1-1-1976

A survey of error-correcting codes for computer applications.

Dhriti Kapur

A SURVEY OF ERROR-CORRECTING CODES

FOR

COMPUTER APPLICATIONS

by

Dhriti Kapur

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Computer Science

Lehigh University
1976

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 5, 1976
(Date)

Professor in Charge

Chairman of the Department

ACKNOWLEDGEMENT

The author wishes to express his gratitude to his advisor, Prof. Kenneth Kai-Ming Tzeng, for his most valuable suggestions and advice during the preparation of this thesis.

Appreciation is also expressed to Lehigh University and Prof. A. K. Susskind, Head of the Department of Electrical Engineering, for providing financial support and encouragement to pursue graduate study at Lehigh University.

TABLE OF CONTENTS

ABSTRACT

The thesis presents an investigative survey of Error-Correcting Codes suitable for application in computer environment. Error-correcting codes have been successfully utilized to improve reliability in transmitting information in communication systems. In recent years the phenomenol increase in information handled by digital computers has enhanced the need for computer system reliability. In the survey with respect to error-correction the overall computer system has been broadly classified into three sections, namely the computer memory system, the computer peripheral system and the central processing unit. Each section is discussed under a separate heading.

Error-correcting codes used in computer memory systems depend upon the configuration of memory. For those memories which are packaged on single bit per card basis, single error-correcting, double error-detecting Hamming type codes, double error-correcting, triple error-detecting BCH codes, and one step majority decodable codes play a useful role in increasing the reliability of memory. Byte error-correcting codes form the basis of correcting errors in memories configured as multiple bit per card. A general class of maximal codes was developed by Hong and Patel whose structure is not restricted to any homogeneous bit per card arrangement and is capable of correcting single random byte errors.

Cyclic codes formed the basis of the error-correcting scheme

in magnetic tape and disc drives which are part of the computer peripheral system. Cyclic Redundancy Code (CRC) and the Orthogonal Rectangular Code (ORC) were found applicable to magnetic tape units. In magnetic disc systems Fire codes with high speed decoding could be used for single channel. Recently Malhotra and Fisher have come up with a practical error-correcting scheme for multichannel disc systems. Reed-Solomon codes were best suited for photodigital mass storage systems. The decoding scheme employed a hybrid hardware-software technique to simplify the complexity of decoding the multiple character correcting code.

In the processing unit of the computer the error-correcting codes used are arithmetic codes. The best known among these which are suitable for computer arithmetic as well as easily implementable are the residue codes. The Biresidue code proposed by Rao involved circuit redundancy of the magnitude of 30-35 percent of the main processor which is definitely more economical than duplicating redundant schemes like Triple Modular Redundancy.

A desirable feature of the error-correcting codes used in computer systems is the fast and simple encoding and decoding procedure. To ensure efficient operation the speed of implementation of the code must be comparable to the speed of operation of the computer system. Most of the codes surveyed in this thesis have been found suitable in computer environment with existing trade-off between redundancy and decoding time.

CHAPTER 1. INTRODUCTION

Error-correcting codes are mathematical codes designed to encode information in such a manner that it can be decoded correctly at the receiving end in the presence of disturbance in the transmitting medium. Typical examples of transmission mediums are telephone lines, high frequency radio links, space communication links and magnetic tape units including writing and reading heads for storage systems. The codes are constructed in binary digits compatible with information handled in digital computers. Error-correcting codes have been used successfully in communication systems to improve reliability. The idea has been carried over to computer environment where the need for reliability in storing and processing information has assumed wide importance with increasing demand for more computer power.

The idea of increasing system reliability through hardware was first initiated by Von Neumann. In his scheme the original network was duplicated an odd number of times. If three identical copies of an original circuit are made and the output of each circuit is taken to a majority voter assumed to be perfect, then this scheme can tolerate error in one of the circuits. The introduction of extra copies required to improve reliability is called redundancy. In particular, the above scheme is known as Triple Modular Redundancy. This is equivalent to triplicating a binary digit (also called encoding) before transmission in a binary symmetric channel

and ensuring safety of the information in the presence of single error due to disturbance in the transmitting medium or transmitting the same digit three times over certain intervals of time. The fact that some form of redundancy either in hardware, binary digits or time was needed is fundamental in the concept of system reliability.

In the field of error-correcting code the rules for encoding and decoding information can be specified a parity check matrix. For instance, if we desired to transmit 4 information binary digits $x_1x_2x_3x_4$ we would transmit instead a binary code sequence $y_1y_2y_3y_4y_5y_-$ $y_5y_6y_7$. This introduction of redundancy to the information digits is known as encoding. Only 2^4 out of the 2^7 possible encoded sequences would be acceptable. If two encoded output sequences differ in at least 3 positions, a single error will always be corrected. This is achieved by the following encoding rules.

$$y_i = x_i \quad i = 1,2,3,4$$
$$y_5 = x_2 \oplus x_3 \oplus x_4$$
$$y_6 = x_1 \oplus x_3 \oplus x_4$$
$$y_7 = x_1 \oplus x_2 \oplus x_4$$

where $\oplus$ indicates mod 2 addition. The equations can be written in terms of y_i

$$0y_1 \oplus 1y_2 \oplus 1y_3 + 1y_4 \oplus 1y_5 \oplus 06_6 \oplus 0y_7 = 0$$
$$1y_1 \oplus 0y_2 \oplus 1y_3 \oplus 1y_4 \oplus 0y_5 \oplus 1y_6 \oplus 0y_7 = 0$$
$$1y_1 \oplus 1y_2 \oplus 0y_3 \oplus 1y_4 \oplus 0y_5 \oplus 0y_6 \oplus 1y_7 = 0$$

which can be compactly written as

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

It is easy to verify that each of the 16 possible encoded sequences is a solution to the above equations. The matrix of 0 and 1 is termed the parity check matrix and is usually denoted by the symbol H. It is the choice of this matrix which is important in designing Algebraic codes.

For decoding the output sequence is taken and the syndrome $s_1 s_2 s_3$ is calculated

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

If the output sequence differs from a codeword in only one position then the vector $[s_1 s_2 s_3]^T \neq 0$ and will be equal to one of the columns of the parity check matrix. Since all columns of the parity check matrix are distinct we will be able to identify the position of the error by observing the vector $[s_1 s_2 s_3]^T$. If an output

sequence 1001011 is received the syndrome vector is calculated as $[111]^T$ which indicates that the error was in the 4th position and thus we should decode to the codeword 1000011.

In general a code is represented as (n,k), where k is the number of information digits and n is the length of the code sequence. In the above example we used a (7,4) code. For a more detailed exposure to Coding Theory, the reader is referred to Peterson [1], Shu Lin [2].

Since a digital computer operates at ultra high speeds, the encoding and decoding process built in the computer system must be fast and simple. The code constructed should have minimum redundancy, maximum efficiency, and low cost to be a viable alternative in computer system reliability.

The survey deals with a large number of such codes which have been developed to meet specifically the requirements of reliability for computer memory systems, peripheral systems and Arithmetic processors. Each system is discussed under a separate heading.

CHAPTER 2. ERROR-CORRECTING CODES FOR COMPUTER MEMORY SYSTEMS

Capacity, access time, data transfer rate and cost/bit are basic performance considerations of mass storage devices. The demand for more storage capacity with high speed access appears as insatiable as the demand for more computer power. Improvement in capacity and access time cannot be at the expense of reliability because most users are placing their vital records under computer control.

In recent years basic technology has shifted from core memory towards monolithic memory and LSI technology decreasing random access cycle time from μs to ns range. However, batch fabricated or integrated memory elements are not as reliable as cores giving rise to permanent stuck type memory faults.

2.1 Error Correction by Parity Checking

By successive read/write operation according to flow chart (fig. 2.1) odd number of errors are corrected by simple parity code. In most events economic considerations will dominate any decision to use the parity code versus reworking of components to obtain a perfect memory. On the other hand, temporary faults caused by the malfunction of the driver, sense amplifier circuits or random electromagnetic noise interference, where the errors are not reproducible, successive read operation with parity check can correct odd errors (fig 2.2). Even number of errors still remain undetected. It is here that random-error-correcting codes play an

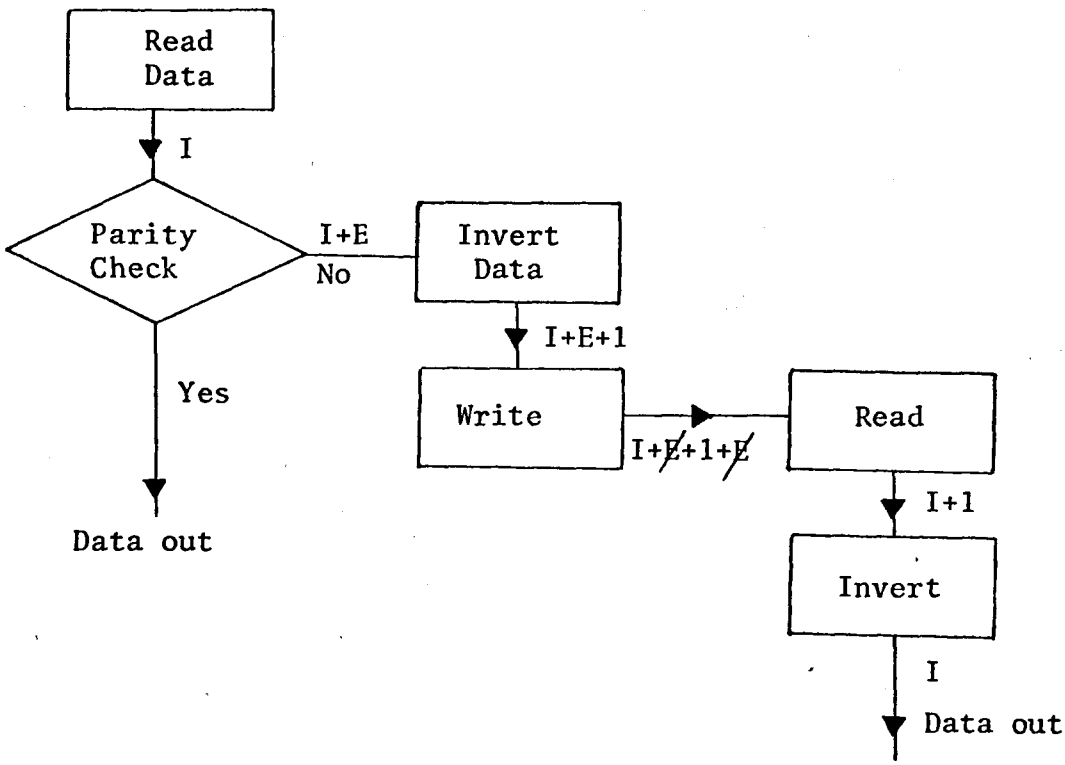

Read
Data
I
Parity
Check
I+E
No
Invert
Data
I+E+1
Write
I+E+1+E
Read
I+1
Invert
I
Data out
Yes
Data out

Fig. 2.1

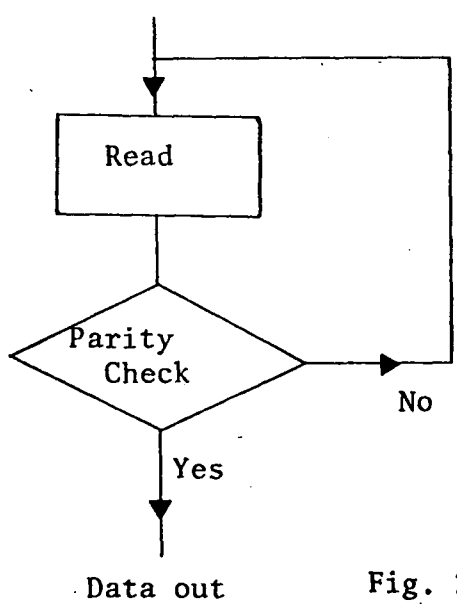

Read
Parity
Check
No
Yes
Data out

Fig. 2.2

important part in system reliability as compared with other techniques.

2.2 Single-error-correcting, Double-error-detecting Code for Single Bit Per Card Memory Configuration

The most widely used error-correcting codes in the computer memory environment are the single-error-correcting, double-error-detecting (SEC-DED) Hamming code [7] and double-error-correcting, triple-error-detecting BCH codes (DED-TED) [8]. These codes are suitable for single bit per card configuration proposed by Allen [5]. The memory is organized such that every bit of a code word appears in a single card. Errors in the card will therefore appear as single errors in the codeword.

SEC-DED Hamming codes have been used in IBM 7030 and IBM System/360 model 85, having 72 bits per word, 64 data bits and 8 check bits. These codes are slightly modified from the conventional single-error-correcting, double-error-detecting codes of distance 4 used in communication channels. The row of all 1's in the parity check matrix of the conventional code is not used. Instead the parity check matrix is constructed such that every column is of odd weight and distinct. Since any linear combination of three odd weight distinct columns of the parity check matrix is independent, i.e. they do not add to all zero, the minimum distance of the code becomes ≥ 4, satisfying the condition for simultaneous detection of two errors and correction of single errors.

The syndrome

$$S = eH^T \qquad \text{(Eq.2.10)}$$

is implemented using an Ex-OR network [7]. The number of inputs to each gate and the number of logic levels required are dependent on the number of 1's in each row of the parity check matrix. The code is therefore constructed such that the number of 1's in the parity check matrix is a minimum. It is for this reason that the overall parity check in the conventional Hamming code is avoided in the construction of SEC-DED via odd-weight-column parity check matrix, thus improving the speed of encoding and decoding. If r parity check bits are used to match the k data bits, the parity check matrix is constructed using the formula

$$\sum_{\substack{i=1 \\ \text{odd}}}^{\leq r} \binom{r}{i} \geq k + r \qquad \text{(Eq.2.11)}$$

where each term stands for the possible combinations of r columns of wt i. This code has minimum redundancy as the unshortened code requires the same number of check bits as the conventional Hamming code.

$$\sum_{\substack{i=1 \\ \text{odd}}}^{\leq r} \binom{r}{i} = \frac{1}{2} 2^r = 2^{r-1} \qquad \text{(Eq.2.12)}$$

The syndrome corresponding to single error matches the particular column of the parity check matrix identifying its location. An

Byte	1	2	3	4	5	6	7	8	Check
Bit	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39	40 41 42 43 44 45 46 47	48 49 50 51 52 53 54 55	56 57 58 59 60 61 62 63	64 65 66 67 68 69 70 71
S_1	11111111	00100110	01001001	10010000	00010000	00010011	00011100	11100000	10000000
S_2	11100000	11111111	00100110	01001001	10010000	00010000	00010011	00011100	01000000
S_3	00011100	11100000	11111111	00100110	01001001	10010000	00010000	00010011	00100000
S_4	00010011	00011100	11100000	11111111	00100110	01001001	10010000	00010000	00010000
S_5	00010000	00010011	00011100	11100000	11111111	00100110	01001001	10010000	00001000
S_6	10010000	00010000	00010011	00011100	11100000	11111111	00100110	01001001	00000100
S_7	01001001	10010000	00010000	00010011	00011100	11100000	11111111	00100110	00000010
S_8	00100110	01001001	10010000	00010000	00010011	00011100	11100000	11111111	00000001

Fig. 2.3

Byte	1	2	3	4	5	6	7	8	Check
Bit	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39	40 41 42 43 44 45 46 47	48 49 50 51 52 53 54 55	56 57 58 59 60 61 62 63	64 65 66 67 68 69 70 71
S_1	11111111	00001111	00001111	00001100	01101000	10001000	10001000	10000000	10000000
S_2	11100000	11111111	00000000	11110011	01100100	01000100	01000100	01000000	01000000
S_3	00011100	11110000	11111111	00001111	00000010	00100010	00100010	00100110	00100000
S_4	00010011	00000000	11110000	11111111	00000001	00010001	00010001	00010110	00010000
S_5	00010000	10001000	10001000	10000000	11111111	00001111	00000000	11110011	00001000
S_6	10010000	01000100	01000100	01000000	11110000	11111111	00001111	00001100	00000100
S_7	01001001	00100010	00100010	00100110	11001111	00000000	11111111	00001111	00000010
S_8	00100110	00010001	00010001	00010110	00110000	11110000	11110000	11111111	00000001

Fig. 2.4

even number of 1's in the syndrome would suggest the existence of multiple even error including detection of double errors. Multiple odd error would be correctely detected if the syndrome pattern lies outside the codes parity check matrix. Otherwise a miscorrection would result. Since the probability of having one bit in error is low (of the order of 10^{-5}) the probability of having a large number of multiple error is even smaller.

Illustrative Example

The (72,64) code used in IBM 7030 and IBM 360 system Model 85 is the shortened version of (128,120) full length code. The 72 columns of the parity check matrix are constructed using Eq. 2.11.

$$72 = \binom{8}{1} + \binom{8}{3} + 8/\binom{8}{5} \qquad \text{(Eq. 2.13)}$$

The stroke in Eq. 2.13 suggests that 8 columns of wt 5 of the possible $\binom{8}{5}$ combinations are used. The total number of 1's in the H matrix is equal to 8 + 3 x 56 + 8 x 5 = 216. The average number of 1's in each row is equal to 216/8 = 27. This implies that if a 3-way ExOR gate is used, the number of logic levels required to generate S_i is equal to 3. Two versions of the parity check matrix of the (72,64) SEC-DED codes are shown in Fig. 2.3 [7] and Fig. 2.4 [7]. A simple layout of the encoder and decoder is shown in Fig. 2.5.

