Two-level coding for error control in magnetic disk storage products

by Arvind M. Patel

Error-control coding has played a significant role in the design and development of magnetic recording storage products. The trend toward higher densities and data rates presents continuing demands for an ability to operate at a lower signal-to-noise ratio and to tolerate an increased number of correctable errors. Heretofore, the magnetic disk storage products used coding schemes that provided correction of one burst of errors in a record of length ranging from a few bytes of data to a full track on the disk. In this paper, we present a new coding architecture that facilitates correction of multiple-burst errors in each record in a typical disk storage application. This architecture embodies a two-level coding scheme which offers high coding efficiency along with a fast decoding strategy that closely matches the requirements of on-line correction of multiple bursts of errors. The first level has a smaller block delay and provides very fast correction of most of the errors commonly encountered in an average disk file. The second level, on a larger block size, provides reserve capability for correcting additional errors which may be encountered in a device with symptoms of a weaker component or an oncoming failure. The new IBM 3380J and 3380K disk files use a two-level scheme that is designed around the coding structure of the extended Reed–Solomon code. This design and the related encoding and decoding methods and implementation are presented in detail.

Introduction

New disk storage products are required to provide better reliability and availability in spite of their higher packing density and delivery rate with a large amount of on-line data. A relatively greater number of soft errors, including multiple-burst errors, must be processed very rapidly. Conventional coding techniques, such as multiple-symbol-correcting Reed–Solomon [1] or BCH [2] codes are very efficient in terms of redundancy. However, the algebraic constraints on their block size and the relative time required for the decoding of multiple-byte errors in each block are major restrictions of their applicability to long records of data in computer products.

Previously, disk products used error-correction schemes that corrected one burst of errors in a variable-length block of up to one full track. In these schemes, decoding for an error resulted in missing a disk revolution; and the error was usually corrected with data in the main memory, requiring CPU involvement in the error-recovery procedure. The occurrence of a multiple-burst error required rereading of the data several times in order to eliminate the presence of multiple errors.

In this paper, we present a new coding architecture that facilitates correction of multiple bursts of errors in each record in a typical disk storage application. This architecture embodies a two-level coding arrangement which offers high coding efficiency with a fast decoding strategy that closely matches the requirements of on-line correction of multiple-burst errors. The product can make use of this capability for improved reliability as well as lower product cost through (1) increase in recording density, (2) trade-off in head and disk tolerances, and (3) partial coverage of small disk defects, particularly those which are difficult to find in costly surface analysis testing.

The first-level capability in the two-level coding arrangement is designed for a specific reliability performance, so that an average disk file will not be required to utilize the second-level capability in its routine daily operations. The second-level capability is the "reserve" capability, which is frequently invoked in the case of a "weaker" device or a "failing" device, in which the soft-error rate is substantially above the average. Errors corrected through the second-level capability may be logged and this information may be used for flagging a "weaker" or "failing" device before it causes data loss through an uncorrectable error or a failure, thus reducing unscheduled service calls.

Two-level coding scheme

The data format in the two-level coding scheme [3] is designed around a two-level architecture consisting of subblocks within a block. An example with a figure appears in a later section. Each subblock consists of m data bytes and r_1 check bytes. The subblock is a code word from a code with minimum Hamming distance of d_1 symbol units, and the symbols (bytes) are elements of the Galois field $[GF(2^b)]$. Each block consists of n subblocks and r_2 check bytes which are shared by all its subblocks. The data part of the block-level code is viewed as modulo-2 superposition of n subblock code words. The r_2 check bytes (either independently or along with the superpositioned r_1 check bytes of all subblocks) provide a minimum Hamming distance of d_2 (over one subblock) at the block level where $d_2 > d_1$. The code words of both levels may be interleaved in order to provide correction for burst errors or clustered multi-symbol errors.

The decoding process provides correction of up to t_1 errors and detection of up to $t_1 + c$ errors in each subblock, where $d_1 = (2t_1 + c + 1)$. If the number of errors in a subblock exceeds the error-correcting capability at the first level, such errors are either left uncorrected or are miscorrected. If all errors are confined to one subblock, the block-level code will provide correction of up to t_2 errors, where $d_2 \geq (2t_2 + 1)$. However, many combinations of errors in multiple subblocks, including t_2 errors not confined to one subblock, are also correctable, as claimed in the following theorem.

Theorem 1 Let d_1 and d_2 denote the minimum Hamming distance in the subblock-level code and block-level code, respectively, where $d_1 = (2t_1 + c + 1)$ and $d_2 \geq (2t_2 + 1)$. This two-level coding scheme provides correction of t_2 errors anywhere in the block, provided $t_2 \leq 2t_1 + c$. In particular, it provides correction of a errors in the subblock with the most errors, up to b errors in each of the other subblocks, and y errors in the block-check bytes, where a, b, and y are any nonnegative integers such that $(a + y) \leq t_2$, $b \leq t_1$, and $(a + b) \leq 2t_1 + c$.

The set of parameters t_2, t_1, and c defines a two-level scheme. Let (a, b) denote an error combination consisting of a errors in one subblock and up to b errors in each of the other subblocks in a block where $a \geq b$. The errors in the block-level check bytes may be considered as part of a errors in the subblock that is corrected at the second level. Then Theorem 1 defines the (a, b) combinations corrected at each level. The (a, b) combinations corrected at the first level are defined by all values $a \leq t_1$ with $b = a$. The (a, b) combinations corrected at the second level are defined by all values a such that $t_1 < a \leq t_1 + c$ with $b = t_1$, and $t_1 + c < a \leq t_2$ with $b = 2t_1 + c - a$.

In the case when $c \geq 2$, the first-level corrections could be increased beyond t_1 errors to include (a, b) combinations defined by $t_1 < a \leq t_1 + c/2$ with $b = a$. These additional corrections may be processed at the second level using the first-level syndromes of all uncorrected subblocks.

Before presenting the proof of the theorem, we present a simple decoding strategy for the two-level coding scheme, which provides correction of all error combinations specified in the theorem. It is assumed that syndromes at each level are decoded for (up to) a fixed number of errors using minimum-distance decoding which finds a code word that differs from the received word in the fewest possible positions. The overall decoding process for the two-level code then follows the steps given below:

1. Compute and decode the syndromes for each subblock for up to t_1 errors. If a subblock is uncorrectable or if the number of decoded errors is larger than that in any of the previously decoded subblocks, then update the contents of a buffer memory. (The buffer holds the error information and the subblock number for a subblock which is either uncorrectable or requires correction of the greatest number of errors.)
2. Compute the block-level syndrome set $\{S\}$ from the received data, including all subblock-level error corrections. If $\{S\} \neq 0$ or if a subblock has uncorrectable errors at the first level, then second-level processing is required. The subblock f is identified by the information in the buffer memory as an uncorrected subblock or a subblock with the greatest number of corrected symbols.
3. Modify the block-level syndrome set $\{S\}$ as follows:

$$\{S'\} \leftarrow \{S\} - \{S_f\},$$

where $\{S_f\}$ is the contribution to the block-level syndromes due to the error corrections in the subblock f. If subblock f was left uncorrected in Step 1, then $\{S_f\} = 0$. The error information for computation of $\{S_f\}$ is available in the buffer memory.

