Arvind M. Patel

Error Recovery Scheme for the IBM 3850 Mass Storage System

The IBM 3850 Mass Storage System (MSS) stores digital data on flexible magnetic tape media; however, it is different in many respects from the conventional multitrack tape machines. In particular, the use of a single-element rotary read-write head imposes new demands in the areas of data encoding and error recovery. This paper presents a comprehensive scheme for error recovery for the 3850 MSS which features a new error-correction code in a serial, single-stripe data format. The recovery procedure is designed around resynchronizable sections of data which are rendered independent of each other in error modes through the use of zero-modulation encoding and self-contained error-detection pointers. These error-detection pointers and the resynchronization signals are utilized in conjunction with interleaved codewords of the error-correction code. The code is designed with a generating polynomial in which the roots are chosen from the set of elements of a 16-element subfield of the Galois field $GF(2^8)$. This choice provides the necessary code structure for desired code capabilities and facilitates fast decoding of errors with an economical implementation of the decoder. The scheme provides correction capabilities for various combinations of mixed-mode short and long errors common to magnetic tape recording of digital data.

Introduction

The IBM 3850 Mass Storage System (MSS) consists of an array of data cartridges about 1.9 in. (4.8 cm) in diameter and 3.5 in. (8.9 cm) long, with a capacity of 50 million characters each. Each cartridge contains magnetic tape 2.7 in. (6.9 cm) wide and 64 ft (19.5 m) long, on which data are organized in cylinders analogous to those of a disk file and can be transferred to the disk file one cylinder at a time. Up to 4720 cartridges are stored in hexagonal compartments in a honeycomb-like apparatus that includes mechanisms for fetching cartridges from the compartments, for the reading and writing of data on them, and for the replacement of cartridges in the compartments.

The data are recorded as coded binary sequences corresponding to the presence or absence of magnetic flux transitions in slanted, fixed-length stripes across the tape at a density of 67 stripes per inch (26.4 stripes per cm) and lineal density ranging from 3444 to 6888 flux transitions per inch (1356 to 2712 flux transitions per cm). A read-write operation always involves the processing of whole stripes, with each data stripe containing exactly 4096 net data bytes after decoding. If the staging adaptor is destaging more than 4096 bytes, it must format the data to fit within stripe boundaries. One destage order can transfer up to 61 stripes, the equivalent of one cylinder of a disk file.

Unlike the conventional fixed head in the multitrack tape machines, the 3850 uses a rotary read-write head. The high cost of a multielement head and the need for a multiple number of read-write electronic channels are eliminated by replacing parallel multitrack recording with serial single-stripe recording. The data are recorded in short slanted stripes across the tape instead of in long tracks along the tape. In this way the jittery motion of the flexible tape over a fixed head is replaced by a smooth, controlled motion of the rotary head over a steady-state tape. The tape follows a helical path around a mandrel and is stepped in position from one slanted stripe to the next over a circular slit in the mandrel which houses the continuously moving read-write transducer element of

the rotary head. These new approaches in transport design bring various improvements in the design of mechanical hardware, reducing head-tape wear and providing controlled spacing between the magnetic transducer and the recording media. However, they also demand new sophistications in data encoding techniques for purposes of waveform design and error recovery.

The read-write signal to the rotary transducer is coupled through a transformer whose primary and secondary coils are in continuous relative motion. This arrangement requires read-write signals with zero dc component. Previous papers [1, 2] reported a novel waveform design scheme referred to as "zero modulation" (ZM), which was created especially for the 3850 application. Zero modulation features waveforms with zero dc component and yet retains the advantages of other commonly known encoding methods, including high efficiency. Furthermore, the stringent coding constraints of ZM provide powerful checks on errors in decoding the read data. Zero modulation is used in the 3850 not only as a waveform encoding method but also as a powerful error-detection code.

This paper describes the design and implementation of the 3850 error recovery scheme which features a new error-correction code fitted into a serial, single-stripe data format. The second section begins with a discussion of the error recovery problem, which is followed by the rationale and approach for the design of the data format with resynchronizable sections, ZM encoding, and interleaved codewords of the error-correction code. In the third section, the design of the error-correction code is given, with a discussion of the salient characteristics and features of the code. This is followed by a description of the implementation of the encoding process, syndrome computation, and decoding process, providing correction of one or two erroneous bytes in a codeword.

A brief summary of zero modulation code is given in Appendix 1; proofs of theorems on the capability of the error-correction code are presented in Appendix 2; and Appendix 3 provides an example, with step-by-step results, of the encoding and decoding process of the error-correction codeword.

Error recovery scheme and data format

Errors in magnetic tape recording are primarily caused by defects on the magnetic media or variations in head-media separation in the presence of dust particles. These errors often affect as many as 100 bits at a time, depending on the density of recording. Furthermore, long errors are often associated with loss of synchronization of the read clock, which renders subsequent data unreadable. In the

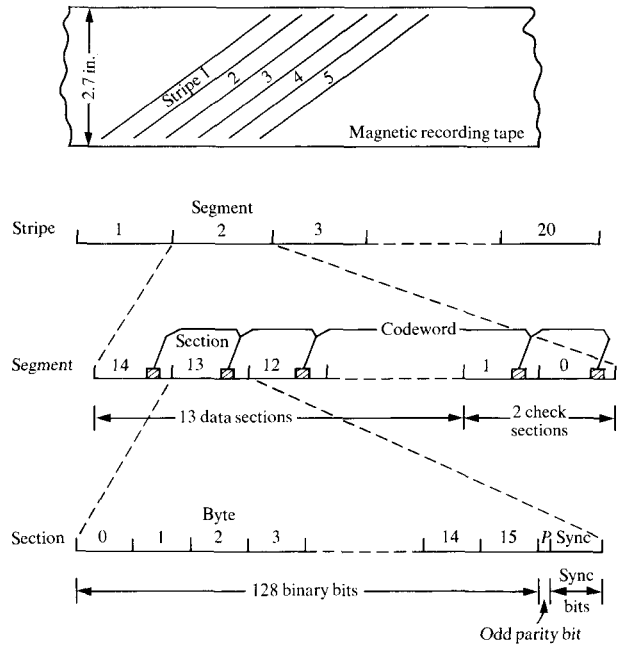

Figure 1 Stripe data format.

standard nine-track 800-bpi magnetic tape machines the errors are identified as track errors and are corrected by means of a specially designed error-correction system [3].