The memory is assumed to be a binary symmetric channel and errors are statistically independent. The logic circuit is simple,

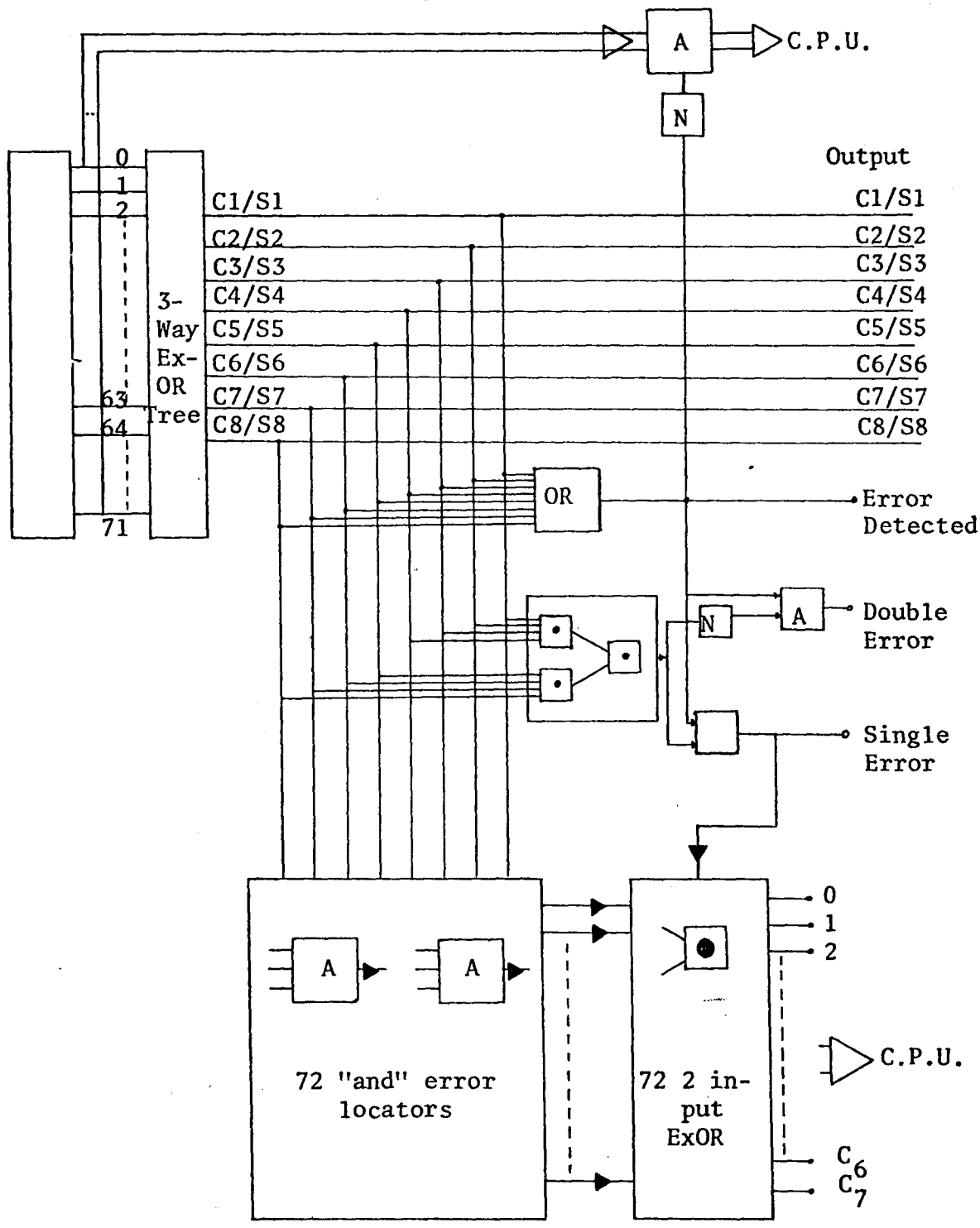

A
C.P.U.
N
Output
0
1
2
63
64
71
3-
Way
Ex-
OR
Tree
C1/S1
C2/S2
C3/S3
C4/S4
C5/S5
C6/S6
C7/S7
C8/S8
C1/S1
C2/S2
C3/S3
C4/S4
C5/S5
C6/S6
C7/S7
C8/S8
OR
Error
Detected
N
A
Double
Error
Single
Error
A
A
72 "and" error
locators
72 2 in-
put
ExOR
0
1
2
C.P.U.
C6
C7

Fig. 2.5

fast and the code has minimum redundancy suitable for computer application.

2.3 Double-error-correcting, Triple-error-detecting Code for Single Bit Per Card Memory Configuration

With a slight increase in the decoding complexity double error correction can be done, improving the reliability of the memory.

The code used is a DEC-TED BCH code. The generator polynomial is of the form $g(x) = (1+x)g_1(x)$

$$g_1(x) = \text{LCM}(m_1(x), m_3(x)) \qquad \text{(Eq. 2.14)}$$

where $m_i(x)$ are the minimum polynomial of α^i, α being a primitive element of the Galois field $GF(2^m)$. The degree of $g_1(x)$ is at most 2m. The minimum distance $d \geq 5$. The (1+x) factor in g(x) increases the distance of the code by one. This ensures that every column of the parity check matrix generated by X^i mod g(x) is of odd weight. Odd weight syndromes then correspond to odd number of errors and an even weight syndrome tells that the error polynomial is of even weight. Also every code word is of even weight.

Example

Using the (79,64) DEC-TED BCH code for illustration, the code has 64 data bits and 15 check bits and is a shortened cyclic code of the full length code (127,112). The polynomial $g_1(x)$ has root in $GF(2^7)$

$$m_1(x) = 1 + x^3 + x^7$$

The roots of $m_1(x)$ are α, α^2, α^4, α^8, α^{16}, α^{32}, α^{64}

$$m_3(x) = 1 + x + x^2 + x^3 + x^7$$

The roots of $m_3(x)$ are α^3, α^6, α^{12}, α^{24}, α^{48}, α^{96}, α^{65}

$$g_1(x) = (1 + x^3 + x^7)(1 + x + x^2 + x^3 + x^7)$$
$$= (1 + x + x^2 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{14}$$

Since $g(x) = (1+x)g_1(x)$

$$= (1+x)(1+x+x^2+x^4+x^5+x^6+x^8+x^9+x^{14})$$
$$= 1 + x^3 + x^4 + x^7 + x^8 + x^{10} + x^{14} + x^{15}$$

Each row of H^T can be generated by a feedback register characterized by g(x) (Fig. 2.6) with initial state 100000000000000. The other states are as follows:

00	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	1	1	0	0	1	1	0	0	1	1	0	1	0	0	0
01	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	1	1	0	0	1	1	0	0	1	1	0	1	0	0
02	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0
03	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1
04	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	21	1	0	0	1	0	1	0	1	0	1	1	0	1	1	1
05	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	22	1	1	0	1	0	0	1	1	0	0	0	1	0	1	0
06	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	23	0	1	1	0	1	0	0	1	1	0	0	0	1	0	1
07	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	24	1	0	1	0	1	1	0	1	0	1	1	0	0	1	1
08	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	25	1	1	0	0	1	1	1	1	0	0	0	1	0	0	0
09	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	26	0	1	1	0	0	1	1	1	1	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	27	0	0	1	1	0	0	1	1	1	1	0	0	0	1	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	28	0	0	0	1	1	0	0	1	1	1	1	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	29	1	0	0	1	0	1	0	1	0	1	0	1	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	30	1	1	0	1	0	0	1	1	0	0	0	0	1	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3111	1	1	1	0	0	0	0	0	0	1	0	0	1	1	
15	1	0	0	1	1	0	0	1	1	0	1	0	0	0	1	32	1	1	1	0	0	0	0	1	1	0	1	1	0	0	0
16	1	1	0	1	0	1	0	1	0	1	1	1	0	0	1	33	0	1	1	1	0	0	0	0	1	1	0	1	1	0	0

34	0 0 1 1 1 0 0 0 0 1 1 0 1 1 0
35	0 0 0 1 1 1 0 0 0 0 1 1 0 1 1
36	1 0 0 1 0 1 1 1 1 0 1 1 1 0 0
37	0 1 0 0 1 0 1 1 1 1 0 1 1 1 0
38	0 0 1 0 0 1 0 1 1 1 1 0 1 1 1
39	1 0 0 0 1 0 1 1 0 1 0 1 0 1 0
40	0 1 0 0 0 1 0 1 1 0 1 0 1 0 1
41	1 0 1 1 1 0 1 1 0 1 1 1 0 1 1
42	1 1 0 0 0 1 0 0 0 0 0 1 1 0 0
43	0 1 1 0 0 0 1 0 0 0 0 0 1 1 0
44	0 0 1 1 0 0 0 1 0 0 0 0 0 1 1
45	1 0 0 0 0 0 0 1 0 0 1 0 0 0 0
46	0 1 0 0 0 0 0 0 1 0 0 1 0 0 0
47	0 0 1 0 0 0 0 0 0 1 0 0 1 0 0
48	0 0 0 1 0 0 0 0 0 0 1 0 0 1 0
49	0 0 0 0 1 0 0 0 0 0 0 1 0 0 1
50	1 0 0 1 1 1 0 1 1 0 1 0 1 0 1
51	1 1 0 1 0 1 1 1 0 1 1 1 0 1 1
52	1 1 1 1 0 0 1 0 0 0 0 1 1 0 0
53	0 1 1 1 1 0 0 1 0 0 0 0 1 1 0
54	0 0 1 1 1 1 0 0 1 0 0 0 0 1 1
55	1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
56	0 1 0 0 0 0 1 1 1 1 1 1 0 0 0
57	0 0 1 0 0 0 0 1 1 1 1 1 1 0 0
58	0 0 0 1 0 0 0 0 1 1 1 1 1 1 0
59	0 0 0 0 1 0 0 0 0 1 1 1 1 1 1
60	1 0 0 1 1 1 0 1 1 0 0 1 1 1 0
61	0 1 0 0 1 1 1 0 1 1 0 0 1 1 1
62	1 0 1 1 1 1 1 0 1 1 0 0 0 1 0
63	0 1 0 1 1 1 1 1 0 1 1 0 0 0 1
64	1 0 1 1 0 1 1 0 0 0 0 1 0 0 1
65	1 1 0 0 0 0 1 0 1 0 1 0 1 0 1
66	1 1 1 1 1 0 0 0 1 1 1 1 0 1 1
67	1 1 1 0 0 1 0 1 1 1 0 1 1 0 0
68	0 1 1 1 0 0 1 0 1 1 1 0 1 1 0
69	0 0 1 1 1 0 0 1 0 1 1 1 0 1 1
70	1 0 0 0 0 1 0 1 0 0 0 1 1 0 0
71	0 1 0 0 0 0 1 0 1 0 0 0 1 1 0
72	0 0 1 0 0 0 0 1 0 1 0 0 0 1 1
73	1 0 0 0 1 0 0 1 0 0 0 0 0 0 0
74	0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
75	0 0 1 0 0 0 1 0 0 1 0 0 0 0 0
76	0 0 0 1 0 0 0 1 0 0 1 0 0 0 0
77	0 0 0 0 1 0 0 0 1 0 0 1 0 0 0
78	0 0 0 0 0 1 0 0 0 1 0 0 1 0 0

The syndrome bits are computed by Ex-OR tree. If S = 0 the data bits are sent to the CPU. When parity of S is odd it implies single or multiple odd errors. Single errors are corrected by the logic circuit similar to Fig. 2.5. Double error correction is done when parity of syndrome bits is even. The syndrome is loaded into the serial linear feedback register (Fig. 2.6) and the codeword in an n-bit register. Both registers are shifted simultaneously till the first error bit has shifted into the rightmost bit position. This is detected when the syndrome pattern matches any of possible 78 double error patterns stored in a ROM.

78 + 77 = 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1
78 + 76 = 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1
⋮
78 + 00 = 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1

The leading bit in the n-bit register is inverted. The remaining error can be corrected by the single error correcting scheme.

2.4 Error-correcting Codes for Multiple Bit Per Card Memory Configuration

The error-correcting codes we have seen so far are suitable for single bit per card memory configuration. Increasing speed and system efficiency demands have pushed the idea of single bit per card to a cluster of bits per card type memory organization. A single fault in these systems would affect many bits within a byte. Consequently a byte-error-correcting ability is demanded of the codes. There are a number of known classes of error-correcting

$$g(x) = 1+x^3+x^4+x^7+x^8+x^{10}+x^{14}+x^{15}$$

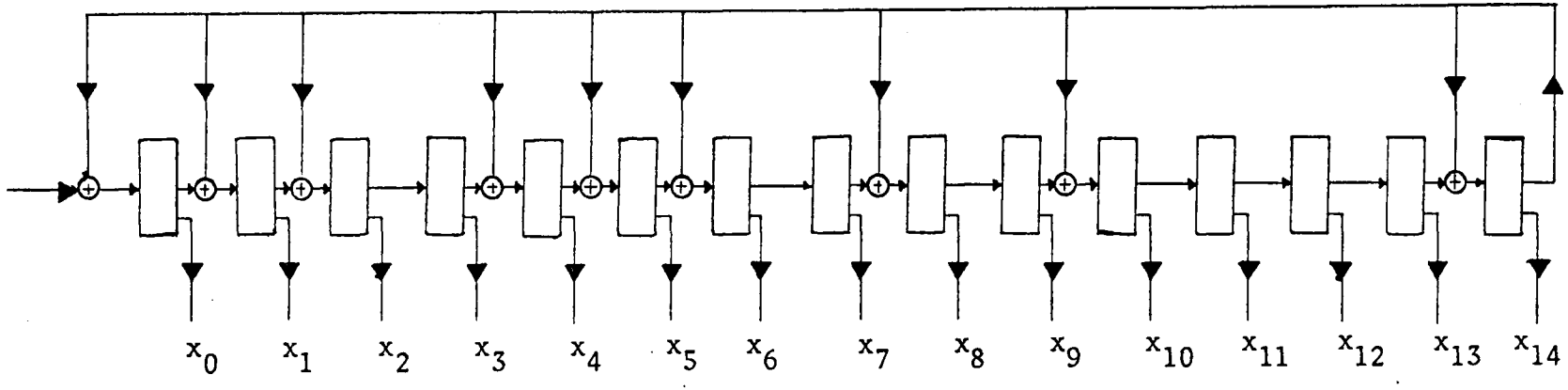


Fig. 2.6

codes that have this property. Best among these, suitable for high speed implementation, are the redundant residue polynomial code [9].

It is well known that the parity check matrix of the single-error-correcting Hamming code can be constructed with each element being a symbol from a finite field $GF(2^b)$. Then any single error corresponds to a block of b bits in error.

As a simple example, if we consider $GF(2)[x]$ mod. $x^2 + x + 1$, the residue classes are $\{0\},\{1\},\{x\},\{x+1\}$. They can be represented by powers of symbol α where $\alpha^2 + \alpha + 1 = 0$. In vector form each symbol appears as a binary 2 tuple

Symbol	Vector Form
0	0 0
1	0 1
α	1 0
α^2	1 1

The companion matrix T of the irreducible polynomial p(x) of degree b is a bxb nonsingular matrix described by

$$T = \begin{bmatrix} | & | & | & & | \\ \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^b \\ | & | & | & & | \end{bmatrix}$$

where α is the root of the irreducible polynomial in $GF(2^b)$. Other symbols in $GF(2^b)$ can be represented by the matrix

$$T = \begin{bmatrix} | & | & & | \\ \alpha^{i+1} & \alpha^{i+2} & \cdots & \alpha^{i+b} \\ | & | & & | \end{bmatrix}$$

The zero and one symbols in $GF(2^b)$ are the bxb zero matrix T_o and the bxb identity matrix T_1 respectively. In our example

$$T_o = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \; ; \; T_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; ; \; T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \; ; \; T' = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

The H matrix for single card correction code with 2-bit per card memory consisting of 64 data bits and 8 check bits is shown (Fig. 2.7). Each element is a bxb binary matrix.

An interesting subclass of these Hamming type codes is 2-redundant codes [9] having two check symbols and capable of single symbol correction in $GF(2^b)$. They always have the parity check matrix of the form

$$H = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ \alpha^0 & \alpha & \alpha^2 & \cdots & \alpha^{2^b-2} & 0 & 1 \end{bmatrix}$$

If α is the root of the primitive polynomial in $GF(2^b)$ then all columns are distinct and the maximum number of information symbol is 2^b-1.

Example

An (80,64) code over $GF(2^8)$ packaged as 8 bit per card memory with 64 data bits and two check cards (16 bits) has the following parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & 0 & 1 \end{bmatrix}$$

Each symbol is replaced by its corresponding T matrix to obtain the binary H matrix for purposes of implementation.

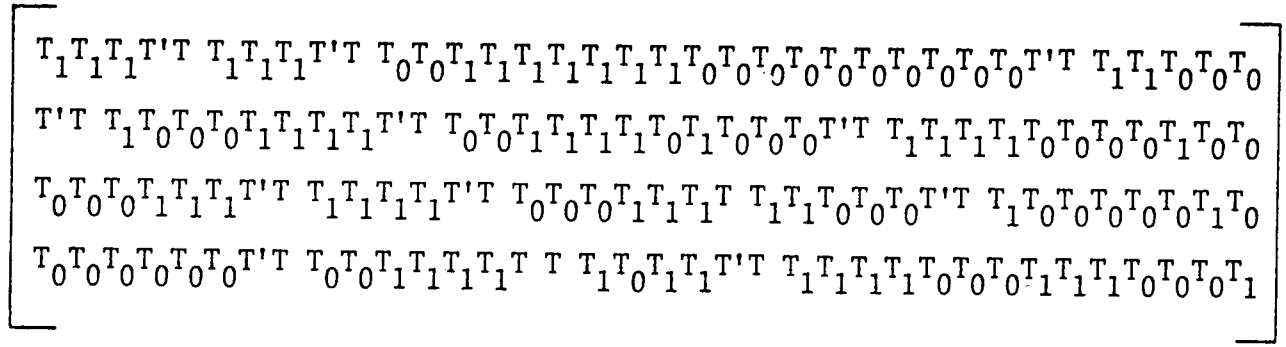


Fig. 2.7

The syndrome for H matrix is computed by the usual Ex-OR tree. Since the error pattern now corresponds to some symbol $e_i \in GF(2^b)$ in block i of data bits the syndrome corresponding to this error has the value

$$\begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} e_i \\ e_i \alpha^i \end{bmatrix}$$

where S_1 and S_2 are binary column vectors of length b. If the error occurs in the check symbol, the syndrome is either

$$\begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} e_i \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} 0 \\ e_i \end{bmatrix}$$

This case can be easily detected by And gates checking either S_1 = all zeros or S_2 = all zeros.

When error is in the information symbols, $S_1 \neq 0$, $S_2 \neq 0$ and $S_2 = \alpha^i S_1$ or $S_2 + \alpha^i S_1 = 0$ where + stands for bit by bit Ex-OR. To test the above condition a set of Ex-OR circuits can be set up for all possible values of α^i, i = 1,2,...2^b-1. This gives the location of the block in error. The error magnitude $e_i = S_1$ can be added mod 2 to block i for error correction.

For K information symbols, the decoding circuit requires on an average K+2 And gates of b inputs and Kb Ex-OR gates with an average of b/2+1 inputs each.

The 2-redundant codes thus provide a simple means for correcting single bit error or b-adjacent bit error by treating it as a symbol in $GF(2^b)$. The check bit are integral multiples of b.

The redundant bits r = kb where k is an integer.

2.5 General Class of Maximal Codes

Hong and Patel [10] proposed a new class of codes whose structure does not depend on $GF(2^b)$. The byte is equated to a suitable cluster of bits and the check bits r = kb+c where $0 \leq C < b$. These codes are either perfect or maximal. Hence they are called the general class of maximal codes (GMC).