4. Use the modified syndrome set $\{S'\}$ of Step 3 and decode for up to t_2 errors in subblock f, including the block-level check bytes. If the errors are found uncorrectable, then there are too many errors.
5. Remove the subblock-level corrections, if any, in subblock f. Enter the block-level corrections in subblock f and the block-level check bytes.

Step 2 of the above decoding procedure detects and identifies the subblock with more than t_1 errors uniquely. This is the key to the proof of the theorem.

Proof of Theorem 1 We assume that the subblock with the most errors ($i = f$) contains up to $(t_1 + c + x)$ errors; each of the other subblocks ($i \neq f$) contains up to $(t_1 - x)$ errors, and block check bytes contain y errors. The subblock number f and the nonnegative integers x and y are not known.

When $x = 0$, all errors at the subblock level are corrected or detected properly, since they are all within the capability of the subblock code. If a subblock f has more than t_1 errors, it will be identified by the first-level decoder as an uncorrectable subblock. When $x > 0$, consider the subblock with $(t_1 + c + x)$ errors. Since the minimum Hamming distance d_1 is $(2t_1 + c + 1)$, the nearest code word must differ from the received word in at least $(2t_1 + c + 1) - (t_1 + c + x)$ positions. Thus, the first-level decoder may miscorrect the subblock f by introducing additional errors in at least $(t_1 - x + 1)$ positions and as many as t_1 positions. Alternatively, the decoder may find subblock f uncorrectable if its Hamming distance from the nearest code word is greater than t_1. In contrast, the first-level decoder will correct $(t_1 - x)$ or fewer positions in each of the other subblocks, all of which will be corrected properly.

At the block level, the syndrome set $\{S\}$ is computed from the received data, which include all subblock-level error corrections. The set $\{S\}$ is, then, the block-level syndrome set for all errors still present in the block, that is, in subblock f and the block-level check bytes. These are at most $(t_2 + t_1)$ errors, which include $(t_1 + c + x) + y$ original errors and up to t_1 miscorrections. Since the minimum Hamming distance d_2 exceeds $(t_2 + t_1)$, the corresponding syndromes $\{S\}$ must be nonzero. Thus, the uncorrected or miscorrected errors in the block will be detected at the block level in Step 2 of the decoding procedure. The subblock f, with $(t_1 + c + x)$ errors, is uniquely identified in Step 2 from the fact that (it was declared uncorrectable or) the number of corrections in subblock f exceeds those in any of the other subblocks by at least 1. If the uncorrected errors are the y errors in block check bytes only, the value of f is of no consequence at the block level.

Let $\{S_f\}$ denote the block-level syndromes due to the error patterns introduced by the decoder in subblock f. We can remove these error patterns from subblock f and their contribution $\{S_f\}$ from the syndromes $\{S\}$. Thus, the set $\{S\} - \{S_f\}$ represents the syndromes for the original $(t_1 + c + x)$ errors in subblock f and y errors in the block-level check bytes. The syndromes $\{S\}$ and $\{S_f\}$ are all known quantities. The block-level decoding of $\{S\} - \{S_f\}$ for t_2 errors will, then, provide proper correction of all remaining errors. This completes the proof of the theorem.

From a practical viewpoint, a given decoding process can be considered on the fly if it meets the following test: The error correction on the previously received subblock is completed before the last byte of the next subblock is received. The processing will require a one-subblock delay [4]. Thus, on-the-fly decoding is possible if and only if the subblock decoding time is smaller than or equal to the subblock read time. The criterion for on-the-fly decoding can be given as

$$\frac{\textit{subblock decoding time}}{\textit{subblock size}} \times \textit{data rate} < 1.$$

Another important consideration is the above-cited one-subblock delay in decoding for errors. This delay directly affects the most important performance parameter of the disk file—namely, the access time. Thus, it is desirable to partition the block into subblocks with corresponding short subblock delay.

The two-level scheme provides a fast decoding strategy that closely matches the requirements of on-the-fly correction of errors in a very flexible data format. The subblock level has the smaller block delay and provides very fast correction of most of the errors commonly encountered in an average disk file. The second level, on a larger block size, provides detection and correction of additional errors that may be encountered in a weaker device. The block-level decoding may be relatively slow and may include the conventional head-shift and reread functions. This design makes the most use of the available redundancy and provides a substantial match to the error-correction requirements of future disk files. The extended shared relationship of all subblocks to the block-level check bytes is a new structure, not found in other two-stage coding schemes such as concatenated codes [5] or product codes [6].

Error-rate performance of two-level coding schemes

The basic error event is a "byte-in-error." A burst error may cause correlated errors in adjacent bytes; however, sufficient interleaving is assumed to effectively randomize these errors. With appropriate interleaving, all bytes are assumed to be seen by the coding scheme as equally likely to be in error. In

disk files, major defects in the media are avoided by means of surface analysis test and defect-skipping strategy. The error-correction code is expected to provide coverage for errors caused by noise and small defects that cannot be identified easily in the surface analysis test. These errors are usually two to four bits long. Typically two-way or three-way byte interleaving of the code words is adequate in disk-file applications, which will also allow for small amounts of error propagation in the encoding and detection process.

Let p denote the probability of the basic error event—a byte in error. For a given value of p, the probability of any combination of multiple errors in a subblock and a block can then be calculated using binomial and multinomial expressions. The total probability of no-error and correctable-error combinations at the first level of decoding will then lead to the uncorrectable (or miscorrected) error rate at the first level. Similarly, the total probability of no-error or correctable-error combinations at the second level of decoding will lead to the uncorrected (or miscorrected) error rate at the second level.

Let $P_{SB}(t)$ denote the probability that t bytes are in error in a subblock of N bytes. Then $P_{SB}(t)$ can be computed from p as

$$P_{SB}(t) = \binom{N}{t} p^t (1 - p)^{N-t}.$$

Next, we compute the probability $P_B(\leq t_1)$ of up to t_1 errors in each of the n subblocks of a block. This is the probability of no-error and all correctable-error combinations at the first-level decoding:

$$P_B(\leq t_1) = \left[\sum_{t=0}^{t_1} P_{SB}(t) \right]^n.$$

We next compute the probability $P_B(a, b)$ of the event with a errors in one subblock and up to b errors in all the other subblocks, in a block where $b < a$. This is the probability of an additional combination (a, b) of errors corrected at the second level. This probability is given by

$$P_B(a, b) = nP_{SB}(a) \left[\sum_{t=0}^{b} P_{SB}(t) \right]^{n-1} \qquad \text{for } b < a.$$