As the recording density is increased, another error mode plays an important role. This is the well-known bit-shift phenomenon where a magnetic flux transition is shifted from its normal position due to interference from neighboring flux transitions. The bit shift usually results in a two-bit error, where 01 is read in place of 10 or *vice versa*. The nine-track 6250-bpi tape machines feature an error-correction code [4] which corrects various combinations of one or two full tracks and multiple numbers of one-bit and two-bit errors.

The parallel multitrack data format is not available in the 3850 MSS. Instead, the data are organized in resynchronizable sections in order to facilitate recovery from mixed-mode errors in magnetic recording read-write processes involving one-bit errors caused by random noise, multiple two-bit errors caused by bit shift, and clusters of errors caused by defects and dust particles—including the capability to resynchronize the clock. The data format of the 3850 stripe is illustrated in Fig. 1. The stripe is divided into 20 segments. The segments are appended to each other, forming a continuous waveform; however, each segment is a separate entity and can be decoded without reference to the data in other segments. Each segment consists of 13 data sections followed by two check sections. In a write operation, the bit values for the two check sections are computed in accordance with an error-

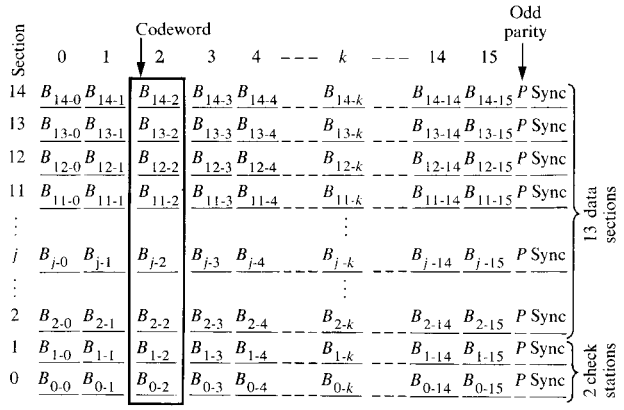

Figure 2 A segment: 15 sections formed with 16 interleaved codewords. (Sections are appended to each other to form a segment.)

correction code by processing the 13 data sections as they are being recorded. The computed check sections are then appended to the 13 data sections, thus completing a segment.

Each section is 129 bits long, consisting of 16 bytes of binary information with an overall odd-parity bit. This sequence of 129 bits is encoded into a 258-digit zero modulation waveform followed by a known unique synchronization signal (see Appendix 1 for details on ZM). The odd-parity bit serves the dual purpose of checking data errors and of limiting the memory requirement in the encoding process by setting the look-back parity of ZM to zero at the end of each section. The sections are appended to one another to form a continuous ZM waveform. Thus, each section is protected by the synchronization signal at both of its ends. This allows resynchronization of the decoding clock, at the beginning and at the end of each section, in the event of a long error causing loss of synchronization.

In a read operation, each section is read through the ZM decoding algorithm, which also checks for errors through stringent runlength and dc charge constraints. Error-free ZM patterns possess runlengths of at least one and at most three zeros between two ones and the dc charge value is always constrained within ± 3 units. Thus, two consecutive ones or four consecutive zeros indicate an error. Acquisition of dc charge in excess of ± 3 units can be detected with an up-down counter which increments for every digit position recorded with a positive level; decrements likewise when the level is negative; and signals an error if the total exceeds ± 3 at any time. The charge value must also be zero at the end of the section, excluding the synchronization pattern. These checks and the odd parity at the end of each section detect most errors, including the two-bit errors caused by bit shift and drop out, and synchronization errors caused by defects and dust particles. It should be noted that the decoding errors in ZM do not propagate and that the decoding process always terminates at a section boundary. Thus, the presence of an error is usually detected by the ZM error-detection circuits in the vicinity of the error within the same section. The resynchronization signal at the beginning and at the end of each section provides or confirms the proper phase of the ZM double-frequency clock, thereby rendering each section independent in error modes.

All detected errors are reported to the decoder of the error-correction code for error recovery. Errors in up to two full sections in a segment can be recovered by means of this error-correction code, including the longest error in a worst-case situation when the defect coincides with a section boundary and affects two adjacent sections. A wide variety of shorter multiple errors is also detected and corrected by the same error-correction code.

The data format for the error-correction code is designed around the resynchronizable sections. The 16 bytes in each section belong to 16 different codewords, as shown in Fig. 2. Each codeword consists of 15 bytes—one from each of the 15 sections in a segment. Let k be the index for the codewords and j be the index for the sections in a segment, where $0 \leq k \leq 15$ and $0 \leq j \leq 14$. Then B_{jk} denotes the byte in position k in section j. The group of bytes $B_{0k}, B_{1k}, B_{2k}, \cdots, B_{14k}$ are recorded in position k of the sections $0, 1, 2, \cdots, 14$, respectively, and form the kth codeword W_k. Thus, the 16 codewords are interleaved in the data format of 15 sections in a segment. This interleaving of the codewords facilitates correction of mixed-mode errors. When a defect or dust particle affects up to two full sections, the resultant error is recoverable by correcting the corresponding two bytes in each of the 16 codewords. On the other hand, many combinations of multiple one-bit and two-bit errors in a segment are also recoverable, since each codeword can detect and correct any one of its bytes. Any stripe with a defect length of more than 128 bits is demarked by the write operation. Every write operation is followed by a read-back check. Every demarked stripe is bypassed by the read operation.

Excluding the two error-checking sections in each segment, the 20 segments in a stripe provide a net recording space for a data stream of 4160 bytes (Fig. 3). The first two bytes in this data stream are reserved for stripe identification. This is followed by a block of 4096 bytes of data, 60 bytes of filler zeros, and two bytes of Cyclic Redundancy Check (CRC) code. The two-byte CRC code

provides an overall check for the data integrity of the stripe, including errors caused by malfunctions in the data flow hardware and error-correction process. The read operation is retried both with and without a change in the read amplifier gain setting when an uncorrectable error is encountered in any codeword or when a miscorrected error is detected by the CRC at the end of the stripe.

Aside from the main body of 20 segments, each stripe begins with one start-up section recorded with an all-ones signal and a special sync character. The leftover space at the end of the stripe is recorded with an all-zeros signal. The start-up section provides a known uniform signal for the purpose of priming the clock and the detection circuits, and the special sync character marks the beginning of the data. The all-zeros signal at the end ensures removal of any previously written data which may be present beyond the twentieth segment due to minor variations in the stripe length.