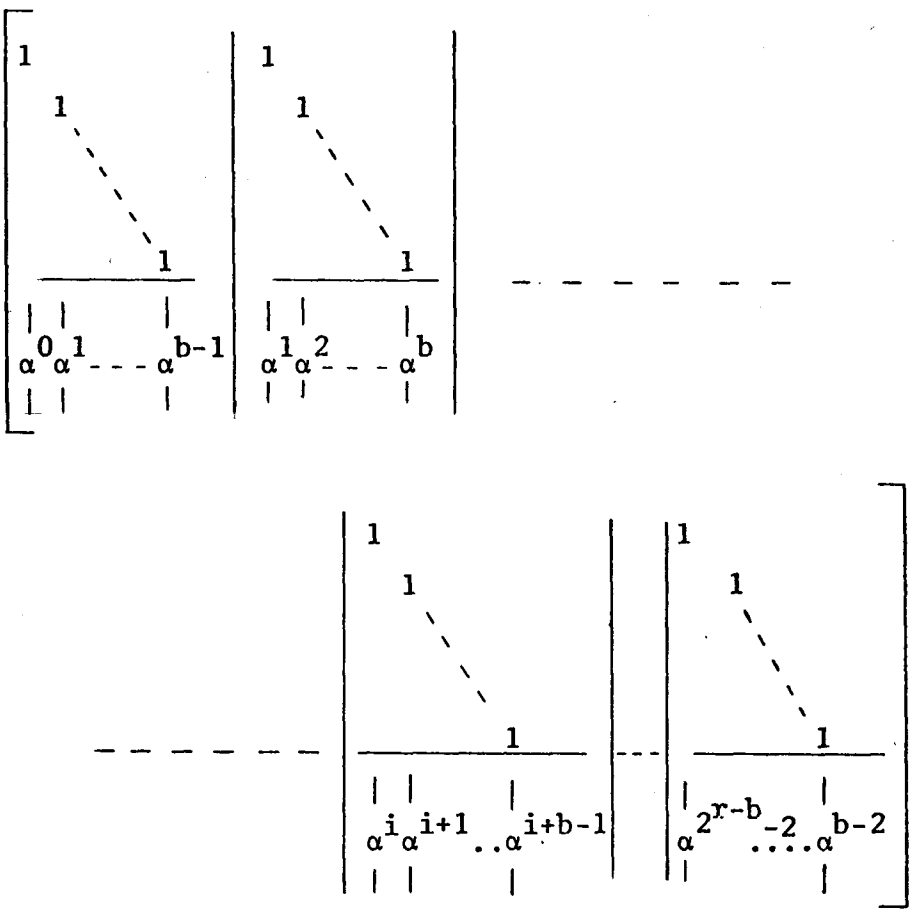
In general, the identity portion of H matrix looks like

$$I_r = \begin{bmatrix} I_b & & & \\ & I_b & 0 & \\ & & \ddots & \\ 0 & & & I_b \\ & & & & I_{b+c} \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} (k-1)b\ I_b \text{ matrices} \\ \left.\begin{matrix} \end{matrix}\right\} 1 \quad I_{b+c} \text{ matrix} \end{matrix} \qquad \text{(Eq.2.15)}$$

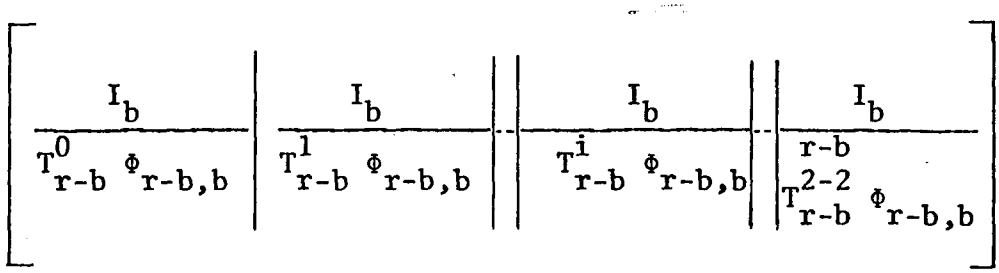
$$\text{or} \quad I_r = \begin{bmatrix} I_b & & & \\ & I_b & 0 & \\ & & I_b & \\ & 0 & & \ddots \\ & & & & I_b \\ & & & & & I_c \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} kb\ I_b \text{ matrices} \\ \left.\begin{matrix} \end{matrix}\right\} 1 \quad I_c \text{ matrix} \end{matrix} \qquad \text{(Eq.2.16)}$$

For the discussion on the code, the form appearing in Eq. 2.15 will be used. The non-identity portion of the parity check matrix

can be formed by successive iteration of the matrix $H_{r,b}$ constructed as follows:



where α is a primitive element from $GF(2^{r-b})$. This can be conveniently represented as



where T_{r-b} is the companion matrix of the primitive polynomial in $GF(2^{r-b})$ and $\Phi_{r-b,b}$ is a r-bxb matrix of the form

$\left[\begin{array}{c} I_b \\ \hline 0_{r-2b,b} \end{array}\right]$ which chops off the rightmost 4-2b columns of the T matrix. I_b is a bxb identity matrix and $0_{r-2b,b}$ is a r-2bxb zero matrix.

For $r \geq 2b$ the parity check matrix formed as

$$H = [H_{r,b} | I_r] \qquad \text{(Eq.2.17)}$$

is capable of correcting single byte errors.

The single byte correcting ability of this code can be proved by showing that a distinct syndrome exists for every single error pattern.

$$S = vH^T = [s_1\ s_2\ \ldots\ s_k]^T$$

S_i represents the syndrome byte corresponding to check byte C_i. The codeword consisting of information and check bytes is $v = B_0\ B_1\ \ldots\ B_{2^{r-b}-2}\ C_1\ C_2\ \ldots\ C_k$ (C_k: special check byte of length b+c). If an error pattern $E \neq 0$ occurs in the i^{th} information byte than $S_1 = E \neq 0$

$$[S_2\ S_3\ \ldots\ S_k]^T = [T^i_{r-b}\ \Phi_{r-b,b}]E \neq 0$$

If an error pattern $E \neq 0$ occurs in the j^{th} check byte

then $S_j = E$

and $S_\lambda = 0 \qquad \lambda \neq j$

All the errors yield distinct syndrome and therefore correctable. In case of the parity check matrix given in Eq. 2.17 both the

syndromes S_1 and $[S_2 \ldots S_k] \neq 0$ for errors in the information section of the codeword.

For $r \geq 3b$ the GMC code is described by the matrix

$$H = \left[\begin{array}{c|c|c} H_{r,b} & \begin{array}{c} O_{b,b(2^{r-2b}-1)} \\ \hline H_{(r-b),b} \end{array} & I_r \end{array}\right]$$

$$\left[\begin{array}{c|c|c} P_1 & P_2 & I_r \end{array}\right]$$

In general for $r \geq kb$

$$H = \left[\begin{array}{c|c|c|c|c} H_{r,b} & \begin{array}{c} O_{bx} \\ \hline H_{(r-b),b} \end{array} & \begin{array}{c} O_{bx} \\ \hline O_{bx} \\ \hline H_{(r-2b),b} \end{array} & \cdots & \begin{array}{c} O_{bx} \\ \hline O_{bx} \\ \hline \vdots \\ \hline H_{(2b+c),b} \end{array} & I_r \end{array}\right] \quad \text{(Eq. 2.18)}$$

$$\left[\begin{array}{c|c|c|c|c|c} P_1 & P_2 & P_3 & \cdots & P_{k-1} & I_r \end{array}\right]$$

Each partition P_j appears to be a single byte correcting code for information symbols in that particular section of the codeword as they all yield syndromes distinct from other partitions. A single error $E \neq 0$ in the i^{th} byte of partition P_j yields the syndrome

$$S_1 = S_2 = \ldots = S_{j-1} = 0$$

$$S_j = E \neq 0$$

$$[S_{j+1} \ldots S_k]^T = [T^i_{(r-jb)} \; \Phi_{(r-jb),b}] \cdot E \neq 0$$

Since H_{2b+c} is the smallest possible partition capable of correcting single byte error, there is a limit on the possible number of partitions which suggest the maximality of the code. The total number of distinct error patterns of this code is given by according to Eq. 2.18

$$Z = \{\sum_{j=1}^{k-1} (2^b-1)(2^{(k-j)b+c}-1)\} + (k-1)(2^b-1) + (2^{b+c}-1) + 1$$

$$= 2^b-1\{\sum_{i=1}^{k-1} 2^{ib+c}\} + 2^{b+c}$$

$$= 2^{b-1}\, 2^c \{\sum_{i=1}^{k-1} 2^{ib}\} + 2^{b+c}$$

$$= 2^{b+c}(2^b-1)\, \frac{2^{(k-1)b}-1}{2^b-1} + 2^{b+c}$$

$$= 2^r$$

This suggests that the code is perfect.

Example

Given $r = 5 \times 1 + 0 \Rightarrow B = 1, \; c = 0$

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

According to Eq. 2.15

The parity check matrix is structured according to Eq. 2.18

$$H = \left[\begin{array}{ccccccccccccccc|ccccccc|ccc|c|ccccc} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\ 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&1&0&0&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0&1&0&0&1&0&1&1&1&1&1&1&0&0&0&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0&0&1&0&1&1&1&1&0&1&0&1&1&0&0&0&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1&0&0&1&0&1&1&1&0&1&1&1&1&0&0&0&0&1 \end{array}\right]$$

(Eq.2.19)

P_1	P_2	P_3	P_4	I_r

P_1 is constructed using primitive polynomial $x^4 + x + 1$ in $GF(2^4)$

P_2 is constructed using primitive polynomial $x^3 + x + 1$ in $GF(2^3)$

P_3 is constructed using primitive polynomial $x^2 + x + 1$ in $GF(2^2)$

P_4 is constructed using primitive polynomial $x + 1$ in $GF(2)$

We see that the code length is increased in an iterative fashion and is maximal. Also this (31,26) GMC Code provides an alternate method for constructing single error correcting Hamming code and is perfect. A shortened version of the code could be constructed using a non-primitive irreducible polynomial in $GF(2^{r-b})$. In decoders using shift registers this saves decoding time due to the smaller exponent of the generating polynomial.

In the encoding and decoding circuitry each partition is processed in parallel. Shift register implementation for P_1 in our example is shown in Fig. 2.8. The check bits of the information is available after 2^4-1 cyclic shifts of the primitive polynomial $g_4(x) = x^4 + x + 1$ of the largest partition. During decoding the syndrome is obtained by adding the check bits of the information

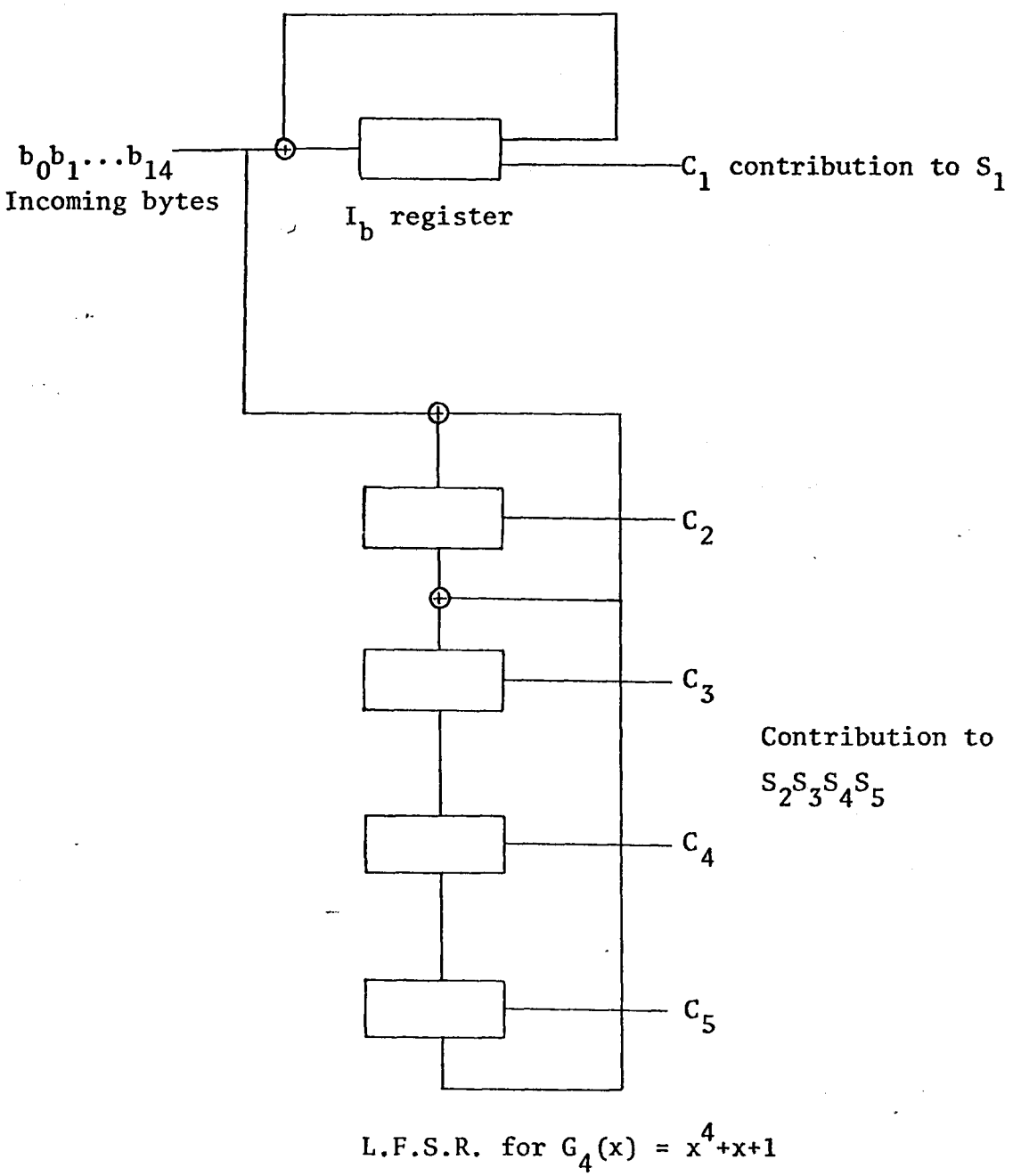

$b_0 b_1 \cdots b_{14}$
Incoming bytes
I_b register
C_1 contribution to S_1
C_2
C_3
C_4
C_5
Contribution to
$S_2 S_3 S_4 S_5$
L.F.S.R. for $G_4(x) = x^4+x+1$

Fig. 2.8

mod 2 to the computed check bits. the I_b portion of the shift register contributes to the syndrome S_1 which gives the error pattern. The linear feedback shift register contributes to the syndrome $[S_2\ S_3\ S_4\ S_5]$ which provides the location of the error. Let v be a codeword given by

v = 10000000000000100000000000000000001

The 5 most significant bits are the check bits of the codeword. Suppose a single error $b_0 = 1$ occurs corresponding to the position P_1 of the codeword. The received codeword becomes

v = 00000000000001000000000000000001.

Bits $b_0 b_1 \ldots b_{14}$ are processed by the circuit (Fig. 2.8). The computed check bits $C_1 = 1$, $[C_2\ C_3\ C_4\ C_5] = 1001$. This is added mod 2 to the check bits of the codeword to give $S_1 = 1$, $[S_2\ S_3\ S_4\ S_5] = 1000$. The error pattern E = 1. The syndrome $[S_2\ S_3\ S_4\ S_5]$ matches the first column of the parity check matrix (Eq. 2.19) giving the location.

The GMC code thus provides us the flexibility of constructing a code to suit any byte arrangement of memory. Moreover, the code processing can be done in parallel, saving precious decoding time.

2.6 Majority Logic Decodable Code

At a time when integrated circuits are being developed rapidly and becoming cheaper, a class of codes which are one-step majority-logic decodable are becoming likely candidates in competition with other codes for computer applications. Although bit is redundancy

is higher compared to other codes the design of the decoder is simple and fast. A class of such codes derived from orthogonal Latin squares has been proposed by Bossen, Hsiao, Chien [11]. The orthogonal property provides a unique feature of adding redundancy systematically such that the decoder can be built in modules.

A code of distance d is said to be one step majority decodable if it is possible to construct 1-1 check sums orthogonal on every digit.. This implies that there must be at least 2t 1's in each column of the parity check matrix for t-error correction. The parity check matrix constructed using Latin squares has exactly 2t 1's in each column corresponding to error in the data bits. The 2t parity check equations formed with any error in the data bit is orthogonal on that particular bit.

Definition: A Latin square of order m is an m x m square array of digits 0,1,...m-1 with each row and column a permutation of the digit 0,1,..m-1. Two Latin squares are orthogonal if when one Latin square is superimposed upon the other every ordered pair of elements appears only once.

Theorem: The maximum error correcting ability of the code constructed from Latin square $t \leq \frac{m\ 1}{2}$. Also these classes of codes have m^2 data bits and 2tm check bits.

Instead of the proof the construction of the parity check matrix is given with an illustrative example.

Given k = 25 m = 5

the orthogonal Latin squares for m = 5 are

$$L_1 = \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \\ 2 & 3 & 4 & 0 & 1 \\ 3 & 4 & 0 & 1 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{matrix} \qquad L_2 = \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 0 & 1 \\ 4 & 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 & 0 \\ 3 & 4 & 0 & 1 & 2 \end{matrix}$$

$$L_3 = \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 4 & 0 & 1 & 2 \\ 1 & 2 & 3 & 4 & 0 \\ 4 & 0 & 1 & 2 & 3 \\ 2 & 3 & 4 & 0 & 1 \end{matrix} \qquad L_4 = \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 1 & 2 & 3 \\ 3 & 4 & 0 & 1 & 2 \\ 2 & 3 & 4 & 0 & 1 \\ 1 & 2 & 3 & 4 & 0 \end{matrix}$$

The parity check matrix is described as

$$H = \left[\begin{array}{c|c} \begin{matrix} M_1 \\ M_2 \\ \vdots \\ M_{2t} \end{matrix} & I_{2tm} \end{array}\right]$$

I_{2tm} is the identity matrix of order 2tm. M_1, $M_2 \ldots M_{2t}$ are sub-matrices of size m x m^2.

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{5 \times 25}$$

$$M_1 = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc} 1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\ 0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1 \end{array}\right]_{5 \times 25}$$

Since the maximum error correcting ability of this code $t = \frac{5+1}{2} = 3$. M_3, M_4, M_5, M_6 are derived from the existing L_1, L_2, L_3, L_4 Latin squares, respectively. The submatrix M_i derived from L_j is given by

$$M_{ij} = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_m \end{bmatrix} \qquad \begin{array}{l} i = 3,4,5,6 \\ j = 1,2,3,4 \end{array}$$

where V_k $k = 1,2...m$ is a row vector of length m^2 derived from L_j

$$V_k = [q^{\mu}_{11}...q^{\mu}_{1m}q^{\mu}_{21}...q^{\mu}_{12m}q^{\mu}_{31}...q^{\mu}_{3m}.....q^{\mu}_{m1}...q^{\mu}_{mm}]$$

If we pick any element μ from the Latin square $0 \leq \mu \leq m-1$

then
$$q^{\mu}_{ij} = \begin{array}{lll} 1 & \mu = \ell_{ij} & i = 1,2...m \\ 0 & \mu \neq \ell_{ij} & j = 1,2...m \end{array}$$

ℓ_{ij} are the elements of the Latin square.