Now it is easy to compute the total probability of no-error and correctable-error combinations at the first and second levels by combining probabilities of mutually exclusive error combinations covered by both levels of decoding. Let $P_T(t_2, t_1; c)$ denote this probability for a scheme characterized by parameters t_2, t_1, and c. Then we write the total probability as

$$P_T(t_2, t_1; c) = P_B(\leq t_1) + \sum_{\{(a,b)\}} P_B(a, b),$$

where $\{(a, b)\}$ represents the set of all (a, b) combinations correctable at the second level, namely, all values of a such that $t_1 < a \leq t_1 + c$ with $b = t_1$ and $t_1 + c < a \leq t_2$ with $b = 2t_1 + c - a$. From the above three equations, we can express $P_T(t_2, t_1; c)$ as

$$P_T(t_2, t_1; c) = \left[\sum_{t=0}^{t_1} P_{SB}(t) \right]^n + \sum_{a=t_1+1}^{t_1+c} nP_{SB}(a) \left[\sum_{t=0}^{t_1} P_{SB}(t) \right]^{n-1} + \sum_{a=t_1+c+1}^{t_2} nP_{SB}(a) \left[\sum_{t=0}^{2t_1+c-a} P_{SB}(t) \right]^{n-1}.$$

Now we can compute the probability PU_B of uncorrectable error in a block after first-level correction,

$$PU_B(\leq t_1) = 1 - P_B(\leq t_1),$$

and the total probability PU_T of an uncorrectable error for the two-level scheme as

$$PU_T(t_2, t_1; c) = 1 - P_T(t_2, t_1; c).$$

The average number of bytes transferred per uncorrectable error event at the first and second levels can be computed from the error probabilities PU_B and PU_T, respectively, as follows:

Bytes per uncorrectable error at first level

$$= (n \times N)/PU_B,$$

Bytes per uncorrectable error at second level

$$= (n \times N)/PU_T.$$

In **Figure 1** we plot the results of these computations for some typical examples. The line marked $t_1 = 3$ represents the performance of a conventional one-level coding scheme with correction of three errors in every 250-byte (sub)block. The next four lines represent the two-level coding arrangements consisting of 40 subblocks in a block and $(t_2, t_1; c) = (4, 3; 0)$, $(5, 3; 0)$, $(5, 3; 1)$, and $(6, 3; 1)$, showing successive and substantial improvements in error-rate performance provided by the small amount of additional shared redundancy at the second level over $t_1 = 3$ at the first level. Also plotted, for comparison, is the performance of a conventional one-level coding scheme with correction of six errors ($t_1 = 6$) in every 250-byte (sub)block.

Two-level coding in IBM 3380J and 3380K files

The J and K models of the 3380 file use the new two-level coding arrangement for correction of multiple bursts of errors. The data format of a disk track is designed around a two-level architecture consisting of subblocks within a block, combined with interleaved code words. The first-level coding structure of the subblock provides one-symbol correction—the primary capability which is routinely processed on-line at

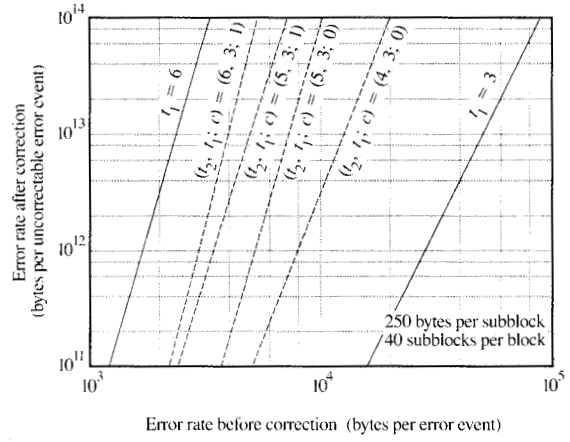

Figure 1

Error-rate performance of two-level coding schemes.

the subblock level. The coding structure is extended to two-symbol correction—the secondary capability at the block level. The code words are two-way interleaved. The data format is shown in **Figure 2**. The 3380 files use the traditional count-key-data (CKD) arrangement for storing and accessing the user data, where each of the count, key, or data fields can be treated as a variable-length block for error-correction coding. Various details relative to count, key, and data fields are excluded from the data format of Figure 2 for the sake of simplicity.

The data are stored in the form of user-defined variable-length blocks (records). Each block is partitioned into fixed-length subblocks, except that the last subblock may be shorter, with fewer user bytes, or may include pad bytes. Each subblock (except the last) consists of 96 user bytes and six first-level check bytes in the form of two interleaved code words. At the end of the block, six additional check bytes are appended, two of which are used for second-level error correction and the remaining four for an overall data-integrity check after correction of the errors at both levels.

The basic error event in the 3380J and 3380K files is a burst error that may affect up to two adjacent bytes. The two-way interleaved two-level code of Figure 2 provides correction of at least one error event in each subblock and up to two error events in any one of the many subblocks of a variable-length block. With this code and an acceptance criterion of one uncorrected or miscorrected record in 10^{12} read bytes, one can accept a device error rate of one error in 10^{7} read bytes using the first-level decoder only and a "weak"-device error rate of one error in 1.2×10^{5} read bytes using both the first- and second-level decoders. This compares with the acceptable device error rate of one error in 2×10^{8} read bytes with the code in the 3380D and E files for an equivalent error performance in reading long records. In the following sections, we present the encoding and decoding processes in the new two-level code, including the details of hardware and software implementations.

Coding equations

The coding equations are created in terms of symbols in GF(2^8). These symbols are represented by 8-bit binary bytes. A subblock consists of two interleaved words of a primary code. The primary code word consists of three check bytes denoted by B_0, B_1, and C_3, and m data bytes denoted by B_2, $B_3, \cdots, B_{m+1}$, which satisfy the following modulo-2 matrix equations:

$$B_0 \oplus TB_1 \oplus T^2B_2 \oplus \cdots \oplus T^{m+1}B_{m+1} = 0, \quad (1)$$

$$B_0 \oplus T^2B_1 \oplus T^4B_2 \oplus \cdots \oplus T^{2(m+1)}B_{m+1} = 0, \quad (2)$$

$$B_0 \oplus T^3B_1 \oplus T^6B_2 \oplus \cdots \oplus T^{3(m+1)}B_{m+1} = C_3, \quad (3)$$

where $\oplus$ signifies modulo-2 sum, B_i is an 8-bit column vector, $i = 0, 1, \cdots, m+1 \le (2^8 - 2)$, T is a companion matrix of a primitive polynomial of degree 8, and T^i denotes T multiplied by itself i times with modulo-2 addition. We use $m = 48$ and the following T matrix corresponding to the polynomial $1 + x^3 + x^5 + x^7 + x^8$:

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \quad (4)$$

The code given by Equations (1), (2), and (3) is an extended Reed–Solomon code [1, 7, 8] with Hamming distance 4. The 8-bit column vectors represent the field elements in GF(2^8), and multiplication by the matrix T^i corresponds to the multiplication by α^i, where α is the primitive element represented by the first column of the matrix T. The matrix T^i for any positive integer i can be computed from T. Then, for a given matrix T^i, it is easy to construct a hard-wired circuit to compute modulo-2 product T^iB corresponding to any input byte B. **Figure 3** illustrates such a circuit for the matrix T^3.