Error-correction code

As just described, the 3850 error recovery scheme is designed around the concept of resynchronizable sections. The basic building block of this scheme is a 15-byte codeword W, designated as

$$W = [B_0, B_1, B_2, \cdots, B_{14}].$$

For simplicity of notation, the word index k has been dropped from the notation for bytes; thus the byte B_{jk} is denoted simply by B_j. In this codeword, B_0 and B_1 are the check bytes, and the remaining 13 bytes are the data bytes. The coding rules are given in the form of the following matrix (modulo-2) equations:

$$B_0 \oplus B_1 \oplus B_2 \oplus \cdots \oplus B_{14} = 0, \tag{1}$$

$$B_0 \oplus \mathbf{T}^{\lambda} B_1 \oplus \mathbf{T}^{2\lambda} B_2 \oplus \cdots \oplus \mathbf{T}^{14\lambda} B_{14} = 0, \tag{2}$$

where

$\oplus$ signifies modulo-2 sum,

B_i is an eight-bit column vector, $i = 0, 1, \cdots, 14$,

λ is 68,

$\mathbf{T}^{i\lambda}$ denotes $\mathbf{T}$ multiplied by itself $i\lambda$ times, and

$\mathbf{T}$ is an 8×8 matrix given by

$$\mathbf{T} = \begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix}. \tag{3}$$

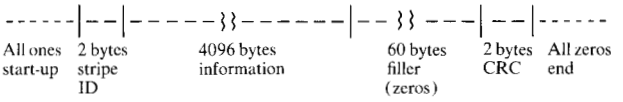

Figure 3 Data stream of 4160 bytes in a stripe (before encoding).

The code given by the coding rules (1) and (2) is derived from single-symbol-correcting Bose-Chaudhuri-Hocquenghem (BCH) codes [5–7] over symbols from the Galois field $GF(2^4)$. It is well known [4, 7–9] that the error-correcting codes for symbols from $GF(2^b)$ can be used for correction of groups of b adjacent errors. Here we present a modification to this concept where a code for symbols from $GF(2^b)$ is used for correction of $m \times b$ adjacent errors. The modification relates to the representation of the elements of the Galois field. We use elements of a 16-element subfield of $GF(2^8)$ in the parity check matrix in place of the conventional elements of $GF(2^4)$. The idea of substituting the subfield elements is a novel contribution in code design. It makes the code applicable to eight-bit binary bytes, provides the necessary code-structure for multiplexed code capabilities, and yet retains the same decoding steps as those for the code over $GF(2^4)$. The resultant code possesses the following two basic capabilities of two different well-known [9, 10] codes:

1. It detects all double bit-errors and corrects all single bit-errors.
2. It detects and corrects all single byte-errors.

This multiplexed capability makes the code applicable in two different ways. More importantly, it provides an effective method of reducing the probability of miscorrections in tape-like, mixed-mode error environment. In particular, the following two assertions are proved in Appendix 2:

1. If the code is used for correction of single byte-errors, then it will not miscorrect any combination of two one-bit errors.
2. If the code is used for correction of single bit-errors, then it will not miscorrect any combination of one byte-error with one bit-error in another byte.

In the 3850 MSS application, the code is used for correction of single byte-errors in the absence of error pointers. In this mode, the code exhibits a high level of protection against miscorrection of noise-induced bit-errors in more than one byte. In the presence of error pointers, the code corrects two erroneous bytes.

The matrix $\mathbf{T}$ represents a primitive element of $GF(2^8)$. Thus, the matrices $\mathbf{T}, \mathbf{T}^2, \mathbf{T}^3, \cdots, \mathbf{T}^{255}$ represent distinct

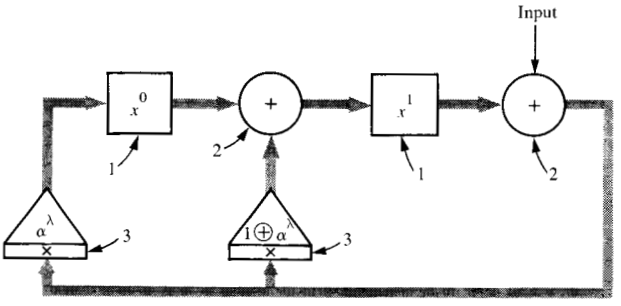

Figure 4 Block diagram of encoding network.
1: Storage of field element.
2: Addition of field elements.
3: Multiplication by a field element.

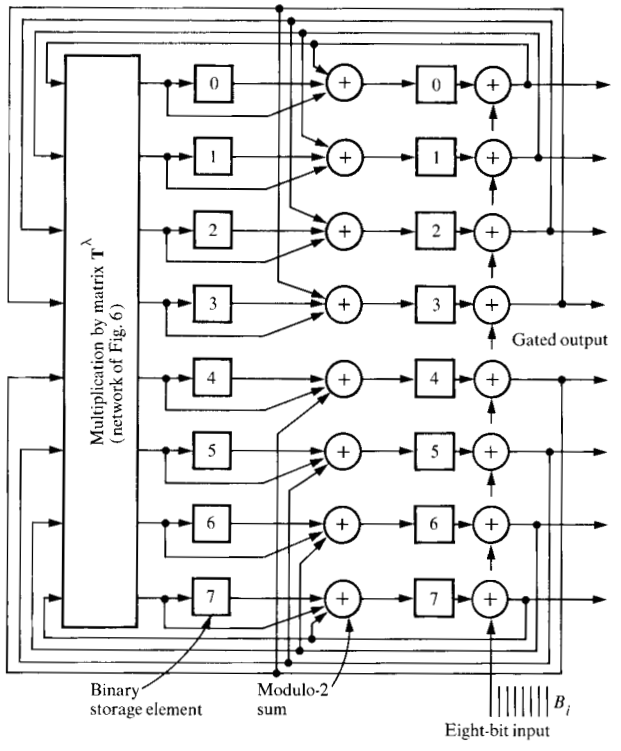

Figure 5 Encoding shift register.
All feedback connections gated open for output.
Generator polynomial: $g(x) = \mathbf{T}^\lambda \oplus (\mathbf{I} \oplus \mathbf{T}^\lambda)x \oplus x^2$.