The submatrices M_3, M_4, M_5, M_6 are shown in Fig. 2.9 [11] marked L_1, L_2, L_3, L_4 respectively. The single error correcting (35,25) code is constructed with submatrices M_1 and M_2 (Fig. 2.9a). The Decoder for data bit d_0 is shown. The output of the majority

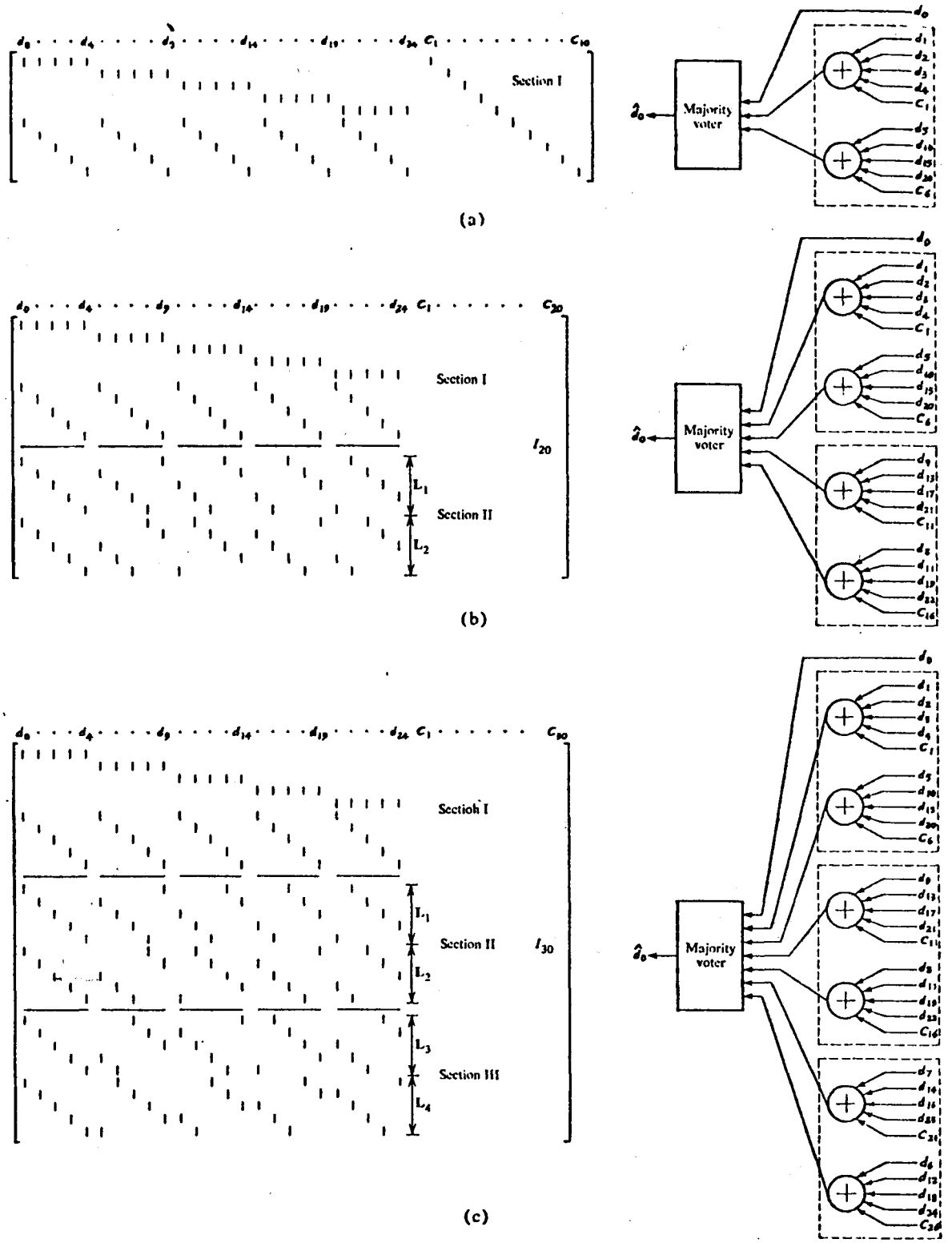

Section I
Majority voter
(a)
Section I
Section II
Majority voter
(b)
Section I
Section II
Section III
Majority voter
(c)

Fig. 2.9

voter is 1 if d_0 is correct and 0 of d_0 is in error. The majority voters for other data bits can be constructed similarly. Referring to Fig. 2.9 the error correcting ability is increased to two by adding section II which includes submatrices M_2 and M_3. Although the number of check bits doubles, the construction is very simple. The circuitry necessary for correcting the additional error is similar to the first and can be added in modular fashion without interfering its mechanization which makes this code a suitable candidate for implementation with I.C. Fig. 29c corresponds to the (55,25) triple error correcting code constructed by adding matrices M_5 and M_6. Since there are no more Latin squares, this is the maximum error correcting capability of the code.

2.7 Summary

In conclusion of this chapter, we have seen some of the error-correcting codes used in computer memory environment. For those memory configurations which are packaged on single bit per card basis, SEC-DED, DEC-TED and one step majority decodable codes were helpful in increasing the reliability of memory. Byte-error-correcting codes formed the basis of correcting errors in memories with multiple bit per card arrangement. Two-redundant codes and Reed-Solomon codes provided character-error-correcting ability. Reed-Solomon codes were not discussed in this chapter because its slow and complicated decoding scheme was a serious handicap. In the next chapter we shall see that this class of codes is not

best suited for photo digital mass memory. Also, we saw the flexibility of the general class of maximal codes whose construction did not depend upon any homogeneous bit arrangement of memory. Emphasis was based throughout on a simple and fast implementation scheme.

CHAPTER 3. ERROR-CORRECTING CODES FOR COMPUTER PERIPHERAL SYSTEM

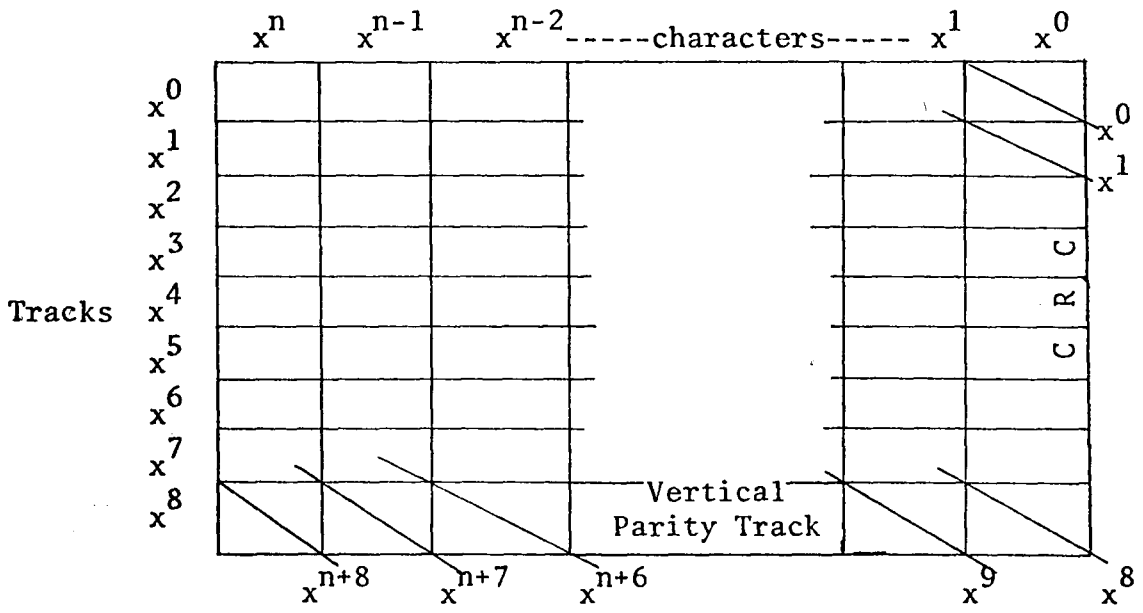
The storage medium in peripheral devices is generally a magnetic tape unit, drum or disc file or an optical unit. They form the bulk of the extended mass memory in computer environment. As more and more information is being placed in mass storage devices, the corresponding improvement in reliability cannot be ignored. Coding techniques are playing a vital role in achieving higher reliability and saving considerable time consuming corrective action in conventional read-write schemes.

3.1 Single-track Correction of Magnetic Tape Unit with Cyclic Redundancy Codes

The IBM/360 system 2400 series uses a standard 1/2" 9 track magnetic tape. Bits are recorded on this tape with a density of 800 bits/inch along the track. Spacing between bits in adjacent tracks is about 40 times larger than those between bits in the same track. This suggests that a single bit in error along any track is likely to affect other bits in the same track rather than those of adjacent tracks. Since bits are so closely packed within a track the probability of burst error patterns occurring simultaneously in one or more tracks is predominant. The errors are generally caused by foreign dust particles, imperfection in the surface coating of the tape and mechanical damage due to handling.

Information is recorded on the tape as characters of 9 bits

across the tracks. The 9th bit is a parity on the other 8 bits and is written on the 8th track of the tape, also called the vertical parity track. The other 8 bits of the character are written from track 0-7. The position of the character along the tape is identified by associating it with a unique power of x. The highest power is attached to the first character in the record, and the lowest power of x is attached to a special character generated from all the previous characters in the track. It functions as a check on the previous characters written in the record. It is also called the cyclic redundancy check or the CRC character. The track positions are also associated with powers of x. In the 9 track tape the powers of x range from x^0-x^8. A record of n character on the tape looks like an array of 9 x n+1 bits. Each bit in the record can therefore be associated with a power of x given by the sum of the track and character power of x to which the bit belongs.



The record is a sequence of n data characters, each 9 bits long consisting of 8 data bits and 1 parity bit. We can also represent the record as a polynomial

$$p(x) = C_n x^n + C_{n-1} x^{n-1} + \ldots C_1 x$$

where the coefficient C_i is a 9 bit character, polynomial

$$C_i(x) = a_0^i + a_1^i x + a_2^i x^2 + a_3^i x^3 \ldots a_8^i x^8$$

and a_k^i is the bit associated with the i^{th} character and k^{th} track. The maximum degree of the polynomial p(x) is n+8. If we now consider a generator polynomial of degree nine, the remainder resulting from the division of p(x) by g(x) would be a polynomial of degree 8 or less which is equivalent to a 9-bit data character that can be added at the end of the information data characters to represent the CRC. The addition of the CRC to the data characters forms a polynomial codeword which is divisible by the generator polynomial g(x). Any burst error along the track will be detected if the code polynomial is not divisible by g(x) after the entire record has been read. If the burst error pattern is confined to a single track it can be corrected with the aid of the parity bits in the vertical parity track also called vertical redundancy check (VRC). Sometimes an undetectable error pattern divisible by g(x) may result. To account for this an additional check character called longitudinal redundancy check (LRC) is written on the record at the end of CRC character. Every bit in the LRC character is an even parity check of all the data bits in the corresponding track. The American National Standard

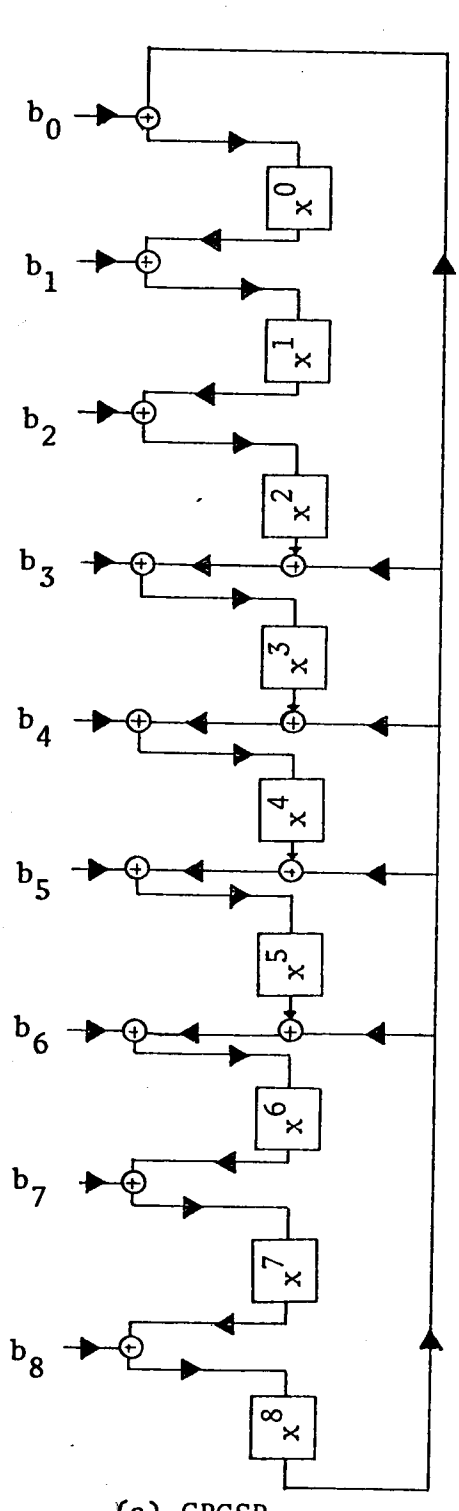
Institute has specified $g(x) = (1+x)G_1(x)$ where $G_1(x) = 1+x+x^2+x^4+x^6+x^7+x^8$ and $g(x) = 1+x^3+x^4+x^5+x^6+x^9$. This g(x) is a symmetric polynomial with (1+x) as a factor. The symmetric property facilitates the read backward mode of operation while the (1+x) factor predetermines the parity of the CRC character and makes error in the CRC character correctable. Also in systems where LRC is used the CRC can be altered by adding an odd wt character to it. This ensures that LRC has odd vertical parity, a desirable trait for identifying the start of the record in the read backward mode of operation. Asymmetric generator polynomials have also been specified. These have larger cycle length than symmetric polynomials and when used along with the symmetric generator polynomial on the data record helps to increase the range of checking ability, especially when bits are recorded at a higher density.

Working example

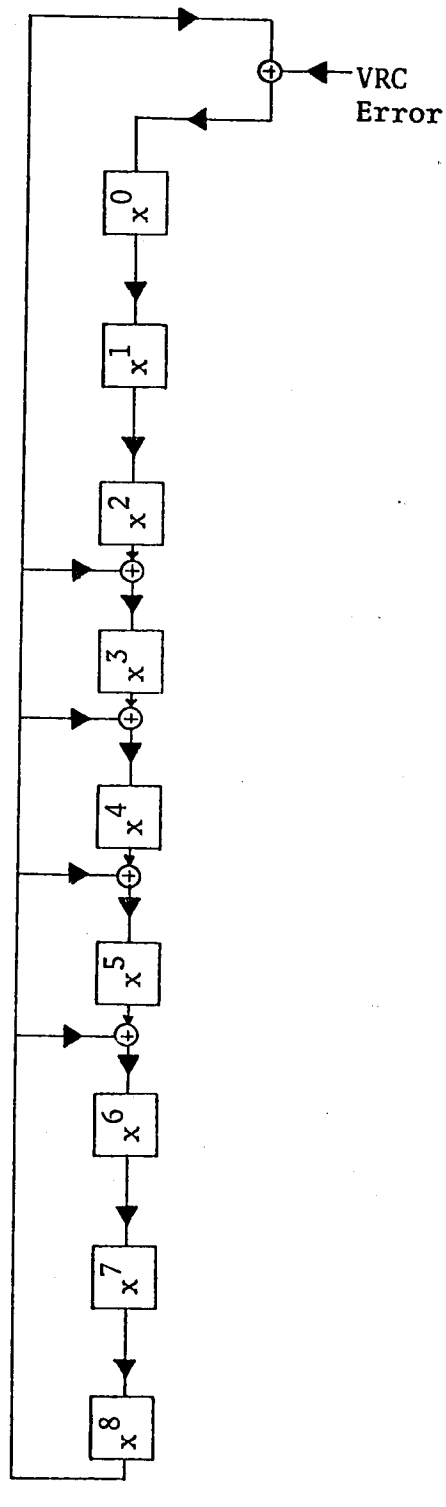
$$g(x) = 1+x^3+x^4+x^5+x^6+x^9$$

The CRC character can be generated by the feedback shift register characterized by g(x) CRCSR (Fig. 31.a).

We will consider here a record of 3 characters written on the tape as shown, using even vertical parity for the data.

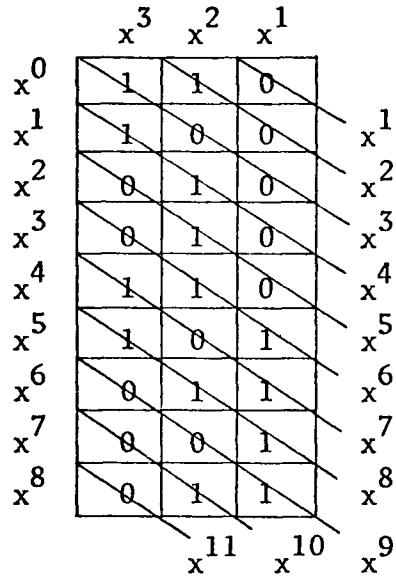


(a) CRCSR



(b) EPSR

Fig. 3.1



Write direction

$$p(x) = x^2+x^3+x^5+x^8+x^9+x^{10}$$

The CRC polynomial $C_0(x)$ is given by the remainder of $[p(x)|g(x)]$

$$\begin{array}{r|l} & x+1 \\ \hline x^9+x^6+x^5+x^4+x^3+1 & x^{10}+x^9+x^8+x^5+x^3+x^2 \\ & \underline{x^{10}+x^7+x^6+x^5+x^4+x} \\ & \quad x^9+x^8+x^7+x^6+x^4+x^3+x^2+x \\ & \quad \underline{x^9+x^6+x^5+x^4+x^3+1} \\ & \quad x^8+x^7+x^5+x^2+x+1 \end{array}$$

In binary C_0 = 111001011

This is written after the last data character on the record. The tape now looks like Fig.(3.2a) below.

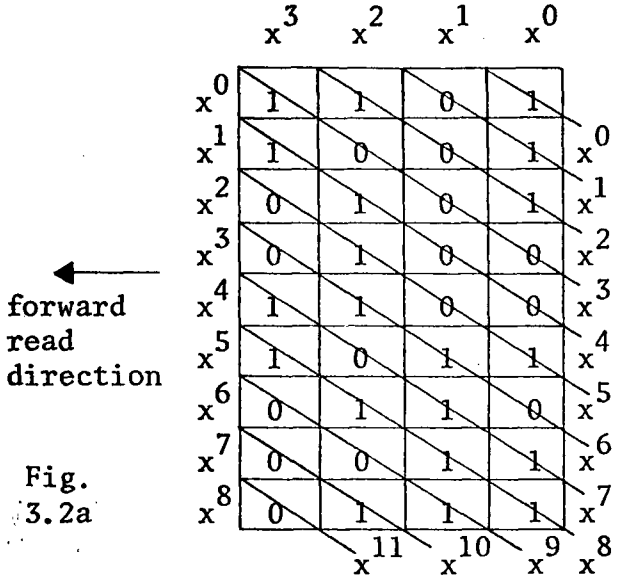


Fig. 3.2a

x^3	x^2	x^1	x^0	LRC
1	1	0	0	0
1	0	0	0	1
0	1	0	0	1
0	1	0	0	1
1	1	0	1	1
1	0	1	1	1
0	1	1	1	1
0	0	1	0	1
0	1	1	0	0

Fig. 3.2b

As (1+x) is a factor of g(x) there are even number of 1's in the record. Since the data has even vertical parity, the parity of C_0 must be even. If an LRC character is written at the end of the record $C_0(x)$ is modified by adding mod 2 $G_1(x)$ which is an odd wt vector. C_0' = 111001011 + 111010111 = 000011100. This ensures that the vertical parity of LRC character is odd (Fig. 3.2b) above. We shall now assume that all bits in track 3 are read as 1's. This corresponds to an error polynomial $E(x) = x^6+x^4+x^3$ and the code polynomial $v(x) = 1+x+x^3+x^7+x^9+x^{10}$. The received polynomial $r(x) = r(x) + e(x) = 1+x+x^4+x^6+x^7+x^9+x^{10}$. The division of v(x) by g(x) is performed by the feedback shift register of Fig. 3.1a. The content of the shift register after division is given by

$$
\begin{array}{r|l}
 & x+1 \\
\hline
x^9+x^6+x^5+x^4+x^3+1 & x^{10}+x^9+x^7+x^6+x^4+x+1 \\
 & \underline{x^{10}+x^7+x^6+x^5+x^4+x} \\
 & x^9+x^5+1 \\
 & \underline{x^9+x^6+x^5+x^4+x^3+1} \\
 & x^6+x^4+x^3 \qquad \equiv 0001101000
\end{array}
$$

As the characters are being read they are checked for even parity. If an odd parity results a 1 is entered in the first bit position of an error pattern feedback shift register (EPSR) characterized by g(x) (Fig. 3.1b) and shifted in synchronism with CRCSR. In our case after 4 shifts the content of ERSR is equal to 110100000. Since the content of CRCSR is not all zero we have detected an error. The error pattern of burst length 9 is indicated in EPSR. The track in which the error occurred can be obtained by simply counting the shifts necessary for the content of EPSR to match that of CRCSR. For our example, after 3 shifts the contents of CRCSR and EPSR are identical, indicating error in the 3rd track. Error correction can now be performed by inverting the bit a_k^3 for k = 1,3,4.