The encoding for the first two check bytes B_0 and B_1 can be performed by means of a shift-register network built for modulo-$g(x)$ operations, where $g(x)$ is a polynomial with roots α and α^2, as given by

$$g(x) = (x \oplus \alpha)(x \oplus \alpha^2)$$
$$= x^2 \oplus (\alpha \oplus \alpha^2)x \oplus \alpha^3. \quad (5)$$

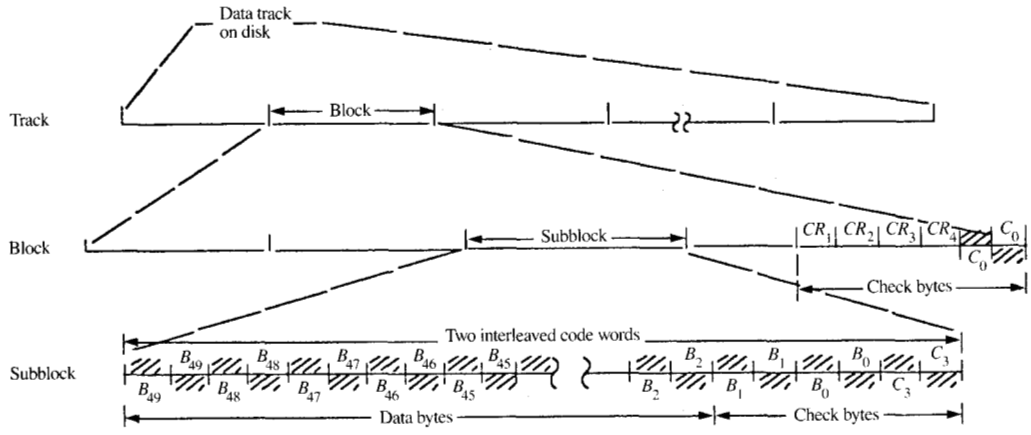

Figure 2

Data format.

$$T^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

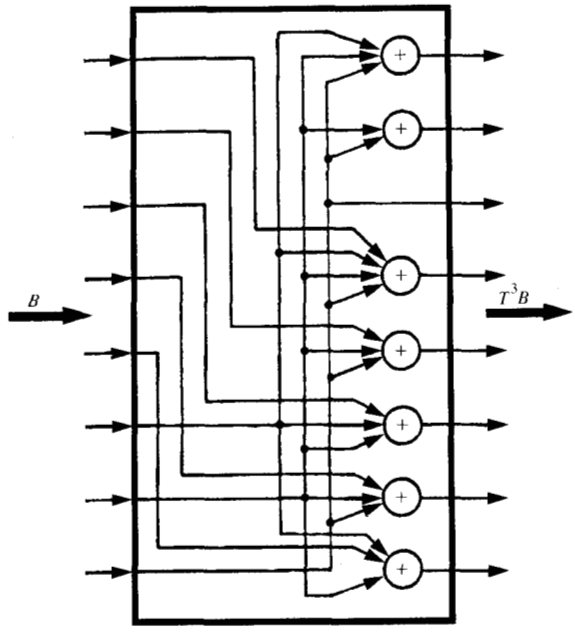

Figure 3

Matrix multiplier T^3.

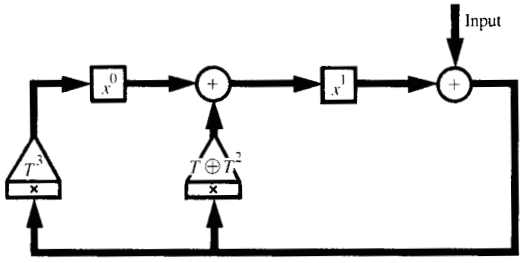

Figure 4

Block diagram of the encoding shift register for B_0 and B_1.

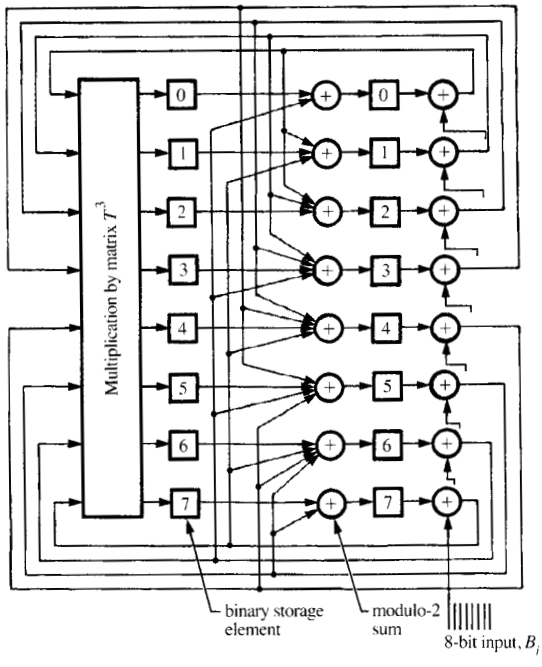

Figure 5

Encoding shift register for B_0 and B_1.

The block diagram of this shift register appears in **Figure 4**, and the shift register with binary-circuit components is shown in **Figure 5**. The check bytes B_0 and B_1 are computed by processing the data bytes in this shift register. Initially, the register is set to all zeros. The ordered sequence of data bytes $B_{m+1}, B_m, B_{m-1}, \cdots, B_3, B_2$ are entered into the register in m successive shifts as 8-bit parallel vector inputs. At the end of this operation, the register contains the check bytes B_0 and B_1 in its low-order and high-order positions, respectively.

Unlike the computations of B_0 and B_1, the computation of C_3 in each subblock is carried out separately by means of a shift register shown in **Figure 6** and the multiplier T^3 of Figure 3. Initially, the register is set to all zeros. The ordered sequence of subblock bytes $B_{m+1}, B_m, B_{m-1}, \cdots, B_1, B_0$ are entered into the register in $m + 2$ successive shifts. At the end of this operation, the register contains the check byte C_3 corresponding to the subblock.

The block consists of any number of subblocks, say n, with one additional check byte denoted by C_0 at the end. This check byte is the modulo-2 sum of all subblock bytes excluding C_3 and accumulated over all subblocks as specified by the following modulo-2 matrix equation:

$$C_0 = \sum_{subbk=1}^{n} \left(\sum_{i=0}^{m+1} B_i \right)_{subbk} . \tag{6}$$

The primary code words described above are two-way interleaved. Thus, there are six check bytes at the end of each subblock and two check bytes at the end of the block that provide the desired error-correction capability. The remaining four check bytes at the end of the block are for data integrity check over the corrected data stream. This error-detection part of the second-level code is described in a separate section entitled *Data integrity check.*

It is readily seen that for $n = 1$, Equations (1), (2), (3), and (6) together represent a code that is an extended Reed–Solomon code for correction of two-symbol errors. In the case of n greater than 1, the block-level code word can be viewed as modulo-2 superposition of n subblock code words. Two-symbol errors in this superpositioned code word are correctable. Suppose that a block consisting of n subblocks encounters multiple symbols in error. If these errors are located in separate subblocks, each error will be corrected as a single-symbol error in the corresponding subblock, and the block-level error syndromes will vanish by the corrected error patterns. If one of the subblocks has two symbols in error, the subblock-level processing will detect these errors and correct all others as single-symbol errors. Then, at the block level, the subblock-level syndromes, together with block-level syndromes, will be reprocessed for the correction of the subblock with two symbols in error. If any subblock has more than two symbols in error, or if two or more subblocks have multiple symbols in error, these errors cannot be corrected.