non-zero elements of $GF(2^8)$. The matrix $\mathbf{T}^\lambda$ represents a primitive element of the 16-element subfield of $GF(2^8)$, where λ is a multiple of 17 and prime to 15. The choice of $\lambda = 68$ was made simply because it provided a minimum number of hardware connections in the implementation of multiplication by matrix $\mathbf{T}^\lambda$. The matrix $\mathbf{T}^{68}$ can be computed from $\mathbf{T}$ and is given by

$$\mathbf{T}^{68} = \begin{bmatrix} 0\,0\,0\,0\,1\,0\,0\,0 \\ 1\,0\,0\,0\,1\,1\,0\,0 \\ 0\,1\,0\,0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1\,0\,1\,1 \\ 1\,0\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,0\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1 \\ 0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix}.$$

The matrices $\mathbf{T}^\lambda$, $\mathbf{T}^{2\lambda}$, $\mathbf{T}^{3\lambda}$, $\cdots$, $\mathbf{T}^{15\lambda}$ represent distinct non-zero elements of the subfield, and $\mathbf{T}^{15\lambda} = \mathbf{I}$ where $\mathbf{I}$ is the 8×8 identity matrix. Thus the inverse $\mathbf{T}^{-i\lambda}$ of the matrix $\mathbf{T}^{i\lambda}$ is the matrix $\mathbf{T}^{(15-i)\lambda}$. The modulo-2 sum and product operations of these matrices are closed, in the sense that the result is always one of the elements of the subfield. It will be seen later that these properties of the subfield elements facilitate implementation of the decoder and save hardware and decoding time.

• *Encoding process*

The code described by Eqs. (1) and (2) is a cyclic code with a generator polynomial $g(x)$ with roots from the set of elements of a 16-element subfield of $GF(2^8)$:

$$g(x) = (1 \oplus x)(\alpha^\lambda \oplus x), \tag{4}$$

where α is a primitive element of $GF(2^8)$ and $\lambda = 68$. The roots 1 and α^λ are subfield elements corresponding to the matrix representations $\mathbf{I}$ and $\mathbf{T}^\lambda$, respectively. On multiplying out the factors of $g(x)$, we have

$$g(x) = \alpha^\lambda \oplus (1 \oplus \alpha^\lambda)x \oplus x^2. \tag{5}$$

The elements of $GF(2^8)$ can also be represented by the eight-digit binary bytes. For example, the first column of the matrix $\mathbf{T}^i$, for any i, is the commonly used eight-digit binary representation of the field element α^i. Then, any 15-byte codeword represents a polynomial over $GF(2^8)$ that is divisible by $g(x)$. Note that although the coefficients of $g(x)$ are restricted to the subfield elements, the coefficients of the codeword polynomial are from the complete set of elements of the field $GF(2^8)$. This allows all possible eight-bit patterns in the codeword and restricts the encoding and decoding operations within the subfield.

The encoding can be performed by a shift register network built for modulo $g(x)$ operations. Figure 4 shows a block diagram of this shift register, which can be constructed from the conventional binary network elements. The sum of any field elements β_1 and β_2, represented by eight-digit binary vectors B_1 and B_2, can be accomplished by the modulo-2 matrix sum of B_1 and B_2. The multiplication of any field element β by the elements α^λ and $1 \oplus \alpha^\lambda$ can be accomplished by the modulo-2 matrix multiplications $\mathbf{T}^\lambda B$ and $[\mathbf{I} \oplus \mathbf{T}^\lambda]B$, respectively, where the eight-digit binary vector B represents β. The resulting encoding

network is, in fact, an eight-channel binary shift register as shown in Fig. 5, in which each storage element of Fig. 4 is replaced by eight binary storage elements, and the modulo-2 matrix multiplication or addition is realized by means of a set of binary modulo-2 gates (XOR circuits). Figure 6 shows separately a network of modulo-2 gates for multiplication of any eight-bit vector with the matrix $\mathbf{T}^{\lambda}$.

The check bytes B_0 and B_1 are computed by processing the data bytes $B_2, B_3, B_4, \cdots, B_{14}$ in the encoding shift register of Fig. 5. Initially, the storage elements of the shift register are all set to zero. The ordered sequence of data bytes $B_{14}, B_{13}, B_{12}, \cdots, B_2$ are entered into the shift register in 13 successive shifts, as eight-bit-parallel vector inputs. At the end of this operation, the shift register contains the check bytes B_1 and B_0 in its high- and low-order positions, respectively. Then B_1 and B_0 are gated out without feedback and appended to the data bytes to form a 15-byte codeword. Appendix 3 presents an example where this encoding process and also the following decoding process, including syndrome computation and correction of one and two byte-errors, are illustrated in detail.

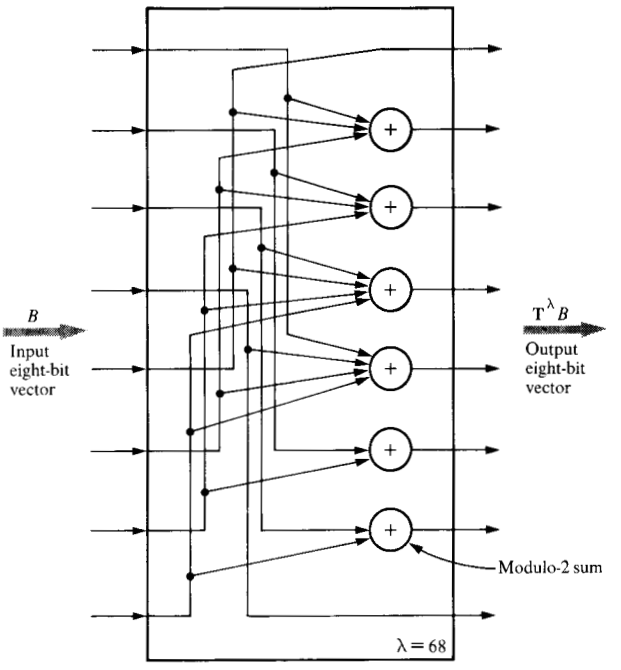

Figure 6 Network for multiplication by $\mathbf{T}^{\lambda}$.