3.2 Double-track Correction of Magnetic Tape Unit with Cyclic Redundancy Codes

We have seen how the CRC polynomial code can correct random and burst error in single tracks. The idea has now extended to double track correction by Malhotra and Fisher [18] to increase reliability. Double channel correction is accomplished if a suitable hardware exists which identifies the two tracks in error. The format of the data recorded on the tape is slightly modified. The length of data characters in a record is limited to 7, and the characters 8 bits long are synbols from $GF(2^8)$. The 8th track is the parity track. In this case the check character is computed

from data characters using an irreducible polynomial in $GF(2^8)$. When the record is read the parity track consisting of 8 bits and the check character of 8 bit length are treated as syndromes S_1 and S_2 of the record. Obviously, if E_i and E_j are the error patterns in tracks i and j respectively, then $S_i = E_i + E_j$ and is computed as the mod 2 sum of the parity track written and read. The other syndrome S_2 is computed by dividing the code polynomial by the generator polynomial using a feedback shift register similar to CRCSR. $S_2 = [x^i E_i + x^j E_j] \bmod g(x)$ where $g(x)$ is the generator polynomial. From S_1 and S_2 the unknowns E_i and E_j can be calculated.

$$E_i = [AS_1 + BS_2] \bmod g(x) \qquad \text{(Eq.3.1)}$$

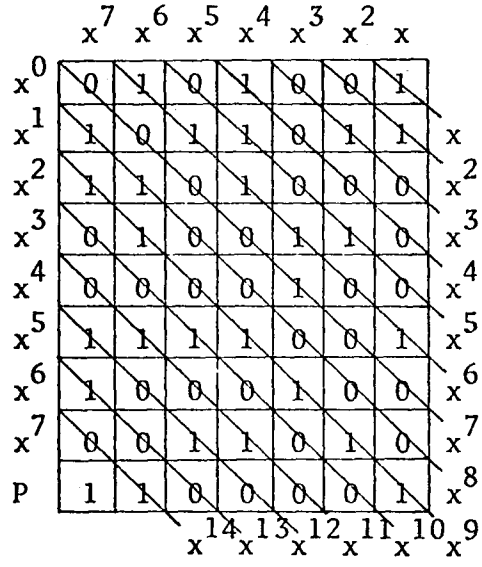
$$E_j = E_i + S_1 \qquad \text{(Eq.3.2)}$$

where $A = \dfrac{x^j}{x^i+x^j} \bmod g(x)$ (Eq. 3.3); $B = \dfrac{1}{x^i+x^j} \bmod g(x)$ (Eq.3.4)

All possible solutions for A and B can be stored in a ROM.

Illustrative example.

$g(x) = 1+x^3+x^4+x^5+x^8$ Vertical parity odd.

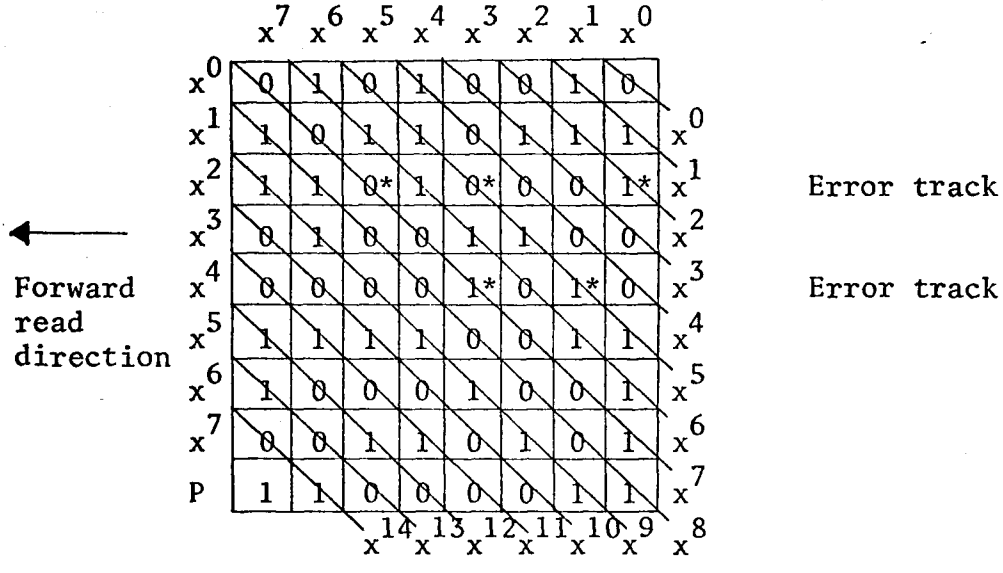


Write direction

data polynomial $p(x) = x+x^2+x^3+x^4+x^5+x^6+x^7+x^9+x^{10}+x^{13}$. The check character is given by Remainder of p(x) divided by g(x) implemented a feedback shift register characterized by g(x).

$$
\begin{array}{r|l}
x^8+x^5+x^4+x^3+1 & x^5+1 \\
 & x^{13}+x^{10}+x^9+x^7+x^6+x^5+x^4+x^3+x^2+x \\
 & \underline{x^{13}+x^{10}+x^9+x^8+x^5} \\
 & \quad x^8+x^7+x^6+x^4+x^3+x^2+x \\
 & \quad \underline{x^8+x^5+x^4+x^3+1} \\
 & \quad\quad x^7+x^6+x^5+x^2+x+1
\end{array}
$$

The check character C_0 = 11100111 is now written at the end of the data characters to form a complete record.



Assuming that error was detected in tracks 2 and 4 as a result of the bits marked with asterisk being inverted (i=2,j=4), the computed parity track during read is equal to 11100000, S_1 = 11000011 $+_2$ 11100000 = 00100011. The received code polynomial $r(x) = 1+x^2+x^3+x^4+x^9+x^{10}+x^{13}$. The syndrome S_2 is computed as follows:

$$
\begin{array}{r|l}
 & x^5+1 \\
\hline
x^8+x^5+x^4+x^3+1 & x^{13}+x^{10}+x^9+x^4+x^3+x^2+1 \\
 & \underline{x^{13}+x^{10}+x^9+x^8+x^5} \\
 & \quad x^8+x^5+x^4+x^3+x^2+1 \\
 & \quad \underline{x^8+x^5+x^4+x^3+1} \\
 & \qquad\qquad x^2
\end{array}
$$

The coefficient A and B in Eq. 3.1 is computed using the formula in equations 3.3 and 3.4. In binary A = 01011001 and B = 00010110. During implementation these are the 8 bit addresses given by the ROM supplied with track pointers. Error pattern E_2 is given by Eq. 3.1. The partial products and sums modulo g(x) is implemented by shift registers. The result Ex-ored with S_1 gives the error pattern E_4.

S_1 = 00100011

S_2 = 00000100

A = 01011001

B = 00010110

g(x) = 100111001

00100011		00010110	
x 01011001		x 00000100	
00100011		01011000	$\equiv BS_2$
00100011000			
001000110000		00101111001011	$\equiv AS_1$
+ 00100011000000		+ 01011000	$\equiv BS_2$
00101111001011	$\equiv AS_1$	00101110010011	$\equiv AS_1 + BS_2$

where x and + stands for shift and addition mod 2.

$$
\begin{array}{r}
\quad\quad\quad\quad 1010 \\
100111001 \;\overline{)\; 101110010011} \\
\quad\quad 100111001 \\
\quad\quad\quad 100101101 \\
\quad\quad\quad 100111001 \\
\quad\quad\quad\quad\; 101001 \;\equiv\; E_2
\end{array}
$$

$E_4 = 00100011 + 00101001 = 00001010$

When E_2 and E_4 are added to the tracks 2 and 4 of the record the correct information is restored. If errors occurred in more than 2 tracks, this scheme is unable to correct the errors. The capability to detect uncorrectable error can be extended by adding additional check characters generated by 9th degree polynomials similar to the generation of the CRC check character at the end of the record.

3.3 Single and Double Track Correction Using Optimal Rectangular Codes

Another code capable of correcting single track error and double track errors with track pointers is the Optimal Rectangular code developed by Hong and Patel [17]. They are being used commercially in models 4, 6, 8 of the IBM 3420 series tape unit. The recording density on these standard $\frac{1}{2}$ inch 9 track tapes is 6250 bits per inch. The code has a rectangular format of 9 rows and 8 columns. Information is recorded as 8-bit bytes along columns. There are 7 information bytes in the code word and a check byte generated from the information byte using an irreducible polynomial

in $GF(2^8)$. This is written on the first column of the rectangular array. Bits in track 0-7 in the array are treated as track bytes. The 8th track is the vertical parity track. Each bit in this track is a vertical parity of the upper 8 bits in the column.

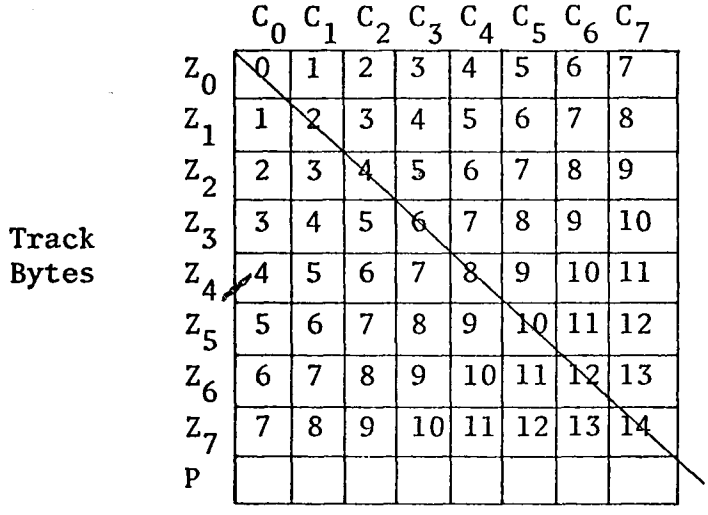


Fig. 3.3

If $Z_i(j)$ is the j^{th} bit in track i and $C_i(j)$ the j^{th} bit in i^{th} character for $0 \leq i \leq 7$ and $0 \leq j \leq 7$, $C_0(j)$ $0 \leq j \leq 7$ represents the check character and $C_i(j)$ $0 \leq j \leq 7$, $1 \leq i \leq 7$, the information characters. If we associate $Z_i(j)$ with the number $x = i+j$ we observe that the array is symmetrical with respect to x (Fig. 3.3). This orthogonal symmetry is used to construct the parity check matrix of the code.

If we recall from the previous chapter Hong and Patel had developed a general class of maximal codes using matrix $H_{r,b}$ for code construction. The ORC code can be represented by an H matrix (Fig. 3.4) which uses $H_{2b,b}$ for code construction. This belongs to the shortened version of the maximal code as α is the root of

H =

Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	P
10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000
01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000
00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000
00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000
00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000
00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100
00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010
00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001
$\alpha^0\alpha^1..\alpha^7$ T^0	$\alpha^1...\alpha^8$ T	$\alpha^2...\alpha^9$ T^2	$\alpha^3...\alpha^{10}$ T^3	$\alpha^4...\alpha^{11}$ T^4	$\alpha^5...\alpha^{12}$ T^5	$\alpha^6...\alpha^{13}$ T^6	$\alpha^7...\alpha^{14}$ T^7	

Fig. 3.4

an irreducible non-primitive polynomial in $GF(2^8)$. The code word using this H matrix is the concatenation of track bytes and parity byte $[Z_0Z_1Z_2Z_3Z_4Z_5Z_6Z_7P]$. Such a code we have seen has single byte error correcting ability. Hence any error in any single track can be correctly decoded. Moreover, if we know the location of the errors we can correct double errors. Due to the orthogonal symmetry of the code the parity check matrix of Fig. 3.4 can be modified to represent the codeword in terms of the information bytes $[C_0C_1C_2C_3C_4C_5C_6C_7P]$. From Fig. 3.3 we find that bit $Z_i(j) = C_j(i)$ and that the powers of α in H matrix (Fig. 2.4) are associated with integer x. In the modified parity check matrix H' (Fig. 3.5) the column that goes with the bit $C_i(j)$ will have the lower half identical to the column of H (Fig. 2.4) associated with bit $Z_j(i)$ and a 1 in the i^{th} row as $0 \leq j \leq 7$ for some i. The parity checking equation becomes $H'[C_0C_1C_2C_3C_4C_5C_6C_7P]^T = \Phi$. The check bytes can now be computed from the information bytes according to the equations

$$C_p = \sum_{i=1}^{7} T^i C_i \qquad \text{(Eq.3.5)}$$

$$P(i) = \sum_{j=0}^{7} C_i(j) \qquad \text{(Eq.3.6)}$$

Eq. 3.5 in polynomial form can be written as

$$C_p(x) = \sum_{i=1}^{7} x^i C_i(x) \mod g(x) \qquad \text{(Eq.3.7)}$$

$H' =$

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	P
11111111	00000000	00000000	00000000	00000000	00000000	00000000	00000000	10000000
00000000	11111111	00000000	00000000	00000000	00000000	00000000	00000000	01000000
00000000	00000000	11111111	00000000	00000000	00000000	00000000	00000000	00100000
00000000	00000000	00000000	11111111	00000000	00000000	00000000	00000000	00010000
00000000	00000000	00000000	00000000	11111111	00000000	00000000	00000000	00001000
00000000	00000000	00000000	00000000	00000000	11111111	00000000	00000000	00000100
00000000	00000000	00000000	00000000	00000000	00000000	11111111	00000000	00000010
00000000	00000000	00000000	00000000	00000000	00000000	00000000	11111111	00000001
$\alpha^0 \ldots \alpha^7$	$\alpha^1 \ldots \alpha^8$	$\alpha^2 \ldots \alpha^9$	$\alpha^3 \ldots \alpha^{10}$	$\alpha^4 \ldots \alpha^{11}$	$\alpha^5 \ldots \alpha^{12}$	$\alpha^6 \ldots \alpha^{13}$	$\alpha^7 \ldots \alpha^{14}$	
T^0	T^1	T^9	T^3	T^4	T^5	T^6	T^7	

Fig. 3.5

Eq. 3.7 can be implemented by a linear feedback shift register characterized by g(x) similar to Fig. 3.1a whereas Eq. 3.6 can be implemented through the usual Ex-Or tree. We are now ready to see the working of ORC with an illustrative example. The generator polynomial selected is $g(x) = 1+x^3+x^4+x^5+x^8$. This is an 8th degree irreducible self reciprocal polynomial in $GF(2^8)$ with the lowest exponent. This property we shall see saves valuable correction time.

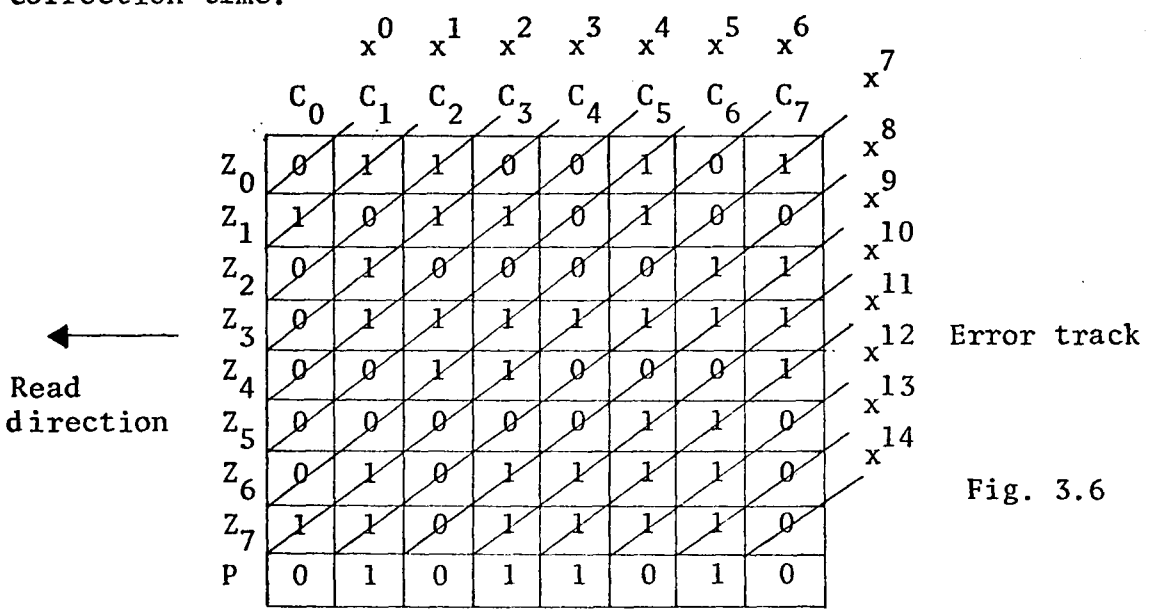


Fig. 3.6

In Fig. 3.6 the information bytes are the characters C_i $1 \leq i \leq 7$. In polynomial form $C_i(x) = a_0^i + a_1^i x + a_2^i x^2 + a_3^i x^3 + a_4^i x^4 + a_5^i x^5 + a_6^i x^6 + a_7^i x^7$, where a_k^i is the bit in the i^{th} character and k^{th} track. Using Eq. 3.7 to compute $C_0(x)$ for the information bytes in Fig. 3.6

$$C_0(x) = [x(1+x^2+x^3+x^6+x^7) + x^2(1+x+x^3+x^4) + x^3(x+x^3+x^4+x^6+x^7) + x^4(x^3+x^6+x^7) + x^5(1+x+x^3+x^5+x^6+x^7) + x^6(x^2+x^3+x^5+x^6+x^7) + x^7(1+x^2+x^3+x^4)] \text{ mod. } 1+x^3+x^4+x^5+x^8$$

where $\oplus$ stands for mod 2 addition

$$C_0(x) = [x+x^2+x^6+x^8+x^9+x^{13}] \bmod 1+x^3+x^4+x^5+x^8 = x+x^7$$

The parity track P is the even parity of the bits in each column according to Eq. 3.6. The complete codeword is now the concatenation of the bytes $[C_0C_1C_2C_3C_4C_5C_6C_7P]$. We shall now assume that an error occurs in track 3 and all the bits in this track are erased. As the erroneous record is read, each character is fed to the feedback register in Fig. 3.7a and also the character along with its parity bit is passed through 9-way Ex-OR gate (Fig. 3.7b) to generate the first bit of the syndrome S_1. The feedback shift register after seven shifts contains the syndrome S_2 and the syndrome S_1 is stored in a buffer. Mathematically the syndromes are calculated as follows:

error pattern E = 011111111 = S_1

error polynomial $E(x) = x^2(x+x^2+x^3+x^4+x^5+x^6+x^7)$

code polynomial $v(x) = x^2+x^6+x^7+x^8+x^9+x^{13}$

received polynomial $r(x) = x^2+x^4+x^5+x^{10}+x^{13}$

$$S_2 = |r(x)| \bmod g(x)$$
$$= 1+x+x^2+x^3+x^4+x^6$$

The content of the feedback register after seven shifts is 01011111. According to the parity check rules of H' $S_1 = E$ and $S_2 = T^iE$ or $S_2 = T^iS_1$ $T^{-1}S_2 = S_1$ $T^{n-j}S_2 = S_1$ where n is the exponent of g(x) and j gives the location of the track in error. Thus if the feedback shift register with content S_2 be shifted k = n-j times the content matches the error pattern S_1. For our