• *Decoding process: First level*

Let $\hat{B}_i$ and $\hat{C}_3$ denote the read bytes corresponding to the written bytes B_i and C_3, respectively. The syndromes of error at the subblock level are denoted by S_1, S_2, and S_3,

corresponding to the coding equations (1), (2), and (3), respectively, and are given by

$$S_1 = \hat{B}_0 \oplus T\hat{B}_1 \oplus T^2\hat{B}_2 \oplus \cdots \oplus T^{m+1}\hat{B}_{m+1}, \quad (7)$$

$$S_2 = \hat{B}_0 \oplus T^2\hat{B}_1 \oplus T^4\hat{B}_2 \oplus \cdots \oplus T^{2(m+1)}\hat{B}_{m+1}, \quad (8)$$

$$S_3 = \hat{B}_0 \oplus T^3\hat{B}_1 \oplus T^6\hat{B}_2 \oplus \cdots \oplus T^{3(+1)}\hat{B}_{m+1} \oplus \hat{C}_3. \quad (9)$$

The syndromes S_1 and S_2 are computed by processing the ordered sequence of read bytes $\hat{B}_{m+1}, \hat{B}_m, \hat{B}_{m-1}, \cdots, \hat{B}_1, \hat{B}_0$ in two shift registers with premultipliers T and T^2, respectively. The shift registers are similar to that shown in Figure 6 except for the premultiplier circuit. Each shifting operation multiplies the previous content of the register by the multiplier matrix and then combines (by modulo-2 vector addition) the results with the entering byte. The resultant 8-bit byte becomes the new content. After $m + 2$ shifting operations, the two shift registers contain the syndromes S_1 and S_2 in accordance with Equations (7) and (8), respectively. The syndrome S_3 is computed similarly except that B_0 is EX-ORed with C_3 before entering the shift register, and the multiplier matrix is T^3.

A nonzero value of S_1, S_2, or S_3 indicates the presence of an error. Suppose the subblock has only one byte in error. Let x and E_x denote the error location and error pattern, respectively. That is,

$$\hat{B}_i = \begin{cases} B_x \oplus E_x & \text{for } i = x, \\ B_i & \text{for } i \neq x. \end{cases} \quad (10)$$

Then, in view of Equations (1)–(3) and (7)–(10), the syndromes reduce to

$$S_1 = T^x E_x, \quad (11)$$

$$S_2 = T^{2x} E_x, \quad (12)$$

$$S_3 = T^{3x} E_x. \quad (13)$$

Thus, the decoding equation is given by

$$E_x = T^{-x} S_1 = T^{-2x} S_2 = T^{-3x} S_3 \quad (14)$$

if B_x is in error.

Note that a one-byte error in check byte C_3 affects the syndrome S_3 only. In that case the decoding equation is given by

$$0 = S_1 = S_2 \neq S_3 \qquad \text{if } C_3 \text{ is in error.} \quad (15)$$

The decoding can be accomplished by means of three shift registers, with T^{-1}, T^{-2}, and T^{-3} multiplier circuits. With S_1, S_2, and S_3 as initial contents, the three registers are shifted and the contents are compared after each shifting operation until they are equal. The resultant number of shifts determines x, and the final content in each register is the error pattern E_x. The byte $\hat{B}_x$ is then corrected as $\hat{B}_x \oplus E_x$.

The hardware implementation of the decoding function described above is shown in **Figure 7**. The syndromes S_1, S_2, and S_3 are entered into appropriate shift registers at clock

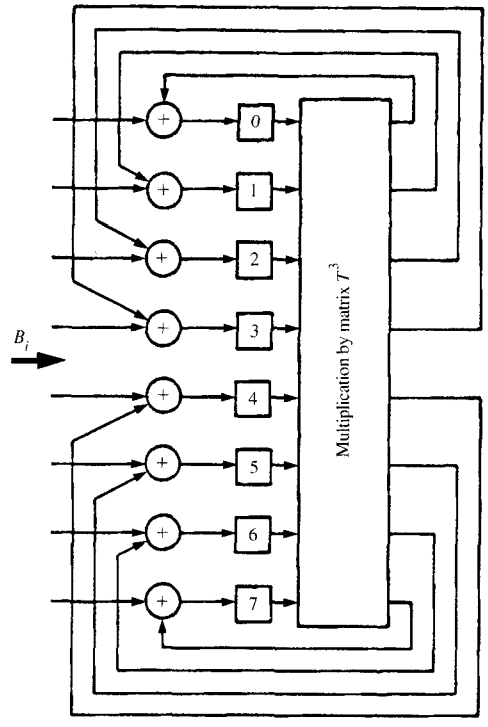

Figure 6

Encoding shift registers for C_3.

zero. Each clock cycle generates a shifting operation of the decoding shift registers, creates a zero or nonzero error pattern for the corresponding byte position, and delivers a corrected data byte. After $m + 1$ clock cycles, if none of the bytes received a correction (indicated by latch L_1 off) and S_1 and S_2 both are not 0 (indicated by latch L_2 on), the decoder declares an uncorrected subblock error by turning on the UE signal. When the UE signal is on, the uncorrected subblock is flagged and the original syndromes S_1, S_2, and S_3 are passed on to the block level for two-error correction.

Note that the corrected bytes in the decoder of Figure 7 are delivered in a reverse order compared to that in the encoder of Figure 5. If desirable, this reversal can be eliminated by modifying the decoder as follows. We substitute $(m + 1) - \bar{x}$ for x in Equation (14) and rewrite it as

$$\begin{aligned} E_{(m+1)-\bar{x}} &= T^{\bar{x}}[T^{-(m+1)}S_1] \\ &= T^{2\bar{x}}[T^{-2(m+1)}S_2] \\ &= T^{3\bar{x}}[T^{-3(m+1)}S_3]. \end{aligned} \quad (16)$$