Syndromes of error

The read data are checked for errors by means of the coding equations (1) and (2). All 16 codewords of a segment are stored in a temporary storage pending any correction of errors. The decoding process is carried out by applying the decoding algorithm to each of the 16 codewords independently. The algorithm will correct any one byte in an unknown position or any two bytes in indicated positions in each of the 16 codewords. Let $\hat{B}_0, \hat{B}_1, \hat{B}_2, \cdots, \hat{B}_{14}$ denote the read bytes corresponding to the written bytes $B_0, B_1, B_2, \cdots, B_{14}$, respectively. Let S_0 and S_1 denote the results of computations when the read byte values are substituted in place of the written byte values in the left-hand side of Eqs. (1) and (2). If the read codeword is error free, then S_0 and S_1 both will be zero, as seen from Eqs. (1) and (2); however, a non-zero value in S_0 or S_1 indicates that one or more read bytes are in error. The eight-bit vectors S_0 and S_1 are called "syndromes of error" and are given by

$$S_0 = \hat{B}_0 \oplus \hat{B}_1 \oplus \hat{B}_2 \oplus \cdots \oplus \hat{B}_{14} \tag{6}$$

and

$$S_1 = \hat{B}_0 \oplus \mathbf{T}\hat{B}_1 \oplus \mathbf{T}^{2\lambda}\hat{B}_2 \oplus \cdots \oplus \mathbf{T}^{14\lambda}\hat{B}_{14}. \tag{7}$$

The syndrome vectors S_0 and S_1 can be computed in a manner similar to the encoding process by means of the shift register of Fig. 5. Alternatively, we compute S_0 and S_1 by means of two separate eight-bit shift registers SR_0 and SR_1, respectively. These shift registers are shown in Fig. 7. The shifting operation in SR_0 and SR_1 corresponds

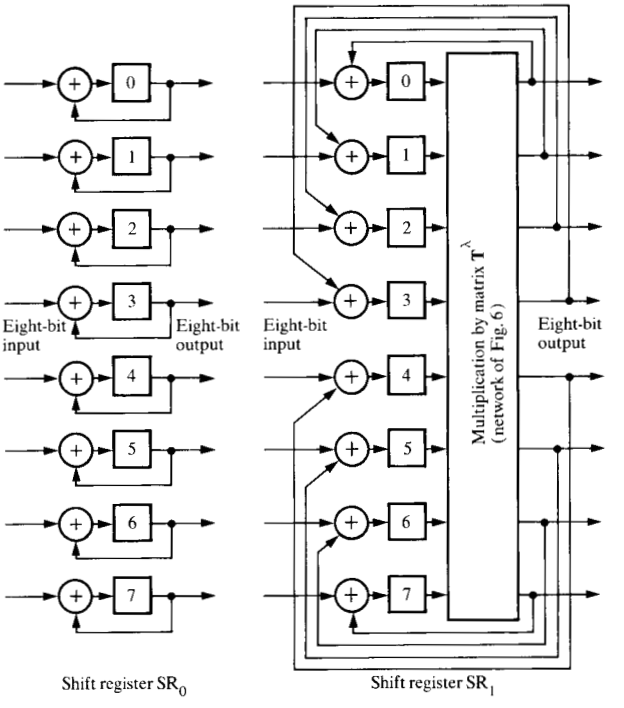

Figure 7 Decoding shift registers.
All feedback connections gated open for output.
Shifting operation in SR_0 is equivalent to multiplication by matrix $\mathbf{I}$.
Shifting operation in SR_1 is equivalent to multiplication by matrix $\mathbf{T}^{\lambda}$.

Table 1 Parameter p as a function of i.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table 2 Parameter q as a function of $(j - i)$.

$j - i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	*
q	3	6	11	12	5	7	2	9	13	10	1	14	8	4	0

*j is undefined and error pattern e_j is *zero*.

to multiplying the content vector by the matrix $\mathbf{I}$ and by the matrix $\mathbf{T}^{\lambda}$, respectively. Initially, the registers are set to contain zeros. The ordered sequence of read bytes $\hat{B}_{14}$, $\hat{B}_{13}$, $\hat{B}_{12}$, $\cdots$, $\hat{B}_1$, $\hat{B}_0$ is entered into both registers, SR_0 and SR_1, in 15 successive shifting operations, with $\hat{B}_{14}$ entering first. As a result, SR_0 contains the syndrome S_0, and SR_1 contains the syndrome S_1. The correction of errors is accomplished by further processing of these syndromes.

• *Correction of two bytes*

When the erroneous sections are indicated by ZM detection, this information is passed on to the decoder in the form of error pointers. Let i and j denote the position values of the two erroneous bytes in a codeword, where $i < j$. The symbols e_i and e_j are used to represent the unknown error patterns in bytes B_i and B_j, respectively, so that

$$\hat{B}_i = B_i \oplus e_i \tag{8}$$

and

$$\hat{B}_j = B_j \oplus e_j. \tag{9}$$

When the indices i and j are known, the unknown error patterns e_i and e_j can be determined by processing the syndromes S_0 and S_1, provided all other bytes are error free. The syndrome equations (6) and (7) can be reduced in terms of these unknown error patterns by combining with the coding equations (1) and (2), respectively. Thus we have

$$S_0 = e_i \oplus e_j \tag{10}$$

and

$$S_1 = \mathbf{T}^{i\lambda} e_i \oplus \mathbf{T}^{j\lambda} e_j. \tag{11}$$

Since i and j are known, the two simultaneous equations (10) and (11) may be solved for the two unknown variables e_i and e_j to obtain

$$e_j = [\mathbf{I} \oplus \mathbf{T}^{(j-i)\lambda}]^{-1}[S_0 \oplus \mathbf{T}^{-i\lambda} S_1] \tag{12}$$

and

$$e_i = S_0 \oplus e_j. \tag{13}$$

Equations (12) and (13) may be implemented with simple hardware. For this, the closure property and the multiplicative inverse of the subfield elements which were discussed before are used. In particular, we note that

$$\mathbf{T}^{p\lambda} = \mathbf{T}^{-i\lambda} \tag{14}$$

and

$$\mathbf{T}^{q\lambda} = [\mathbf{I} \oplus \mathbf{T}^{(j-i)\lambda}]^{-1}, \tag{15}$$

where p and q depend only on the known values of i and j. The parameters p and q are precalculated for all possible values of i and j and are given in Tables 1 and 2, respectively.

Thus, the decoding equations (12) and (13) can be rewritten into much simpler form as

$$e_j = \mathbf{T}^{q\lambda}[S_0 \oplus \mathbf{T}^{p\lambda} S_1] \tag{16}$$

and

$$e_i = S_0 \oplus e_j. \tag{17}$$

The decoder then consists of the following four simple steps:

Step 1: Multiply S_1 by the matrix $\mathbf{T}^{p\lambda}$.
Step 2: Add S_0 to the result of step 1.
Step 3: Multiply the result of step 2 by $\mathbf{T}^{q\lambda}$.
Step 4: Add S_0 to the result of step 3.