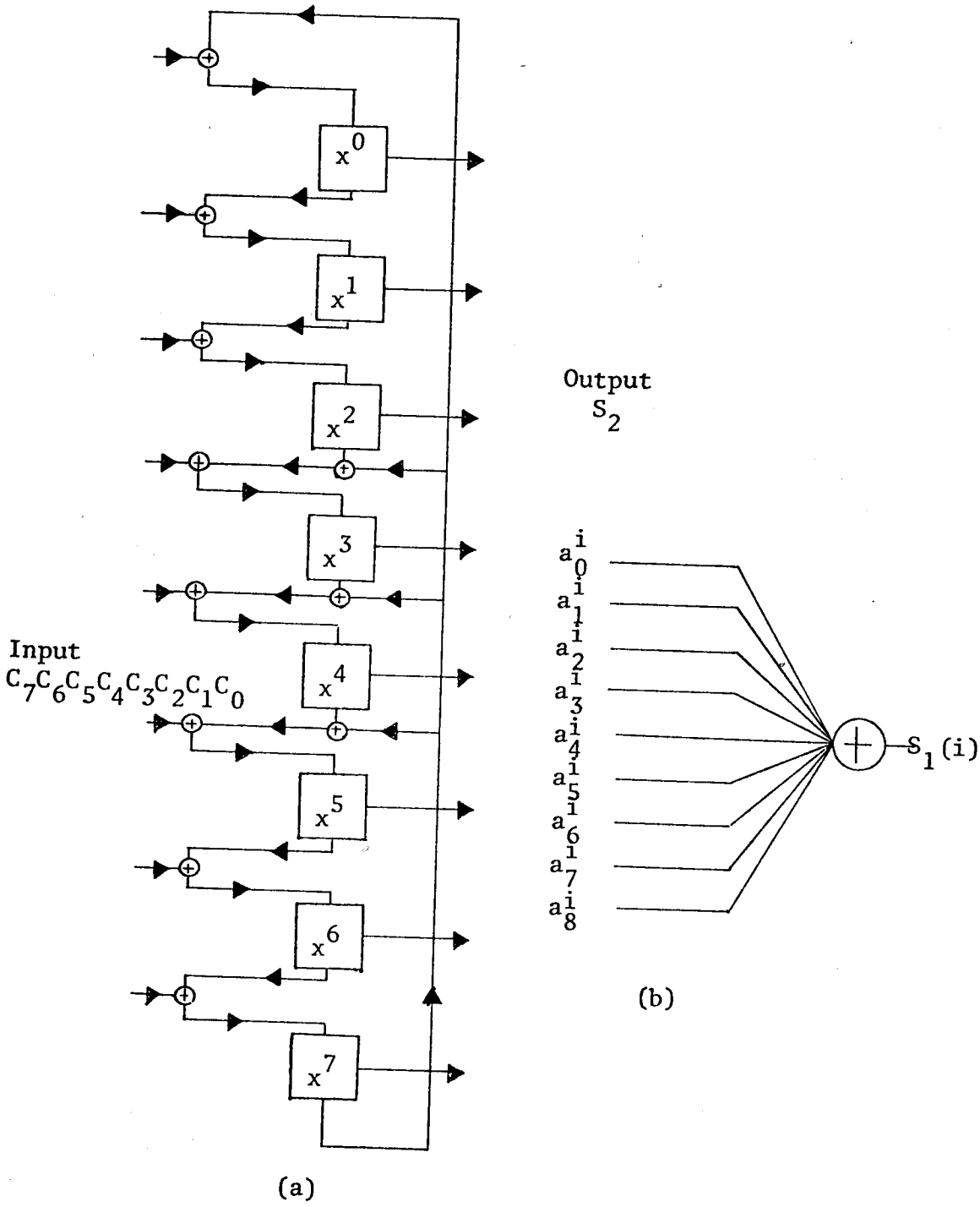

Input
$C_7C_6C_5C_4C_3C_2C_1C_0$
x^0
x^1
x^2
x^3
x^4
x^5
x^6
x^7
(a)
Output
S_2
a_0^i
a_1^i
a_2^i
a_3^i
a_4^i
a_5^i
a_6^i
a_7^i
a_8^i
$S_1(i)$
(b)

Fig. 3.7

example k = 14, n = 17, j = 3. The error magnitude S_1 can now be added to the erroneous track 3 to retrieve the correct information.

The error correction ability of this code can be increased to two if the location of the tracks in error can be identified. Parity checking rules for double error correction using the parity check matrix H' are given by

$$S_1 = e_1 \oplus e_2$$

$$S_2 = \begin{cases} T^i e_1 \oplus T^j e_2 & \text{if } i \neq j \neq 8 \\ T^i e_1 & \text{if } j = 8 \end{cases}$$

From the above two equations e_1 and e_2 can be uniquely determined if i and j are known erroneous tracks.

$$e_1 = S_1 \oplus e_2$$

$$e_2 = \begin{cases} [I \oplus T^{j-i}]^{-1} (S_1 \oplus T^{-1} S_2) & \text{if } i \neq j \neq 8 \\ S_1 \oplus T^{-i} S_2 & \text{if } j = 8 \end{cases}$$

$[I \oplus T^{j-i}]^{-1}$ is an 8x8 matrix connected as an Ex-Or network for j-k=1,2,3...7. The implementation scheme is identical to single track correction. In general, the ORC can be applied to any number of t tracks involving computation with elements from $GF(2^{t-1})$.

3.4 Error-correcting Codes for Single Channel Disc Systems

The codes discussed so far were suitable for encoding information on magnetic tapes. However, in magnetic disc systems information between tracks are distinct and there is essentially

no coordination between them. Basically, therefore, the coding system must serve a single track file containing long serial records. As was the case with magnetic tapes burst errors predominate in high speed disc file. The best known cyclic codes for single burst correction with simple decoding scheme are the Fire codes. These codes are generated by polynomials of the form $g(x) = (x^c-1)\ p(x)$ where $p(x)$ is an irreducible polynomial of degree m and order e and e does not divide c with code length $n = LCM(e,c)$. The code is capable of correcting single burst of maximum length b and detecting simultaneously single burst of maximum length $d \geq b$ provided $m \geq b$ and $c \geq d+b-1$. The decoding can easily be accomplished with r stage feedback shift register characterized by $g(x)$, r being the degree of $g(x)$ using the error trapping technique. Information encoded on the disc track using generator polynomial $g(x)$ appears at both input 1 and 2 of Fig. 3.8. After the entire record has been read the feedback shift register contains the syndrome. If the content of the feedback shift register is zero the record is error free. However, if the record includes a burst error the syndrome is non-zero. Information is shifted out bit by bit and synchronized with a cyclic shift of the feedback register till the output of the OR gate is zero. The rightmost b bits of shift register contain the error pattern. Gate 2 is opened and Gate 1 is closed and the error is added to the next b bits coming out of the buffer. If the leftmost r-b stages of the shift register never contain all zero till

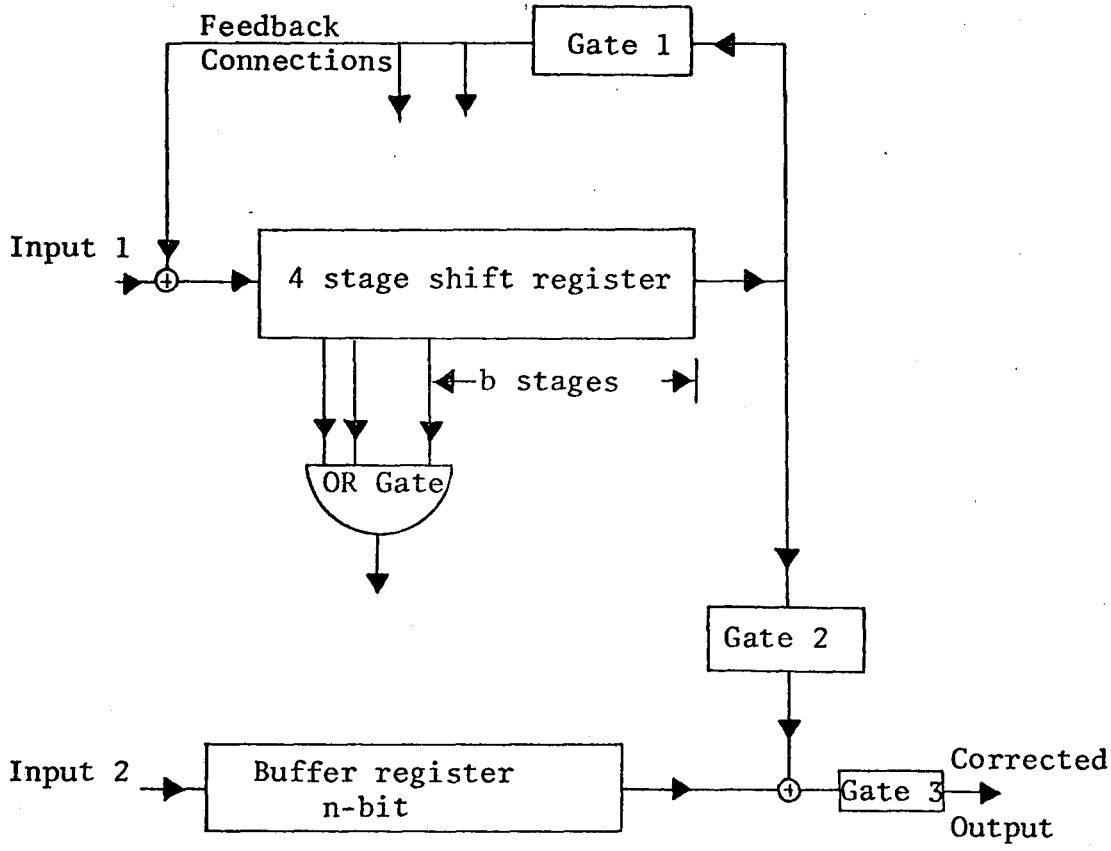

Feedback
Connections
Gate 1
Input 1
4 stage shift register
b stages
OR Gate
Gate 2
Input 2
Buffer register
n-bit
Gate 3
Corrected
Output

Fig. 3.8

the k information bits are shifted out of the record then an uncorrectable burst has been detected.

Although the decoder has simple hardware the delay in shifting records of long length is considerable to make it unsuitable for high speed application. To overcome this handicap Chien [20] has proposed a high speed decoding algorithm using the Chinese Remainder Theorem. The decoding circuit requires additional feedback registers. For example, the Fire code for 19-bit burst correction generated by $g(x) = (x^{37}+1)(x^{19}+x^5+x^2+1)$ would require 19,360,731 shifts for decoding. With high speed decoding algorithm $g(x) = (x^{37}+1)(x^9+x^4+1)(x^{10}+x^3+1)$. It will require two additional feedback registers of length 9 and 10 characterized by x^9+x^4+1 and $x^{10}+x^3+1$. However, 100 percent of all burst up to 9 bits and 99.6 percent of all burst from 10 to 19 bits can be corrected and the other 0.4 percent detected within 1060 shifts.

3.5 Error-correcting Schemestor Multi-channel Disc Systems

Recently an error correcting scheme for multi-channel disc systems having relatively fast access time has been proposed by Malhotra [18]. The scheme is capable of correcting single errors in any one track and single errors in two tracks if track pointers are available. Information is written on the disc in the following format:

	Tracks	Data bits				
Data Tracks	0	D_{01}	D_{02}	...	D_{08}	
	1	D_{11}	D_{12}	...	D_{18}	
	2					
	3					
	4					
	5					
	6	D_{61}	D_{62}	...	D_{68}	
Check Tracks	7	C_1	C_2	...	C_8	(Check track)
	8	P_1	P_2	...	P_2	(Parity track)

The 8 data bits along a track are divided to form two clusters of 4 bits each such that each cluster represents an element in $GF(2^4)$. Each such element from track 0-8 are combined to form a codeword according to the parity check matrix.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 0 & 1 \end{bmatrix}$$

proposed by Bossen for b-adjacent bit correction, where α is the primitive element in $GF(2^4)$. The codeword is constructed as $W = [D_{01}...D_{04}\ D_{11}...D_{14}\ D_{21}...D_{24}\ D_{31}...D_{34}\ D_{41}...D_{44}\ D_{51}...D_{54}\ D_{61}...D_{64}\ P_1...P_4\ C_1...C_4]$ such that $WH^T = \phi$. The check bits $C_1...C_4$ can be computed from the data bits using an Ex-OR network connected according to H and the parity bit through a 7-way Ex-OR Gate. For single track correction, the parity checking rules are given by

$S_1 = e_i$	$0 \leq i \leq 6$	$S_1 = 3_i$	$i = 7$	$S_1 = 0$	$i = 8$
$S_2 = {}^{i}e_i$		$S_2 = 0$		$S_2 = e_i$	

The solution of i for all possible values of S_1 and S_2 can be stored in a 256 word x 4 bit ROM chip such that S_1 and S_2 form an 8-bit address to correct the error. S_1 is added to the faulty track indicated by the output of the ROM.

For double track correction with available track indicators i and j the parity checking rules become

$$S_1 = \begin{cases} e_i + e_j & i \neq j \neq \\ e_i & j = 8 \quad 0 \leq i \leq 7 \end{cases} \qquad \text{(Eq. 3.8)}$$

$$S_2 = \begin{cases} \alpha^i e_i + {}^j e_j & i \neq j \neq 8 \\ \alpha^i e_i + e_j & j = 8 \quad 0 \leq i \leq 6 \\ \alpha^i e_i & j = 7 \quad 0 \leq i \leq 6 \\ e_j & i = 7 \quad j = 8 \end{cases} \qquad \text{(Eq. 3.9)}$$

In all cases e_i and e_j can be uniquely determined from S_1 and S_2.

The implementation scheme for error in single track is similar to the 2-redundant Hamming type code. If S_1 and S_2 computed by the Ex-OR tree as an 8-bit address is nonzero, the magnitude of the error is given by the first four bits and the location of the error is provided by testing the condition $S_2 + {}^i S_1 = 0$. The basic correcting scheme is shown in Fig. 3.9.

Correction of errors in two track can be illustrated with an example.

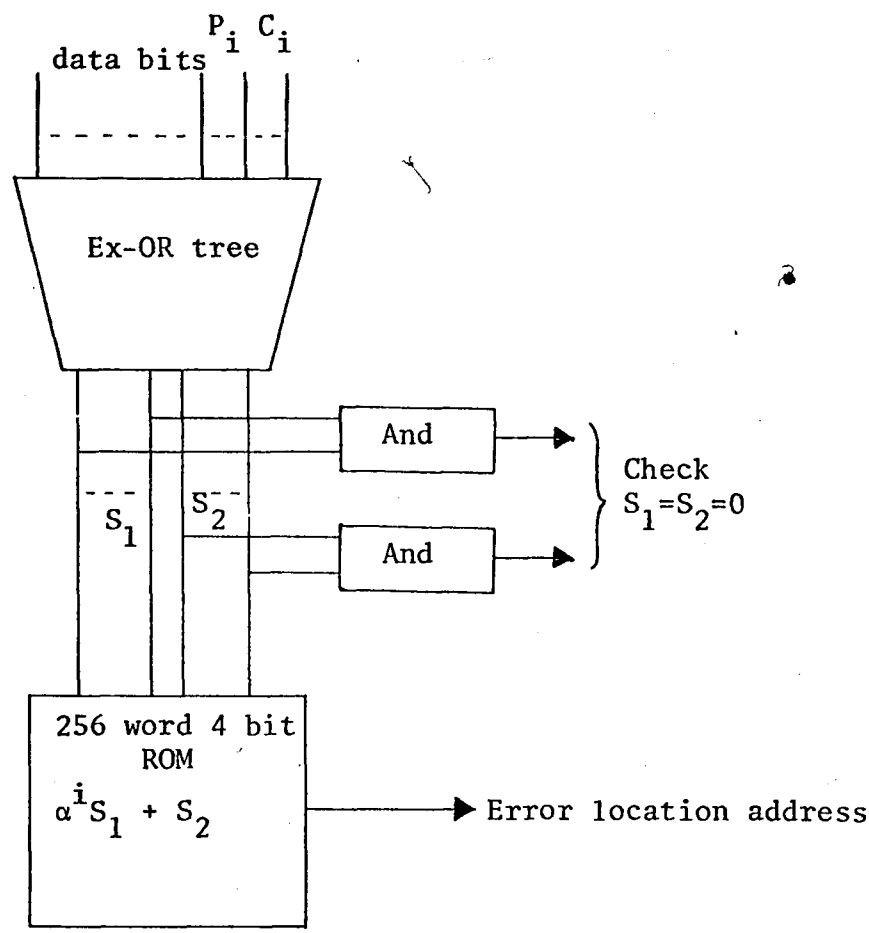

P_i C_i
data bits
Ex-OR tree
And
And
Check
S_1=S_2=0
S_1
S_2
256 word 4 bit
ROM
$\alpha^i S_1 + S_2$
Error location address

Fig. 3.9

Track		Data	
0		1001	
1	(0010)	1101*	
2		0101	
3		1100	Data elements
4	(1100)	1010*	
5		0011	
6		1111	
7		1011	(Parity element)
8		0010	(Check element)

Assuming that elements marked with asterisk are altered and the data elements are read as 0010 and 1100 instead. The hardware detects error in tracks 1 and 4. Using Eq. 3.8 and Eq. 3.9 for $i = 1$, $j = 4$, we get

$$S_1 = e_1 \oplus e_4 \qquad \text{(Eq.3.10)}$$

$$S_2 = \alpha^1 e_1 \oplus \alpha^4 e_4 \qquad \text{(Eq.3.11)}$$

The syndromes S_1 and S_2 are computed through the Ex-OR tree as $S_1 = 1001$, $S_2 = 0111$.

Solving for e_1 and e_4 from Eq. 2.10 and Eq. 2.11 we get

$$e_1 = S_2 \oplus \alpha^4 S_1$$

$$e_4 = S_1 \oplus e_1$$

For our example $e_1 = 0111 \oplus 1000 = 1111$

$$e_4 = 1001 \oplus 1111 = 0110$$

When e_1 and e_4 are added mod 2 to the elements in track 1 and 4, correction is achieved.

3.6 Error Correction in Photo Digital Storage Systems

The error-correcting scheme used in IBM Photo Digital Storage System (PDSS) designed by Oldham, Chien, Tang [21] uses the Reed-Solomon code. These codes can be constructed to correct random multiple burst errors but require a complicated decoding scheme which prevents it from being used in many systems.