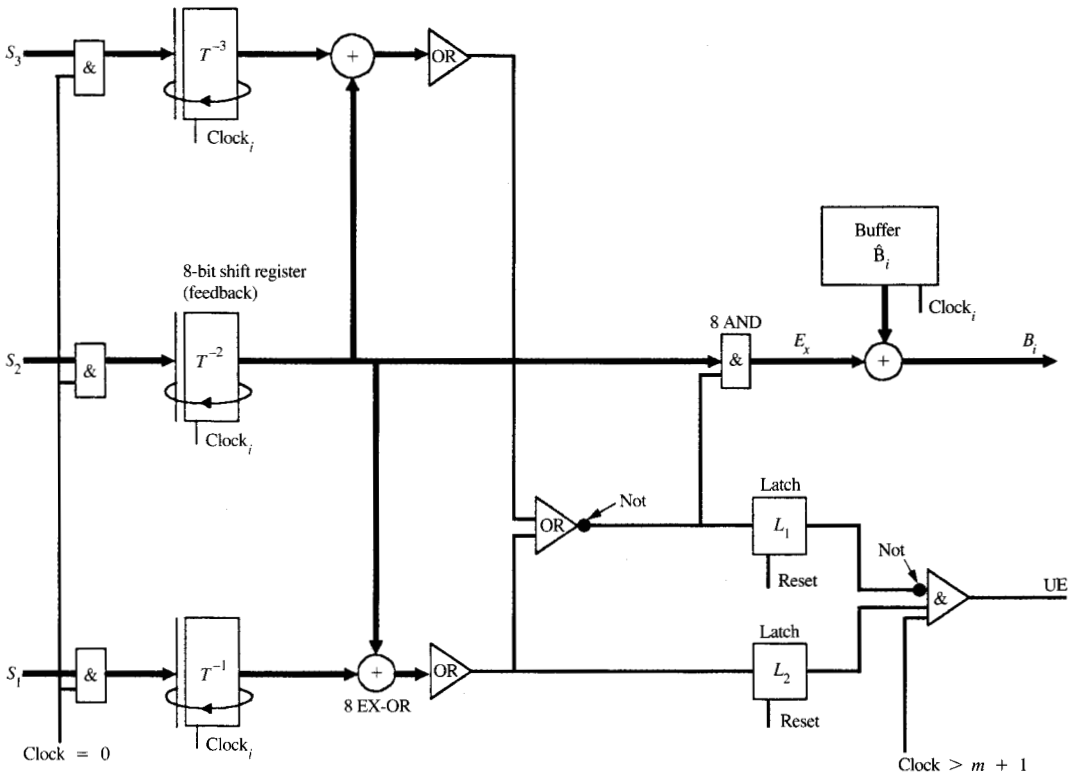

Figure 7

First-level decoder.

In this equation, $\hat{x}$ represents the number of shifts, and $\hat{x} = 0, 1, 2, \cdots, m, m+1$ corresponds to the byte-position values $x = m+1, m, m-1, \cdots, 1, 0$, thereby canceling the reversal of the byte order. To accomplish this, the decoder hardware in Figure 7 must be modified in accordance with Equation (16) as follows:

1. The shift-register multipliers T^{-1}, T^{-2}, and T^{-3} in Figure 7 must be replaced by the multipliers T, T^2, and T^3, respectively.
2. The syndromes S_1, S_2, and S_3 must be premultiplied by $T^{-(m+1)}$, $T^{-2(m+1)}$, and $T^{-3(m+1)}$, respectively, before being entered into the respective shift registers. Note that when $(m+1) = 255$, this premultiplication is not required, since $T^{255} = I$.

In general, the circuits for premultiplication by $T^{-(m+1)}$, $T^{-2(m+1)}$, and $T^{-3(m+1)}$ depend on the value of m, and each circuit requires a small number of EX-OR gates.

With these modifications, the decoder delivers the corrected bytes in the same order as they were processed by the encoder, starting with B_{m+1} and ending with B_0.

• *Decoding process: Second level*

The syndrome of error at the block level is denoted by S_0, corresponding to the coding equation (6), and is given by

$$S_0 = \hat{C}_0 \oplus \sum_{subbk} \left(\sum_i \hat{B}_i \right)_{subbk} . \tag{17}$$

This syndrome is computed as the modulo-2 sum of the (first-level-corrected) read bytes excluding the check bytes $\hat{C}_3$ for all subblocks and including the block-level check byte $\hat{C}_0$.

If each subblock in a block encounters, at the most, one byte in error, they will all be corrected by the first-level processing. In the absence of any uncorrected error, the second-level syndrome S_0 will be zero. However, if one of

the subblocks encounters two bytes in error, then the first-level processing for that subblock will give an uncorrectable-error (UE) signal and pass on the syndromes S_1, S_2, and S_3 to the second level for further processing. Let y and z denote the locations and E_y and E_z denote the corresponding patterns for the uncorrected two errors. Then the second-level syndrome equation (17) for S_0 and the first-level syndrome equations (7), (8), and (9) for S_1, S_2, and S_3 will reduce to the following relations in terms of the error locations y and z and the patterns E_y and E_z for the two errors:

$$S_0 = E_y \oplus E_z, \tag{18}$$

$$S_1 = T^y E_y \oplus T^z E_z, \tag{19}$$

$$S_2 = T^{2y} E_y \oplus T^{2z} E_z, \tag{20}$$

$$S_3 = T^{3y} E_y \oplus T^{3z} E_z. \tag{21}$$

Next, we proceed to decode the combined set of subblock- and block-level syndromes for two-symbol errors. First we obtain the 8-digit vectors P, Q, and R, which are functions of the syndromes S_0, S_1, S_2, and S_3, as given by

$$P = (S_2 \otimes S_2) \oplus (S_3 \otimes S_1), \tag{22}$$

$$Q = (S_2 \otimes S_1) \oplus (S_3 \oplus S_0), \tag{23}$$

$$R = (S_0 \otimes S_2) \oplus (S_1 \otimes S_2), \tag{24}$$

where $\otimes$ denotes the product operation of the field elements in GF(2^8), and the field elements are represented by binary 8-digit vectors. The product operation can be realized using hard-wired logic or through the use of log and antilog tables in GF(2^8), as discussed in Appendix A.

We note here that P, Q, and R are necessarily nonzero when there are two bytes in error and both are in one of the subblocks. In contrast, when the check byte C_0 or C_3 is among the two erroneous bytes, this is indicated by $P = R = 0$.

Assertion 1 Suppose that there are exactly two bytes in error in one of the subblocks; then, the error-location values y and z are two unique solutions of i in the equation

$$T^{-2i} P \oplus T^{-i} Q = R, \tag{25}$$

where P, Q, and R are functions of the syndromes S_0, S_1, S_2, and S_3, as given by Equations (22)–(24). For each of the two solution values of i, the error pattern is given by

$$E_i = R/(T^{2i} S_0 \oplus S_2). \tag{26}$$

The proof of Assertion 1 appears in Appendix B.

The decoding can be accomplished using a software method involving table-lookup operations. The vectors P, Q, and R are computed from syndromes S_0, S_1, S_2, and S_3 using the log and antilog tables of Appendix A. This requires, at the most, 18 references to the tables in memory, six binary-add (modulo-255) operations, and three vector-add (modulo-2) operations.

The error-location values y and z can be obtained through a simple table-lookup procedure. The table and the theory behind this procedure appear in Appendix C. In this method, the error-location values y and z are obtained through the following four-step procedure:

1. Obtain binary numbers u and v from vectors P, Q, and R, using log tables of Appendix A:

$$u = (\log_\alpha P - \log_\alpha Q) \text{ modulo } 255, \tag{27}$$

$$v = (\log_\alpha R - \log_\alpha Q) \text{ modulo } 255. \tag{28}$$

2. Obtain the value d:

$$d = (u + v) \text{ modulo } 255. \tag{29}$$

3. Obtain the value t, corresponding to value d, from Table 1 of Appendix C.
4. Obtain error-location values y and z:

$$y = (u - t) \text{ modulo } 255, \tag{30}$$

$$z = (t - v) \text{ modulo } 255. \tag{31}$$

All terms in Equations (27)–(31) of the above procedure are 8-digit binary sequences undergoing modulo-255 add or subtract operations. The procedure requires four table-lookup operations, four modulo-255 subtract operations, and one modulo-255 add operation. In this procedure, an invalid value of d (the one with no entry in Table 1) or an invalid value for y or z (greater than $m + 1$) indicates an uncorrectable error involving three or more bytes in error.