The multiplication by $\mathbf{T}^{p\lambda}$ and $\mathbf{T}^{q\lambda}$ of steps 1 and 3 can be performed by means of the shift register SR_1 of Fig. 7 with p and q shifting operations, respectively. The addition of S_0 of steps 2 and 4 can be accomplished by entering the vector S_0 into SR_1 at the time of the last shifting operation of the previous step. The results of steps 3 and 4 provide the correction patterns e_i and e_j for bytes $\hat{B}_i$ and $\hat{B}_j$, respectively. When only one byte is in error, as indicated by pointer i, and the second pointer value j is undefined, the syndrome processing still determines e_i and e_j in which e_j must result in a zero value. A non-zero value of e_j in this case indicates an uncorrectable error in one or more unknown byte positions.

• *Detection and correction of one byte*

Through violations of one or more ZM constraints, almost all errors are detected. However, if any error escapes this detection, the decoder may encounter a code-

word with non-zero syndromes and absence of error pointers. In this case the syndromes are processed for detection and correction of one-byte error. Here the decoder determines the index i of the erroneous byte and the corresponding error pattern e_i. When all other bytes are error free, the syndrome equations (6) and (7), in view of the coding equations (1) and (2), reduce to

$$S_0 = e_i \tag{18}$$

and

$$S_1 = \mathbf{T}^{i\lambda} e_i. \tag{19}$$

Thus, the error pattern e_i is determined by the syndrome S_0. Also, from Eqs. (18) and (19) we have

$$\mathbf{T}^{-i\lambda} S_1 = S_0; \tag{20}$$

that is,

$$\mathbf{T}^{(15-i)\lambda} S_1 = S_0. \tag{21}$$

Once again, using the shift register SR_1 of Fig. 7, the index i can be determined. With S_1 as the initial content, SR_1 is shifted and its contents are compared with S_0 while counting down from 15 with each shift. When the contents do compare with S_0, the count indicates the index i of the erroneous byte. If the contents do not compare with S_0 even when the counter reaches zero, then this indicates the presence of two or more erroneous bytes which cannot be corrected without ZM (or other) pointers. The occurrence of two erroneous bytes in one codeword without ZM pointers is not very likely.

Summary and comments

The error recovery scheme for the IBM 3850 MSS is designed to deal with mixed-mode errors comprised of single-bit errors caused by random noise, two-bit errors caused by the bit-shift phenomenon, and the long-burst errors caused by media defects, dirt particles, etc. The loss of clock synchronization over a media defect was an added problem, and the transformer coupling of the rotary transducer was a given constraint.

In this paper, the error recovery scheme as it is implemented in the IBM 3850 system has been described. It features a new error-correction code, with dual capability, designed with a polynomial whose roots are chosen from the set of a 16-element subfield of $GF(2^8)$. This code not only possesses the necessary code capability for a mixed-mode error environment, but also provides the structure for economy in decoder hardware. The concept of multiplexed code capability can be extended for applications with other special error environments.

Another feature of this error recovery scheme is the zero modulation encoding and its dual-function role. Zero modulation ensures the absence of dc component in the recording signal passing through the transformer coupling and also provides very reliable error-detection pointers in the read process. The error recovery scheme is designed around a data format consisting of resynchronizable sections which are rendered independent in error modes through the use of zero modulation encoding and self-contained error-detection pointers. A segment of 15 resynchronizable sections is formed by interleaving 16 codewords. Each 15-byte codeword provides correction of one unknown or two known erroneous bytes. As a result, the scheme provides correction of various combinations of mixed-mode short and long errors—ranging from a multiple number of one-bit and two-bit errors to one or two full sections in each segment.

Appendix 1: Zero modulation

The ZM algorithm maps every data bit sequentially into two binary digits. The mapping is described in terms of a data bit to be encoded, one preceding data bit, and the two coded digits corresponding to the preceding data bit; and in terms of two parity functions that look ahead and back relative to the bit being encoded. Look-ahead parity $P(A)$ is the count, modulo 2, of *ones* in the data stream, beginning with the data bit being encoded and counting forward to the next *zero* bit; look-back parity $P(B)$ is the count, modulo 2, of all *zeros* in the data stream from its beginning up to the present bit. For example, in the data sequence 01011110, with bit positions considered from left to right, $P(A) = 1$ at the second, fifth, and seventh bits and $P(B) = 1$ at the first, second, and eighth bits.

The encoding and decoding rules are expressed in the form of binary logic functions. The encoding function is

$$a_0 = \bar{d}_0 \bar{d}_{-1} + d_0 \bar{d}_{-1} \overline{P(A)} P(B) + d_{-1} \bar{a}_{-1} \bar{b}_{-1},$$
$$b_0 = d_0 [P(A) \bar{d}_{-1} + \overline{P(B)} + b_{-1}];$$

and the decoding function is

$$d_0 = b_0 + a_0 \bar{a}_1 \bar{b}_1 + a_0 a_{-1} \bar{b}_{-1},$$

where the symbol d represents a data bit; a and b represent coded digits; and subscripts -1, 0, and 1 signify preceding, current, and succeeding bits, respectively. For convenience, the nonexistent bit preceding the first data bit is assumed to be *one* and its look-back parity is *zero*; the nonexistent bit following the last bit is *zero*.

Look-back parity $P(B)$ can be obtained by updating a *one-bit* storage cell for every *zero* in the data as data bits are encoded. Look-ahead parity $P(A)$ depends on the length of a string of *ones* in the succeeding data sequence. When the algorithm imposes no limit on the length of this string, the computation of $P(A)$ requires an encoder with

Table A1 Computation of check bytes B_0 and B_1.

	Write data input	*Shift count*	*Contents low-order byte*	*Contents high-order byte*
B_{14}	10010111	1	00000111	10010000
B_{13}	11101000	2	11101111	10010000
B_{12}	10101010	3	11111111	00101010
B_{11}	11111000	4	01010001	01111100
B_{10}	11011001	5	00101000	11011100
B_9	10010001	6	10000110	11100011
B_8	00010101	7	00101011	01011011
B_7	01111111	8	01111010	01110101
B_6	00000000	9	01001101	01000010
B_5	00100111	10	01000100	01101100
B_4	10000001	11	11011100	01110101
B_3	01010101	12	00010010	11101110
B_2	10111110	13	00101101 = B_0	01101111 = B_1

Table A2 Syndrome computation (one byte in error).