The photodigital storage system is a photographic system in which data is recorded on a photographic film chip by an electron beam. Each bit occupies an area 14μ by 16μ and is written sequentially in lines. The film is developed and stored in small boxes called cells which are stored in file modules. The data can be read optically with a flying spot scanner. Errors in such systems generally result from surface contamination of film chips during recording and development or during subsequent storage and reading. Consequently, 90 percent of the errors seem to be isolated single errors or multiple burst errors. A search among the different burst correcting codes conducted showed that the Reed-Solomon code [22] over $GF(2^6)$ with 11 characters of redundancy was best suited to meet the error control requirements. The code has a capacity to correct up to 5 character errors and detect up to 6 in a single memory line and is generated by the following polynomial:

$$g(x) = (x-\alpha^{58})(x-\alpha^{59})(x-\alpha^{60})(x-\alpha^{61})(x-\alpha^{62})(x-\alpha^{0})$$
$$(x-\alpha)(x-\alpha^{2})(x-\alpha^{3})(x-\alpha^{4})(x-\alpha^{5})$$
$$= x^{11}+\alpha^{14}x^{10}+\alpha^{59}x^{9}+\alpha^{6}x^{8}+\alpha^{28}x^{7}+\alpha^{54}$$
$$x^{6}+\alpha^{54}x^{5}+\alpha^{28}x^{4}+\alpha^{6}x^{3}+\alpha^{59}x^{2}+\alpha^{14}x+\alpha^{0}$$

where α is a primitive element in $GF(2^6)$. Incoming data to be recorded are broken up into 300 bit lines to which is appended a 12 bit line number, which can be treated as 52 six bit characters considered to be elements of $GF(2^6)$. The line is now treated as a polynomial and encoded using the feedback circuit in Fig. 3.10. Each line and block in the figure handles six data bits. The remainder in the encoder is appended to the data to form a complete (378,300) code of length 63 characters and recorded as a line of memory on the film chip.

The complex decoding scheme has been simplified by using a hybrid hardware and software technique. Encoding, calculation of power sums and error detection is done by hardware whereas the correction is implemented by software using a control processor on a short time shared basis which can be interrupted to handle the error correcting routine. When a line is read it is divided by the factors of the generator polynomial instead of g(x) and the power sums are individually transmitted to the processor. If all the power sums are not zero an error has been detected. A feature of the correcting routine is to assume the number of errors rather than solve for it. Initially it is assumed that only one character is in error which requires only two power sums for error correction. This computation is completed in 0.34 ms less than the time needed to read one memory line. After trying single-error correction and rereading several times, two error correction is attempted. If this fails then three error correction is attempted. The execution time for double and triple error correction

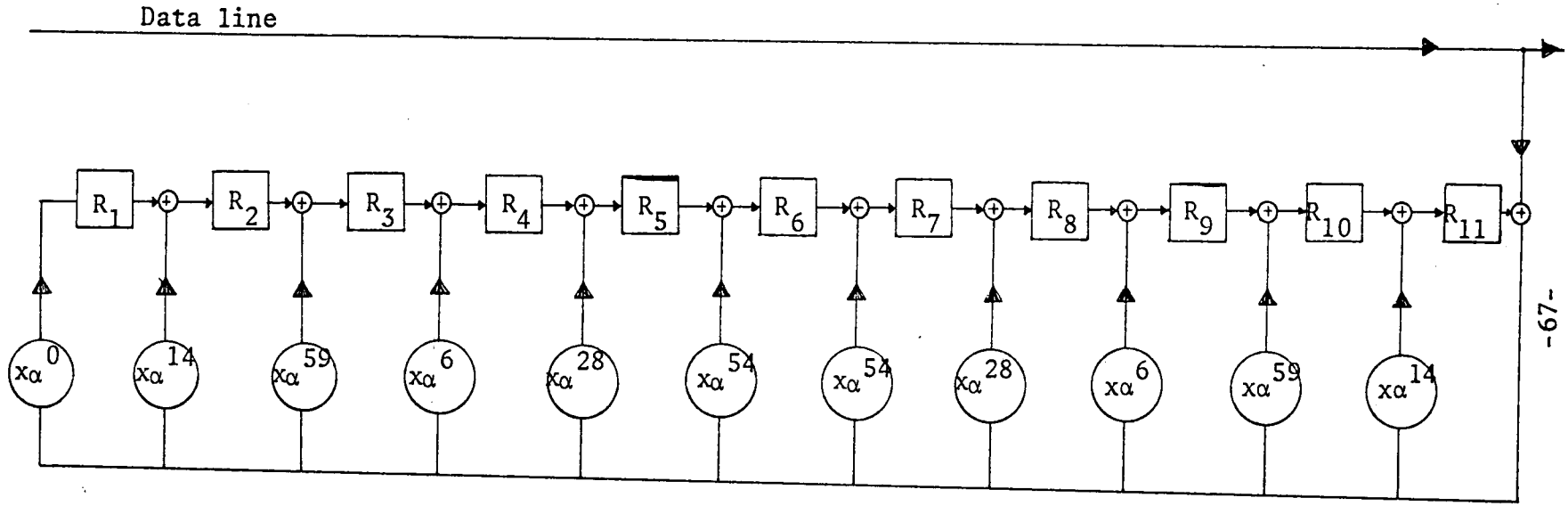

Data line
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
xα0
xα14
xα59
xα6
xα28
xα54
xα54
xα28
xα6
xα59
xα14

Fig. 3.10

is 3.5 and 16 ms. The correction of four and five errors is slow, requiring 25 ms and 37 ms respectively, but is seldom attempted as 99.5 percent of the time single character correction is essential.

The scheme has been effectively demonstrated in the IBM Photo Digital Storage System and it is found that only one in 2.14 million lines contain non-decoded errors.

3.7 Summary

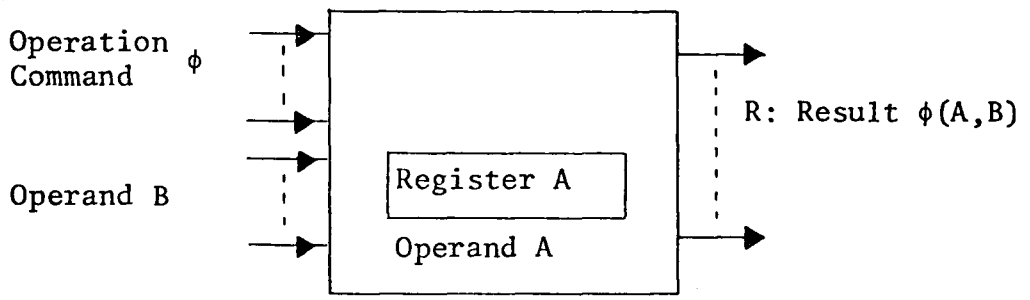
In this chapter we saw the use of error correcting codes in computer peripheral devices. The three main devices covered were the magnetic tape drives, magnetic disc drives and photodigital mass memory. CRC and ORC codes were found applicable to magnetic tape units. In magnetic disc systems Fire codes with high speed decoding could be used for single channel. Malhotra and Fisher (Ref. 18) proposed a practical error correcting scheme for multi-channel disc systems. Reed-Solomon codes were best suited for photodigital mass memories and employed both hardware and software to simplify the decoding procedure.

CHAPTER 4. ERROR-CORRECTING CODES FOR COMPUTER ARITHMETIC PROCESSORS

The arithmetic processor is the part of the computer where all the arithmetic operations, such as complement shift, rotate, add, subtract, multiply, divide, etc., are done. Any error in this area may lead to an erroneous result, causing total system failure if no protective redundancy is provided. As in the memory and peripheral system, all arithmetic operations are in binary mode with the difference that the error correcting code used in arithmetic processors may not preserve its mathematical structure under them. In the light of this observation the codes discussed so far are not suitable for computation in processors. The first major contribution for the development of codes suitable for arithmetic processors were proposed by Diamond and Brown in the early sixties. Many of the concepts, however, have been borrowed from algebraic codes which were developed earlier.

4.1 Nature of Errors in Arithmetic Processors

In the organization of the arithmetic processor some operations are done on single operand like complement and shift, while others like subtract and multiply would require two operand. For purpose of computation, the operands and the operation code must be specified to the arithmetic processor and the result checked.



The input operand B, the internal operand A and the result R are each assumed to be in binary digits. The operation code could be k bits long where k is sufficiently long to accommodate all the possible operations of an arithmetic processor. If Φ is an odd operation then R may represent the sum A+B modulo m denoted as $|A+B|_m$ where $m = 2^n$ for two's complement or $m=2^n-1$ for one's complement arithmetic. Simple arithmetic logic faults may be caused by $E = \pm 2^j$. An error is said to occur whenever the actual outpur $R' = (r'_{n-1}r'_{n-2}\ldots r'_0)$ differs from the expected value $R = (r_{n-1},r_{n-2}\ldots r_0)$ due to the error pattern $E = (e_{n-1},e_{n-2}\ldots e_0)$ where $e_i = r_i - r_i'$ for $i = 0,1\ldots n-1$. For binary output r_i' and r_i can be only 0 or 1 and consequently e_i can be 0,-1,1. As the length of the registers in the processor are of finite length R, R' and E are considered as elements in the finite ring of integers modulo m denoted by $Z_m = \{0,1,\ldots m-1\}$. The modular arithmetic weight of an element N Z_m is given by

$$W_m(N) = \min(W(N),W(m-N))$$

where W(N) is the binary arithmetic weight of the integer N

expressed in the nonadjacent form given by

$$N = \sum_{i=0}^{n-1} a_i r^i \qquad a_i = 0, 1, \text{ or } -1$$

such that $a_i a_{i+1} = 0$ for $i = 0,1,\ldots n-2$. The modular arithmetic wt corresponds to the Hamming wt. Similarly the modular distance between N_1 and N_2 is given by

$$D_m(N_1,N_2) = W_m(N_1-N_2)$$

which plays the same role as the Hamming distance in error control. All errors of modular wt d in Z_m is denoted by a set V(m,d) and all errors of not less than equal to d by U(m,d). For example,

V(31,1) = U(31,1) = {1,2,4,8,16,15,23,27,29,30}

V(31,2 = {V(3,5,6,7,9,10,11,12,13,14,17,18,19,20,21,22,24,25,26,28}

U(31,2) = {V(31,1), V(31,2)}

4.2 Checking Arithmetic Operations Using Residue Codes

The class of codes found most suitable for arithmetic processors are residue codes. Diminished binary complement arithmetic is used for most arithmetic codes as it is easy to implement. Each of the registers are n binary bits in length. The range of numbers that can be represented $-2^{n-1}+1$, $-2^{n-1}+2\ldots1,0,1,2,\ldots2^{n-1}-1$, zero has two representations all n 0's and all n 1's. The operations are done with end around carry.

A schematic diagram of the processor and check circuitry is shown in Fig. 4.1 [23]. The check circuitry is derived to check add, complement, shift and rotate operations. The processor

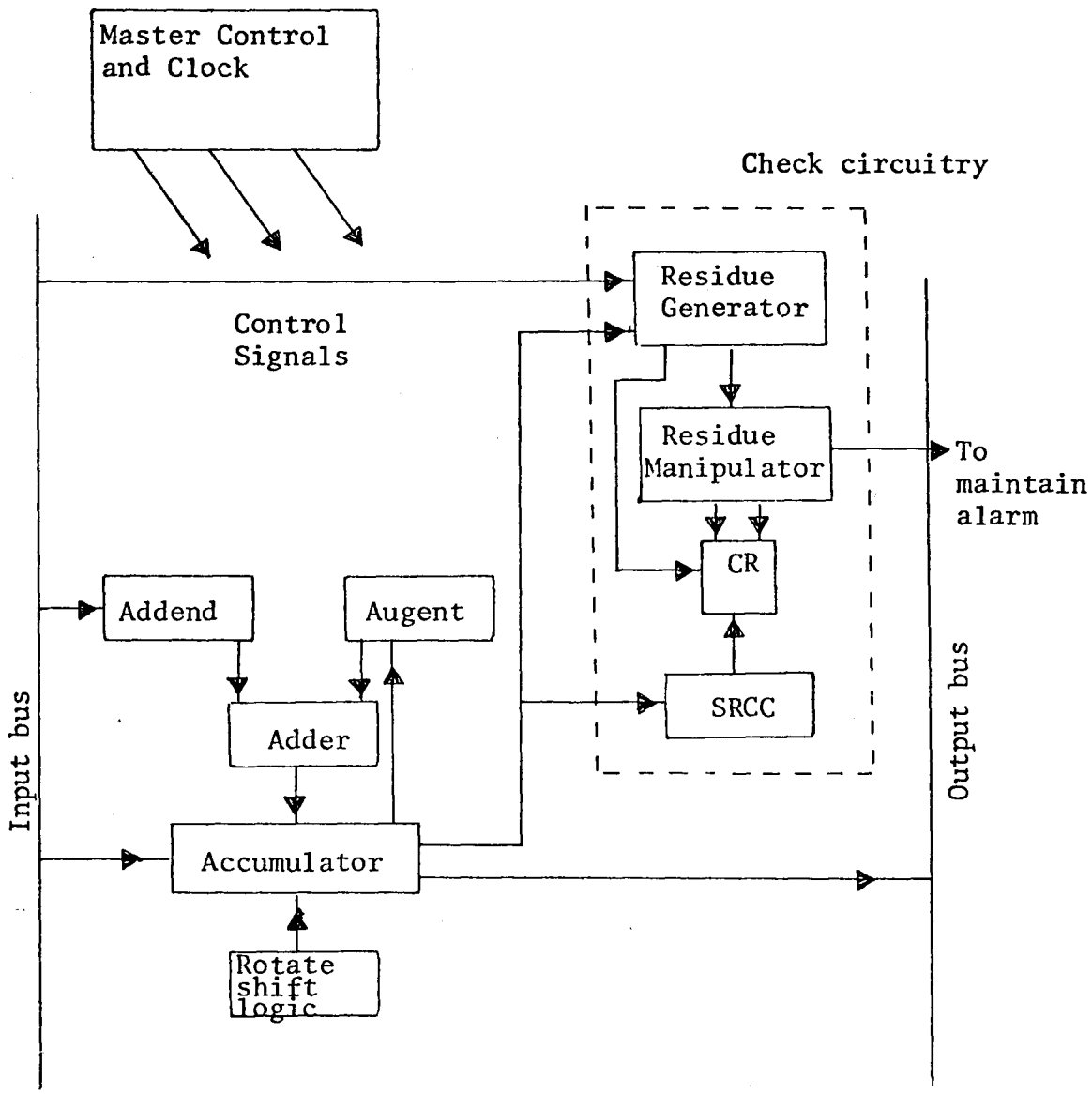

Master Control and Clock
Check circuitry
Residue Generator
Control Signals
Residue Manipulator
To maintain alarm
CR
Addend
Augent
Input bus
Adder
SRCC
Output bus
Accumulator
Rotate shift logic

Fig. 4.1

part in Fig. 4.1 is of.the order of 1000 gates and the check circuitry is realized by less than 400 gates increasing the hardware cost by 40 percent. Any increase in the size of the processor results in a proportional increase in the checker of the order of 30 to 35 percent.

The scheme of Fig. 4.1 employes residue checking. If we consider the operation of complementation of the n-bit accumulator an integer N in the accumulator it is operated upon under the control of the master controller to produce $\phi(N) = M-N$ where $M = 2^n-1$. Before this instant of time the quantity $|N|_r$ which is the least non-negative integer congruent to N modulo r is generated by the residue generator and stored in the check register, r is generally chosen to be 2^k-1 for (k = 2,3,4...) so that modulo r can be accomplished without actual division. Also, r is chosen such that 2^k-1 divides 2^n-1 as this simplifies the checking logic. Next the quantity $|M-N|_r$ is determined and compared with quantity $|N|_r$ in the check register

$$|M-N|_r = ||M|_r - |N|_r|_r$$
$$= |N|_r \qquad \text{as } |M|_r = 0$$

Any disagreement in the comparison initiates an alarm and interrupts the program.

The arrangement in which the processor operates on A and the checker operates on $|N|_r$ can provide only error detection but no correction.

4.3 Single Error Correction with AN Code

Representing an integer N by the product AN where A is a suitable prime integer is known to yield error detection and correction if A is an odd prime and 2 is a primitive element in the field of integers mod A, then correction of single errors is realized if N is constrained to the range

$$0 \leq N < [2^{(a-1)/2} + 1]/A$$

This class of codes is called the Brown Peterson code [3]. Each codeword has a minimum weight 3 and is therefore single error correcting. Every error in V(M,1) has a unique syndrome in GF(A) and therefore perfect like the single error correcting Hamming code.

4.4 Multiple Error-detection and Correction with Mandelbaum Barrow Code

Mandelbaum and Barrow [3] discussed codes using large distance by choosing $A = \frac{2^{p-1}-1}{p}$ for a suitable prime P. These codes provide multiple error detection and correction. The range of these codes is, however, too small for application to computer arithmetic. Besides, the AN codes like the Brown Peterson code are questionable from the practical point of view of computer arithmetic.

4.5 Biresidue Codes for Single Error Correction

Rao [24] extended the scheme of detecting errors by residue checking to correcting single errors in the accumulator by using two residue checkers instead of one. The code is called the Biresidue code. An integer N is coded as a 3 tuple denoted by $(N, |N|_A, |N|_B)$ where A and B are two relatively prime integers. The addition of two codewords defined as $(N_1, |N_1|_A, |N_1|_B)$ is equal to $(|N_1+N2|_M, |N_1+N_2|_A, |N_1+N_2|_B)$. The three components of the code are called the accumulator part, the checker A and the checker B. The syndrome for a triple (X,Y,Z) with respect to moduli A and B denoted as S(X,Y,Z) is a pair (S_a,S_b) where $S_a = |X-Y|_A$, $S_b = |X-Z|_B$. There are three sources of error. If the error is in the accumulator X' = X+e and S(X'YZ) is a pair (S_a,S_b) where $S_a = |X'-Y|_A$, $S_b = |X'-Z|_A$ is equal to $(|e|_A,|e|_B)$. The error goes undetected if $|e|_A = |e|_B = 0$. If the error is in the checker A, the erroneous codeword is denoted by (X,Y',Z). Y' = Y+e and $(S_a,S_b) = (|-e|_A,0)$. Similarly, an error in checker B results in the syndrome $(S_a,S_b) = (0,|-e|_B)$. If we consider class of error e such that $e_A \neq 0$, $e_B \neq 0$ then the error in any component is detected and located. For any single error in the n-bit accumulator $e = \pm 2^j$ for some j = 0,1,...n-1, there is a distinct syndrome pair provided n is no greater than the ab where $A = 2^a-1$, $B = 2^b-1$ are the two residue bases.