The error pattern E_y can be computed using the log and antilog tables of Appendix A in accordance with Equation (26), in which matrix multiplication $T^{2y} S_0$ is replaced by the product $\alpha^{2y} \otimes S_0$ of the two field elements α^{2y} and S_0. The error pattern E_z can be computed similarly, using Equation (26), or alternatively from Equation (18), which gives

$$E_z = S_0 \oplus E_y. \tag{32}$$

The subblock correction is then accomplished by correcting bytes $\hat{B}_y$ and $\hat{B}_z$ with error patterns E_y and E_z.

Alternatively, the second-level processing [Equations (22)–(26)] can be carried out with a hardware decoder [4] using shift registers and binary logic.

• *Data integrity check*

In the IBM 3380J and 3380K disk files, the two-level code includes four additional check bytes at the second level (Figure 2). These check bytes provide a strong data-integrity check over the corrected data stream. Each of these four check bytes is applied to both code words, disregarding the two-way interleaving. They are denoted as CRC checks and are given by

$$CRC1 = \sum_i T^{-i} D_i, \quad (33)$$

$$CRC2 = \sum_i T^{-2i} Z_i, \quad (34)$$

$$CRC3 = \sum_i T^{-3i} D_i, \quad (35)$$

$$CRC4 = \sum_i T^{-4i} Z_i, \quad (36)$$

where matrix T^{-i} and the sequences $\{D_i\}$ and $\{Z_i\}$ are defined as follows.

The matrix T is the companion matrix of a degree-8 primitive polynomial $P(x)$, and $T^{-i} = T^{255-i}$. We use different powers of the same T to create all CRC as well as error-correction functions. It provides a computational convenience when second-level error correction requires recomputation of the CRC check in software. In particular, the software will be able to use the same tables for computation of Galois field operations.

The byte-sequence $\{D_i\}$ and the byte-sequence $\{Z_i\}$ consist of all bytes in a record excluding the second-level check bytes; $\{Z_i\}$ also excludes two check bytes corresponding to the last two byte positions of each *normal-length* subblock. The first-level check bytes in these sequences (and the corresponding error patterns) may be replaced by null bytes (all-zeros) in CRC calculations. This eliminates the need for correcting errors in the check bytes.

The sequence $\{Z_i\}$ is offset from the sequence $\{D_i\}$ by two bytes at the end of each subblock. With 255 as the exponent for T^{-1} and T^{-2}, this offset arrangement provides an effective CRC cycle length of $255 \times 256 = 65\,280$ bytes in sequence $\{D_i\}$ and $255 \times 254 = 64\,770$ bytes in sequence $\{Z_i\}$. This means that two identical detectable error patterns will not cancel each other in both types of CRC checks in any record of up to 255 subblocks of data. This is a novel method of obtaining long cycle length while still operating in a Galois field of reasonable size, namely GF(2^8).

In the readback process, the CRC syndromes are generated on-the-fly along with the error-correction syndromes as the data are received, using shift registers with premultipliers T^{-1}, T^{-2}, T^{-3}, and T^{-4}. Furthermore, CRC syndromes will be adjusted for the first-level error correction in hardware and later for the second-level error correction in software. The equations for the four CRC syndromes S_{-1}, S_{-2}, S_{-3}, and S_{-4} are

$$S_{-1} = \hat{C}RC1 \oplus \sum_i T^{-i} \hat{D}_i \oplus \sum_i T^{-i} E_1(D_i) \oplus \sum_i T^{-i} E_2(D_i), \quad (37)$$

$$S_{-2} = \hat{C}RC2 \oplus \sum_i T^{-2i} \hat{Z}_i \oplus \sum_i T^{-2i} E_1(Z_i) \oplus \sum_i T^{-2i} E_2(Z_i), \quad (38)$$

$$S_{-3} = \hat{C}RC3 \oplus \sum_i T^{-3i} \hat{D}_i \oplus \sum_i T^{-3i} E_1(D_i) \oplus \sum_i T^{-3i} E_2(D_i), \quad (39)$$

$$S_{-4} = \hat{C}RC4 \oplus \sum_i T^{-4i} \hat{Z}_i \oplus \sum_i T^{-4i} E_1(Z_i) \oplus \sum_i T^{-4i} E_2(Z_i), \quad (40)$$

where the symbol $\hat{}$ indicates readback bytes, $E_1(D_i)$ and $E_1(Z_i)$ represent the first-level error-correction patterns, and $E_2(D_i)$ and $E_2(Z_i)$ represent the second-level error-correction patterns corresponding to the recorded bytes D_i and Z_i, respectively. The error patterns corresponding to the check bytes will be replaced by null bytes.

The data sequence $\{D_i\}$ and the corresponding sequence of error-correction patterns $\{E_1(D_i)\}$ appear at the input and output of the first-level decoder, respectively, with exactly one subblock delay. If second-level error correction is required, the error patterns $\{E_2(D_i)\}$ will be available later through software decoding at the second level.

If the first-level decoder includes a subblock buffer and on-the-fly error correction, first-level CRC syndromes can be computed from the corrected data which is the combined sequence $\{D_i \oplus E_1(D_i)\}$. However, if the first-level error correction is deferred, the first-level CRC syndromes may be computed by combining separate computations with $\{D_i\}$ and $\{E_1(D_i)\}$ sequences.

The second-level error correction, if any, will involve one of the subblocks and up to four error bytes. The byte count i in sequence $\{D_i\}$ for each of the four error bytes will be determined from the error-location numbers. Then the terms such as $T^{-3i} E_2(D_i)$ for each error can be computed using the antilog tables,

$$T^{-3i} E_2(D_i) = \log_\alpha^{-1} \{[\log_\alpha E_2(D_i) - 3i] \text{ modulo } 255\}. \quad (41)$$

This leads to the second-level CRC check corresponding to the syndromes S_{-1} and S_{-3}. The byte count i in sequence $\{Z_i\}$ for each of the four errors will also be determined, and the computations for syndromes S_{-2} and S_{-4} will be done in a similar manner. These final values of CRC syndromes must be zero if no errors are present.

The four CRC bytes in two-level code provide error detection for protection against miscorrections of excessive errors. However, any errors in CRC bytes will tend to create unnecessary reread operations with undue performance penalty. In order to avoid this, one nonzero CRC check on the $\{D_i\}$ sequence and one on the $\{Z_i\}$ sequence may be ignored whenever errors are not corrected at the second level. All nonzero CRC checks may be ignored whenever errors are not corrected at either level. Alternatively, the

CRC bytes may be placed within the last subblock so that the two-level correction is applicable to the CRC bytes as well.

The four CRC check bytes provide more than adequate data integrity check. Four separate CRC checks in GF(2^8) using the same primitive field element are not only convenient but also provide the desired long cycle length without using degree-16 polynomials.

Conclusion

A two-level coding scheme with subblocks within a block is defined with parameters (t_2, t_1; c). A general result regarding various combinations of correctable error patterns at the two levels is established, and a practical decoding strategy is given. The scheme is designed for disk files; the emphasis is on correcting multiple-burst errors on-line in most devices, while providing reserve capability at the block level for protecting an uncommon weaker or failing device against potential data loss.