	Read data input	*Shift count*	*Contents of SR_0*	*Contents of SR_1*
$\hat{B}_{14}$	10010111	1	10010111	10010111
$\hat{B}_{13}$	11101000	2	01111111	11101111
$\hat{B}_{12}$	10101010	3	11010101	01000010
$\hat{B}_{11}$	11111000	4	00101101	11101000
$\hat{B}_{10}$	01110011	5	01011110	11011101
$\hat{B}_9$	10010001	6	11001111	01010110
$\hat{B}_8$	00010101	7	11011010	01100100
$\hat{B}_7$	01111111	8	10100101	00100001
$\hat{B}_6$	00000000	9	10100101	00001000
$\hat{B}_5$	00100111	10	10000010	11110111
$\hat{B}_4$	10000001	11	00000011	10110000
$\hat{B}_3$	01010101	12	01010110	00000110
$\hat{B}_2$	10111110	13	11101000	11100010
$\hat{B}_1$	01101111	14	10000111	00100101
$\hat{B}_0$	00101101	15	10101010 = S_0	01001101 = S_1

infinite memory. In order to limit the amount of memory, a parity is inserted at the end of every section of f data bits, which makes $P(B)$ equal to *zero* at position $f + 1$ at the end of each section. When $P(B)$ is *zero*, the encoding functions no longer depend on $P(A)$. Thus, $P(A)$ has no effect on ZM mapping at the boundary of every section of $f + 1$ bits in the data sequence with parity bits, and the computation of $P(A)$ at any data bit need not extend farther than f bits. Then $P(A)$ is given by the binary logic function of the data stored in f bits of memory:

$$P(A) = d_0\bar{d}_1 + d_0d_2\bar{d}_3 + d_0d_2d_4\bar{d}_5 + \cdots + d_0d_2d_4 \cdots d_{t-4}\bar{d}_{t-3} + d_0d_2d_4 \cdots d_{t-4}d_{t-2},$$

where $t = f$ if f is even and $t = f - 1$ if f is odd.

The encoding process is delayed by f bit periods in a continuous stream of data for computing $P(A)$, but the decoding process is delayed by only one bit period. The decoding errors in ZM do not propagate, and the decoding process always terminates at the section boundary.

In the coded ZM binary sequence, any two consecutive *ones* are separated by at least one and at the most three *zeros*. This sequence is converted into a waveform using a transition for a *one* and no transition for a *zero* in the binary sequence. Consequently, the narrowest pulse in the ZM waveform spans two digits in the coded sequence, thus keeping the ratio of the data density to the highest recorded transition density close to *one*. Similarly, the widest pulse spans four digits, thus limiting the range of different pulse widths. The ZM waveform has zero dc component, the accumulated dc charge being the difference between the numbers of positive and negative pulses from the beginning to any digit position in the waveform. In the ZM waveform, the accumulated dc charge value always remains within ±3 units, and it always returns to a zero value at the section boundary.

In the IBM 3850 MSS the value of f is 128. A known unique pattern is inserted at the end of each section and used as a synchronization pattern. This pattern is

S = 01000100101000101000100010100 1.

The waveform corresponding to S satisfies the ZM pulse-width constraints. It has zero dc component, but does not satisfy the ±3 charge constraint. The pattern S contains the sequence 00101000101000, which is the shortest among those that never occur in the valid ZM pattern in its original or shifted position. Thus, when the synchronization is lost, the sequence S can be still identified from the shifted data, which then re-establishes the synchronization.

The reverse of the waveform corresponding to S also makes a good sync mark. The IBM 3850 MSS uses this distinction to mark the beginning of the data in a segment by means of the reverse sync waveform in contrast with the regular sync waveform at the end of each section.

One interesting property of the ZM coded waveform that is not used in 3850 MSS and not reported in previous papers [1, 2] is read-backward symmetry. A properly terminated ZM waveform, when read backward, is a ZM waveform corresponding to the same data in reverse. In particular, when a parity bit is appended to the data to make $P(B)$ equal to *zero* at the end, the encoding process terminates the dc charge value at zero at the end of the corresponding waveform. It can be shown that such a waveform, which has zero dc component, can be decoded forward or backward by means of the ZM decoding algorithm.

Appendix 2: Theorems on code capability

Theorem 1 When the code given by coding equations (1) and (2) is used for correction of single byte-errors, it will not miscorrect any combination of two one-bit errors.

Proof The syndrome expressions for two one-bit errors are

$$S_0 = e_1 \oplus e_2 \tag{A1}$$

and

$$S_1 = \mathbf{T}^{x\lambda} e_1 \oplus \mathbf{T}^{y\lambda} e_2 \qquad x \neq y, \tag{A2}$$

where e_1 and e_2 are error patterns with weight one, and x and y denote the locations of bytes in error.

The syndrome expressions for single byte-error are

$$S_0 = e \tag{A3}$$

and

$$S_1 = \mathbf{T}^{z\lambda} e, \tag{A4}$$

where e is the error pattern and z denotes the location of the erroneous byte.

A miscorrection of two one-bit errors as one byte-error implies that $e = e_1 \oplus e_2$ and

$$\mathbf{T}^{z\lambda}(e_1 \oplus e_2) = \mathbf{T}^{x\lambda} e_1 \oplus \mathbf{T}^{y\lambda} e_2. \tag{A5}$$

Equation (A5) can be rewritten as

$$[\mathbf{T}^{z\lambda} \oplus \mathbf{T}^{y\lambda}]^{-1}[\mathbf{T}^{z\lambda} \oplus \mathbf{T}^{x\lambda}] e_1 = e_2. \tag{A6}$$

Using the closure property of the subfield elements, we can write Eq. (A6) as

$$\mathbf{T}^{m\lambda} e_1 = e_2 \qquad \text{for some } m \neq 0. \tag{A7}$$

From the properties of algebraic elements represented by e_1, e_2, and $\mathbf{T}^{m\lambda}$, it can be shown that Eq. (A7) presents a contradiction. This proves the theorem.

Theorem 2 When the code given by coding equations (1) and (2) is used for correction of single bit-errors, it will not miscorrect any combination of one byte-error with one bit-error in another byte.

