-Holland Publishing Co. (1977).
8. *Error Detection and Correction Using Exclusive-OR gates and Parity Trees.* Prepared by John Linford, Motorola AN-496, Applications Engineering, Box 20917, Phoenix, Arizona, 85036, U.S.A.
9. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York (1968).

Received June 1983