Example: $a = 2^3-1 = 7$, $b = 2^4-1 = 15$, $n = ab=12$

j	$(\|2^j\|_7, \|2^j\|_{15})$	$(\|-2^j\|_7, \|-2^j\|_{15})$
0	(1,1)	(6.14)
1	(2,2)	(5,13)
2	(4,4)	(3,11)
3	(1,8)	(6,7)
4	(2,1)	(5,14)
5	(4,2)	(3,13)
6	(1,4)	(6,11)
7	(2,8)	(5,7)
8	(4,1)	(3,14)
9	(1,2)	(6,13)
10	(2,4)	(5,11)
11	(4,8)	(3,7)

The 12-bit accumulator has 22 single errors of magnitude $|2^j|$ j = 0,1,...11. Each of these errors is an element in the error set $V(2^{12}-1,1)$ and is associated with a distinct syndrome pair. Thus single errors in the accumulator can be corrected. Since $|\pm 2^j|_7 \neq 0$ and $|\pm 2^j|_{15} \neq 0$, the range of the codeword is $M = 2^{12}-1$. Every operation ϕ in the accumulator has a corresponding operation ϕ_A and ϕ_B in the two residue checkers RCA,RCB respectively. Let us consider the codeword $(4051, |4051|_7, |4051|_{15}) = (4051,5,1)$. 4051 is the number in the accumulator and 5,1 are the numbers in the check registers CRA,CRB of RCA and RCB respectively. The residue checkers are similar to the check circuitry shown in Fig. 41. In our example, CRA has length 3 and the length of CRB is 4. If we define ϕ as complementation mod M, then ϕ_A, ϕ_B are complementation mod A and mod B respectively.

$\phi(4051) = 4096-4051 = 44$, $\phi_A(5) = 2$, $\phi_B(1) = 14$. The result of the operation is checked by calculating the syndrome (44,2,14), $S_a - |44-2|_7 = 0$, $S_b - |44-14|_{15} = 0$. Thus no error is detected. If we now suppose an error $E = 2^3$ occurs in the accumulator, the syndrome (52,2,14) is given as $(|52-2|_7, |52-14|_7) = (1,8)$. As both S_a and $S_b \neq 0$ we detect an error in the accumulator. Error correction can be achieved because the syndrome has 1-1 correspondence with the error magnitude $+2^3$. The syndrome decoder is shown in Fig. 4.2. E_0, E_1, E_2, E_3 are outputs denoting the type of error

$E_0 = 1$ no error ($S_a = 0$, $S_b = 0$)

$E_1 = 1$ error checker A ($S_a \neq 0$, $S_b = 0$)

$E_2 = 1$ error checker B ($S_a = 0$, $S_b \neq 0$)

$E_3 = 1$ error in accumulator ($S_a \neq 0$, $S_b \neq 0$)

I is the sign indicator

$I = 0$ if E is positive

$= 1$ if E is negative

P_i is the error position

$P_i = 1$ if $E = 2^i$ for i - 0,1,...n-1

$= 0$ otherwise

In our example $a_0 = 1$, $a_1 = 0$, $a_2 = 0$, $b_0 = 0$, $b_1 = 0$, $b_2 = 0$, $b_3 = 1$, E is positive as SRA and SRB have only 1 nonzero bit. I is therefore zero. $P_i = 1$ for $i = 3$ and $P_i = 0$ for $i \neq 3$. The error pattern is therefore 000000001000. This is subtracted

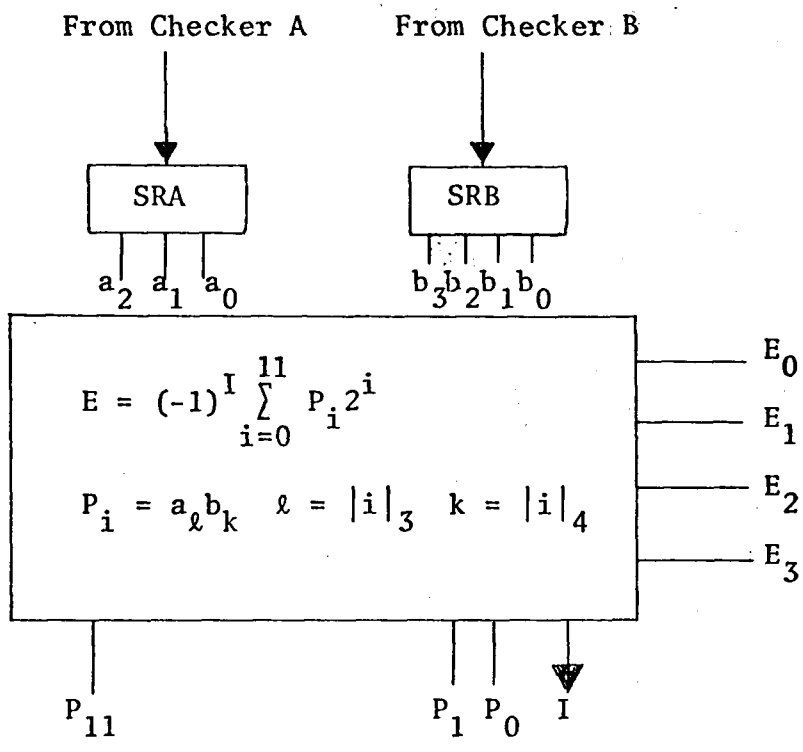

From Checker A
From Checker B
SRA
SRB
a_2 a_1 a_0
b_3 b_2 b_1 b_0
$E = (-1)^I \sum_{i=0}^{11} P_i 2^i$
$P_i = a_\ell b_k \quad \ell = |i|_3 \quad k = |i|_4$
E_0
E_1
E_2
E_3
P_{11}
P_1 P_0 I

Fig. 4.2

from the content of the accumulator to obtain the correct result using 1's complement arithmetic.

52	000000110100	Incorrect result
-8	111111110111	
	000000101011	
	1	
44	000000101100	Correct result

The residue codes are also known as separate codes as the arithmetic unit and the checkers operate independently in the sense that faults in any one unit will not normally contaminate the other. The range of information under error control $0<N<AM$ is much larger than $0<N<M$ for nonseparate codes. Implementation is further simplified by using 1's complement arithmetic and selecting the moduli checkers as 2^x-1 such that x divides n.

4.6 Multiresidue Codes

The Biresidue code was capable of detecting as well as correcting all single errors in the processor. Two residue checkers were required for the code. Now if we increase the number of residue checkers we can correct multiple errors in the processors through the use of the multiresidue code.

The multiresidue code [25] also falls into the class of separate code and is represented by a (k+1) tuple as follows

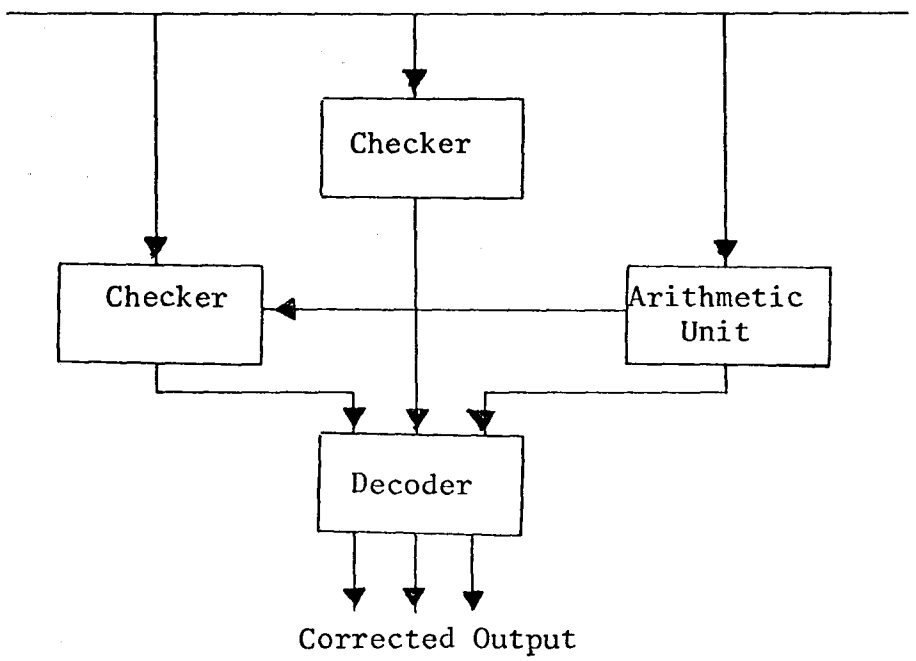
$$x = [x, x_1, x_2, \ldots x_k]$$

where $x_i = |x|_{m_i}$ is the i^{th} check and least non-negative residue of the information x mod m_i for $1<i<k$. The m_i are called the

check bases and often are pairwise relatively prime. The sum of two codewords X,Y is represented as follows

$$X + Y = [|x+y|_{m_0}, |x_1+y_1|_{m_1} \ldots, |x_k+y_k|_{m_k}]$$

The addition of each component is carried out in (k+1) independent unit and the arithmetic is independent in the sense that no carries are transferred from one unit to the next.



The error detecting and correcting ability of the multi-residue code is related to an AN code in a well specified way. If we consider an AN code that corrects all the errors E in the set U(AM,d) where M is the least positive integer whose binary arithmetic weight is less than d, denoted by M(A,d) and let $A = LCM\ (m_1, m_2 \ldots m_k)$ then we can form a corresponding multiresidue

codeword with processor range $m_0 = AM$ with m_i being the check bases. There exist two categories of error: 1) errors in the processor, 2) errors in one or more of the checkers modulo m_i. It is assumed that errors in the processor and checker do not occur simultaneously. Now, if the check bases m_i are selected such that $|E|_{m_i} \neq 0$ for $E \varepsilon U(AM,d)$ for at least s check bases ($S \leq k$) then every error in the processor will result in a unique syndrome having s non-zero component with 1-1 correspondence. For instance, the codeword Z and syndrome S(Z) with an error in the processor are given by $Z = [|x+E|_{m_0}, x_1, x_2 \ldots x_k]$ and $S(Z) = [S_1 S_2 \ldots S_k]$ where $S_i = |x+E-x_1|_{m_i} = |E|_{m_i}$. An error in checker i results in the erroneous codeword $Z = [x, x_1 \ldots x_i + E_{m_i}, \ldots x_k]$ and $S(Z) = [0.0 \ldots x_i + E_{m_i} \ldots 0,0]$. If t checkers are in error there will be t nonzero components. Errors in (S-1) or fewer checkers are correctable.

Illustration

We have seen that the biresidue code $[N, |N|_7, |N|_{15}]$ can correct single errors in the processor. The $S(N) = (S_a, S_b)$ are distinct and nonzero corresponding to all single errors $\pm 2^j$ in the processor. This Biresidue code can be shown to have direct correspondence 1-1 with a single error correcting AN code.

$$A = LCM\ (7,15) = 105$$

$$105M(105,3) = 2^{12}-1$$

$U(2_{12}-1,1)$ = {1,2,4,8,16,32,64,128,256,512,1024,2048, 2047,3071,3583,3838,3967,4031,4063,4079,4087,4091,4093, 4094}.

Every $E \varepsilon U(AM,d)$ has a dinstinct syndrome $|E|_A$ given by the syndrome set {1,2,4,8,16,32,64,23,46,92,79,53,52,26,13,59,82, 41,73,89,97,101,103,104}. And there is 1-1 correspondence between $|E|_{105}$ and the syndrome pair $[|E|_7, |E|_{15}]$ listed in the following table.

Error pattern E	$\|E\|_{105}$	$\|E\|_7, \|E\|_{15}$
2^0	1	(1,1)
2^1	2	(2,2)
2^2	4	(4,4)
2^3	8	(1,8)
2^4	16	(2,1)
2^5	32	(4,2)
2^6	64	(1,4)
2^7	23	(2,8)
2^8	46	(4,1)
2^9	92	(1,2)
2^{10}	79	(2,4)
2^{11}	53	(4,8)
-2^0	104	(6,14)
-2^1	103	(5,13)
-2^2	101	(3,11)
-2^3	97	(6,7)
-2^4	89	(5,14)
-2^5	73	(3,13)
-2^6	41	(6,11)
-2^7	82	(5,7)
-2^8	59	(3,14)

Error pattern E	$\|E\|_{105}$	$\|E\|_7, \|E\|_{15}$
-2^9	13	(6,13)
-2^{10}	26	(5,11)
-2^{11}	52	(3,7)

The error control properties of the biresidue code and its corresponding AN code is as follows:

Codeword	$[N, N_7, N_{15}]$	105N
N = 44	[44,2,14]	4620
E = 8	[52,2,14]	4628
Syndrome	[1,8]	$\|4628\|_{105} = 8$

Any error in any one of the checkers will result in a syndrome $[|E|_7,0]$ of $[0,|E|_{15}]$ having one nonzero component and is therefore distinguishable and correctable.

Consequently, every multiresidue code capable of correcting t errors in the processor must be associated with a t error correcting AN code and vice versa. If there exists a t error correcting AN code we can always construct a multiresidue code of the same error correcting ability with suitable selection of check bases.

Example of multi-error correcting multiresidue code

We have seen that Mandelbaum Borrow code has multi-error correcting property, $A = \frac{2^{B-1}-1}{B}$ where B is a prime with 2 as a primitive element of the field of integers mod B. A distance 6 MB code is given by $A = \frac{2^{18}-1}{19} = (3^3)(7)(73)$ with range $m_0 =$

2^{18}-1. The code can correct $E\varepsilon U(m_0,2)$. From this we can construct a triresidue code having double error correction ability in the processor. The range of the processor is equal to $m_0 = 2^{18}-1$ and the check bases are suitably chosen to be $m_1 = 27x7$, $m_2 = 73x3$, $m_3 = 73x3$ such that $\alpha CM\ (m_1,m_2,m_3) = A = \frac{2^{18}-1}{19}$ and for each $E\varepsilon U(2^{18}-1,2)$ in the processor, the syndrome will have 3 nonzero components which is 1-1 correspondence with residue $|E|_A$. The code can therefore correct any error $E\varepsilon U(m_0,2)$ or any error in two or fewer checkers.

4.7 Summary

In this chapter we viewed error correcting codes used in arithmetic processors. The arithmetic codes were developed on similar lines as algebraic codes. The non-separate AN codes like the Brown Peterson code and the large distance Mandelbaum Barrow had single error and multiple error correcting ability but were undesirable from the practical point of view. However, they were shown to have a direct correspondence with separate residue codes which could be implemented easily. The increase in hardware cost due to the residue checkers was of the order 30-35 percent, proving more economical than the scheme of Triple Modular Redundancy.

CHAPTER 5. CONCLUSIONS

The survey covered a variety of error-correcting codes. Some of the codes were modifications of the existing communication codes, while others were specifically designed to meet the computer system requirement. The proofs of the mathematical theorems and derivations leading to the construction of the codes were replaced by working and illustrative examples to facilitate understanding. A general outline of the encoding and decoding scheme for the implementation of the code was presented with emphasis on simplicity, speed and efficiency. In terms of economics the error-correcting codes offered a viable alternative to other schemes for improving computer system reliability. Advances in integrated circuits or large scale integration (LSI) have accelerated the use of error coding techniques in digital systems. The survey presents a good reference for further work in this field.

REFERENCES

1. Peterson, W. W., and E. J. Weldon, "Error Correcting Codes," second edition, M.I.T. Press, Mass., 1972.

2. Lin, Shu, "An Introduction to Error Correcting Codes," Prentice-Hall, 1970.

3. Rao, T.R.N., "Error Correcting Codes for Arithmetic Processors," Academic Press, 1974.

4. Sellers, Hsiao, Brown, "Error Detecting Logic for Digital Computers," McGraw-Hill, 1968.

5. Hsiao, M.Y., and J. T. Tou, "Application of Error Correcting Codes in Computer Reliability Studies," IEEE Transaction on Reliability, Vol. R-18, No. 3, pp. 108-117, August 1969.

6. Wolf, J. K., M. L. Shooman and R. Boorstyn, "Algebraic Coding and Digital Redundancy," IEEE Transaction on Reliability, Vol. R-18, No. 3, pp. 91-107, August 1969.

7. Hsiao, M.Y., "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," IBM Journal of Research and Development, Vol. 14, No. 4, pp. 395-401, July 1970.

8. Hsiao, M.Y., and Bassen, "Double Error-Correcting BCH Code in Computer Memory Environment," Proceedings of the Mervin J. Kelly Communication Conference, University of Missouri, Rolla, Missouri, October 5-7, 1970.

9. Bossen, D.C., "b-Adjacent Error Correction," IBM Journal of Research and Development, Vol. 14, No. 4, pp. 402-408, July 1970.

10. Hong, S. J., and A. M. Patel, "A General Class of Maximal Codes for Computer Application," IEEE Transaction on Computers, Vol. C-21, No. 12, pp. 1322-1331, December 1972.

11. Hsiao, Bossen, Chein, "Orthogonal Latin Squares," IBM Journal of Research and Development, Vol. 14, No. 4, pp. 390-394, July 1970.

12. Brown, D.T., and F. F. Sellers, "Error Correction for IBM 800 bits per inch Magnetic Tape," IBM Journal of Research and Development, Vol. 14, No. 4, pp. 384-389, July 1970.

13. Swanson, Robert, "Understanding Cyclic Redundancy Codes," Computing Designs, Tucson, Arizona, pp. 93-99, November 1975.

14. Helness, Karl M., "Implementation of a Parallel Cyclic Redundancy Check Generator," Hewlett-Packard Company, Data System Division,Cupertino, California, pp. 91-95, March 1975.

15. Pandeya, A., and T. Cassa, "Parallel CRC Lets Many Lines Use One Circuit," Interdata Incorporated,Oceanport, New Jersey, pp. 87-91, September 1975.

16. Chien, R. T., "Memory Error Control - Beyond Parity," IEEE Spectrum, Vol. 10, No. 7, pp. 17-23, July 1973.

17. Hong, S. J., and A. M. Patel, "Optimal Rectangular Code for High Density Magnetic Tape," IBM Journal of Research and Development, No. 6, pp. 579-588, November 1974.

18. Malhotra, V.K., and R. D. Fisher, "A Double Error Correcting Scheme for Peripheral System," IEEE Transaction on Computers, Vol. C-25, No. 2, PP. 105-115, February 1976.

19. Patel, A.M., "Shift Register Implementation of b-Adjacent Codes," Technical Report 00-217 IBM Corporation, Poughkeepsie, New York, 1970.

20. Chien, R.T., "Burst Error Correcting Codes with High Speed Decoding," IEEE Transaction on Information Theory, Vol. IT-15, No. 1, pp. 109-113, January 1969.

21. Oldham, Chien, Tang, "Error Detection and Correction in a Photo Digital Storage System," IBM Journal of Research and Development, Vol. 12, No. 6, pp. 422-430, November, 1968.

22. Solomon and I.S. Reed, "Polynomial codes over certain finite fields," Journal of S.I.A.M., 8, 300 (1960).

23. Rao, R. N., "Error Checking Logic for Arithmetic Type Operation of a Processor," IEEE Transactions on Computers, Vol. C-17, No. 9, pp. 845-849, September 1968.

24. Rao, R.N., "Biresidue Error Correcting Codes for Computer Arithmetic," IEEE Transaction on Computers, Vol. C-19, No. 5, pp. 398-402, May 1970.

25. Rao, R.N., and Garcia, "Cyclic and Multiresidue Codes for Arithmetic Operations," IEEE Transaction on Information Theory, Vol. IT-17, No. 1, pp. 85-91, January 1971.

VITA

Mr. D. Kapur was born in Calcutta, India, on August 22, 1951, the son of Mr. and Mrs. B. K. Kapur. In June 1974 he received his B. Tech (Hons) in Electrical Engineering from Indian Institute of Technology, Kharagpur, India. Currently he is working towards an M.S. degree in Computer Science at Lehigh University in the Department of Electrical Engineering, and serves as a Teaching Assistant in the Center for Applied Mathematics at Lehigh University.