Proof The syndrome expressions for a combination of one byte-error with one one-bit error are

$$S_0 = e \oplus e_1 \tag{A8}$$

and

$$S_1 = \mathbf{T}^{z\lambda} e \oplus \mathbf{T}^{x\lambda} e_1, \tag{A9}$$

where e and e_1 are error patterns for a byte-error and a one-bit error, respectively, and z and x are corresponding error locations. The syndrome expressions for a single bit-error are

$$S_0 = e_2 \tag{A10}$$

and

$$S_1 = \mathbf{T}^{y\lambda} e_2, \tag{A11}$$

where e_2 is an error pattern with weight one and y is the error location.

A miscorrection of one as the other implies that $e = e_1 \oplus e_2$ and

$$\mathbf{T}^{z\lambda}(e_1 \oplus e_2) = \mathbf{T}^{x\lambda} e_1 \oplus \mathbf{T}^{y\lambda} e_2. \tag{A12}$$

As seen in Theorem 1, Eq. (A12) leads to a contradiction except when $x = y$ and $e_1 = e_2$. Thus miscorrection is not possible.

Table A3 Syndrome processing for one erroneous byte.

Shift count	*Contents of SR_0*	*Contents of SR_1*	*Equal?*
15	10101010	01001101	No
14	10101010	10000110	No
13	10101010	00010100	No
12	10101010	01100001	No
11	10101010	00101100	No
10	10101010	10101010	Yes

Erroneous byte position = 10

$\hat{B}_{10}$ = 01110011
Error = 10101010 (contents of SR_0)
Corrected B_{10} = 11011001

Appendix 3: Example

• *Encoding process*
The check bytes B_0 and B_1 are computed by shifting the data bytes $B_{14}, B_{13}, \cdots, B_2$ into the encoding shift register of Fig. 5. Table A1 presents the contents of this shift register after each shift.

• *Decoding process (one unknown byte in error)*
The syndromes S_0 and S_1 are computed by processing the read data bytes $\hat{B}_{14}, \hat{B}_{13}, \cdots, \hat{B}_2, \hat{B}_1, \hat{B}_0$ into shift registers SR_0 and SR_1 of Fig. 7. Table A2 presents the contents of these shift registers after each shift. The error in one byte is detected and corrected by further processing the computed syndromes in the same shift registers. Table A3 presents the steps of this decoding process.

• *Decoding process (two known bytes in error)*
Table A4 presents the steps of the syndrome computation process. The erroneous bytes are $\hat{B}_4$ and $\hat{B}_{10}$. Thus, $i = 4$

Table A4 Syndrome computation (two known bytes in error).

	Read data input	*Shift count*	*Contents of* SR_0	*Contents of* SR_1
$\hat{B}_{14}$	10010111	1	10010111	10010111
$\hat{B}_{13}$	11101000	2	01111111	11101111
$\hat{B}_{12}$	10101010	3	11010101	01000010
$\hat{B}_{11}$	11111000	4	00101101	11101000
$\hat{B}_{10}$	01110011	5	01011110	11011101
$\hat{B}_{9}$	10010001	6	11001111	01010110
$\hat{B}_{8}$	00010101	7	11011010	01100100
$\hat{B}_{7}$	01111111	8	10100101	00100001
$\hat{B}_{6}$	00000000	9	10100101	00001000
$\hat{B}_{5}$	00100111	10	10000010	11110111
$\hat{B}_{4}$	00000111	11	10000101	00110110
$\hat{B}_{3}$	01010101	12	11010000	00010010
$\hat{B}_{2}$	10111110	13	01101110	10000011
$\hat{B}_{1}$	01101111	14	00000001	00001001
$\hat{B}_{0}$	00101101	15	$00101100 = S_0$	$11100111 = S_1$

Table A5 Syndrome processing for two erroneous bytes.

Step	*Operation*	*Contents of* SR_1
1	Shift SR_1 $p = 11$ times	10111110
2	Add contents of SR_0 to SR_1	10010010
3	Shift SR_1 $q = 7$ times	$10101010 = e_j$
4	Add contents of SR_0 to SR_1	$10000110 = e_i$
	$\hat{B}_i = 00000111$	$\hat{B}_j = 01110011$
	$e_i = 10000110$	$e_j = 10101010$
	Corrected $B_i = 10000001$	Corrected $B_j = 11011001$

and $j = 10$. From Table 1 we find $p = 11$ corresponding to $i = 4$. From Table 2 we find $q = 7$ corresponding to $(j - i) = 6$. Table A5 presents the steps in the syndrome decoding process for correction of the erroneous bytes.

Acknowledgments

The author gratefully acknowledges the interest and cooperation of the various members of the IBM Poughkeepsie Laboratory, particularly the support and encouragement provided by J. A. Haddad, during the design and development phase of the work reported in this paper. Thanks and appreciation are also due J. T. Smith and R. B. Humphrey for creating an environment for quick acceptance of new ideas relative to the 3850 MSS project at the IBM Boulder Laboratory.

References

1. A. M. Patel, "Zero Modulation Encoding in Magnetic Recording," *IBM J. Res. Develop.* **19,** 366-378 (1975).
2. A. M. Patel, "New Method for Magnetic Encoding Combines Advantages of Older Techniques," *Computer Design* **15,** 85-91 (1976).
3. D. T. Brown and F. F. Sellers, Jr., "Error Correction for IBM 800-bit-per-inch Magnetic Tape," *IBM J. Res. Develop.* **14,** 384-389 (1970).
4. A. M. Patel and S. J. Hong, "Optimal Rectangular Code for High Density Magnetic Tapes," *IBM J. Res. Develop.* **18,** 579-588 (1974).
5. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Info. Control* **3,** 68-79 (1960).
6. D. C. Gorenstein and N. Zeirler, "A Class of Error Correcting Codes in p^m Symbols," *J. Soc. Indust. App. Math.* **9,** 207-214 (1961).
7. W. W. Peterson, *Error Correcting Codes*, M.I.T. Press, Cambridge, MA, 1961, pp. 162.
8. D. C. Bossen, "b-Adjacent Error Correction," *IBM J. Res. Develop.* **14,** 402-408 (1970).
9. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Indust. Appl. Math.* **8,** 300-304 (1960).
10. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.* **29,** 147-160 (1950).

Received March 13, 1979; revised June 13, 1979

The author is located at the IBM General Products Division laboratory, 5600 Cottle Road, San Jose, California 95193.