The two-level coding scheme used in the IBM 3380J and 3380K disk files is described in detail, including the encoding and decoding implementations. This design is characterized by the two-level coding parameters ($t_2 = 2$, $t_1 = 1$; $c = 1$) using the coding structure of the extended Reed–Solomon code with two-way interleaving. A three-register hardware decoder provides correction of one-symbol errors and detection of two-symbol errors in each subblock at the first level. A table-lookup software decoder provides correction of two-symbol errors in one of the subblocks at the second level. An additional check at the end of the block provides an overall data integrity confirmation against miscorrections in the presence of an excessive number of errors.

Appendix A: Logarithms of field elements

Let $G(x)$ denote a primitive polynomial of degree 8 with binary coefficients,

$$G(x) = g_0 \oplus g_1 x \oplus g_2 x^2 \oplus \cdots \oplus g_7 x^7 \oplus x^8 .$$

The companion matrix of the polynomial $G(x)$ is defined as the following nonsingular matrix:

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & g_0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & g_1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & g_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & g_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & g_4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & g_5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & g_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & g_7 \end{bmatrix} .$$

We use matrix T of Equation (4) corresponding to $G(x) = 1 + x^3 + x^5 + x^7 + x^8$. The matrix T^i denotes T multiplied by itself i times with all results reduced modulo 2. The matrices $T, T^2, T^3, \cdots, T^{255}$ are all distinct, and T^{255} is the identity matrix, which can also be written as T^0. These 255 matrices represent ($2^8 - 1$) nonzero elements of GF(2^8). Let α denote the primitive element of GF(2^8). Then T^i represents the nonzero element α^i for all i. The zero element is represented by the 8×8 all-zero matrix. The sum and product operations in GF(2^8) are, then, defined by modulo-2 matrix-sum and matrix-product operations, using these matrix representations of the field elements.

The elements of GF(2^8) can also be represented by the 8-digit binary vectors. The square matrices in the above representation are very redundant. In fact, each matrix can be uniquely identified by just one of its columns (in a specific position), which can very well be used for representation of the corresponding field element without ambiguity. In particular, the first column of each 8×8 matrix in the above set is the commonly used 8-digit vector representation of the corresponding field element. This establishes a one-to-one correspondence between the set of all nonzero 8-digit vectors and the set of T^i matrices representing the field elements α^i. Thus, each nonzero 8-digit vector S corresponds to a unique integer i ($0 \le i \le 254$) which can be regarded as its logarithm to the base α. That is,

$$i = \log_\alpha S$$

and

$$S = \log_\alpha^{-1} i.$$

A table of logarithms which maps all field elements into powers of α and a table of antilogarithms which maps integer powers of α into corresponding field elements can be generated using the companion matrix of Equation (4) as the representation for α. Each table can be stored in a memory of 8×256 bits in which the word number or memory location expressed as an 8-bit vector is the input vector. The stored 8-bit vector in that memory location represents the logarithm and the antilogarithm corresponding to the input vector in the two tables, respectively. With the help of these tables, the product $S_1 \otimes S_2$ (of the two elements represented by 8-digit vectors S_1 and S_2) can be computed as follows:

$S_1 = \alpha^{i_1}$	Use log table: $\log_\alpha S_1 = i_1$.
$S_2 = \alpha^{i_2}$	Use log table: $\log_\alpha S_2 = i_2$.
$S_1 \otimes S_2 = \alpha^{i_1 + i_2}$	Add (modulo 255): $i = i_1 + i_2$.
$\alpha^i = S$	Use antilog table: $\log_\alpha^{-1} i = S$.

Appendix B: Proof of Assertion 1

Assertion 1 provides the decoding algorithm for two-symbol errors. This assertion follows from the general results in decoding Reed–Solomon or the generalized BCH code [4, 9]. Here, we present a proof for this assertion from the basic matrix equations.

Assertion 1 Suppose that there are exactly two bytes in error in one of the subblocks; then the location values y and z for these bytes are two unique solutions of i in the equation

$$T^{-2i}P \oplus T^{-i}Q = R, \tag{B1}$$

where P, Q, and R are functions of the syndromes S_0, S_1, S_2, and S_3, as given by Equations (22)–(24). The error pattern E_i for $i = y$ or $i = z$ is given by

$$E_i = R/(T^{2i}S_0 \oplus S_2). \tag{B2}$$

Proof The syndromes are expressed as a function of the two errors in Equations (18)–(21). These equations are rewritten here as

$$S_0 = E_y \oplus E_z, \tag{B3}$$

$$S_1 = T^y E_y \oplus T^z E_z, \tag{B4}$$

$$S_2 = T^{2y} E_y \oplus T^{2z} E_z, \tag{B5}$$

$$S_3 = T^{3y} E_y \oplus T^{3z} E_z. \tag{B6}$$

Combining the appropriate equations from (B3) through (B6), we have

$$T^y S_0 \oplus S_1 = (T^y \oplus T^z)E_z, \tag{B7}$$

$$T^y S_1 \oplus S_2 = T^z(T^y \oplus T^z)E_z, \tag{B8}$$

$$T^y S_2 \oplus S_3 = T^{2z}(T^y \oplus T^z)E_z. \tag{B9}$$

Matrix equations (B7), (B8), and (B9) are relations among field elements in GF(2^8) represented by matrices. In particular, the matrix multiplication of the type $T^i B$ represents the product of field element α^i and β, where α^i is represented by the first column of matrix T^i, and β is represented by the column vector B. In view of this interpretation, Equations (B7), (B8), and (B9) yield the relationship

$$(T^y S_0 \oplus S_1) \otimes (T^y S_2 \oplus S_3) = (T^y S_1 \oplus S_2)_2 \otimes (T^y S_1 \oplus S_2), \tag{B10}$$

where $\otimes$ denotes the product of corresponding elements in GF(2^8). Equation (B10) can be rearranged into the matrix equation

$$T^{2y}R \oplus T^y Q \oplus P = 0. \tag{B11}$$

In these equations, P, Q, and R are column vectors given by

$$P = (S_2 \otimes S_2) \oplus (S_3 \otimes S_1), \tag{B12}$$

$$Q = (S_2 \otimes S_1) \oplus (S_3 \otimes S_0), \tag{B13}$$

$$R = (S_0 \otimes S_2) \oplus (S_1 \otimes S_1). \tag{B14}$$

Thus y is one of the solutions for i in the equation

$$T^{-2i}P \oplus T^{-i}Q = R. \tag{B15}$$

By exchanging the variables y and z in the above process, we can show that z is the second solution for i in Equation (B15).

Equation (B2) for $i = y$ and $i = z$ can be verified by direct substitution of values for R, S_0, S_1, and S_2 using Equations (B14), (B3), (B4), and (B5), respectively. This completes the proof of Assertion 1.

Appendix C: Table-lookup solution for two error locations

In Assertion 1 of Appendix B, it was shown that the error locations y and z for two errors in a subblock can be determined by solving for i in Equation (B1). That equation is rewritten here as

$$T^{-2i}P \oplus T^{-i}Q = R. \tag{C1}$$

The constants P, Q, and R are functions of the syndromes S_0, S_1, S_2, and S_3, given by Equations (22)–(24), respectively. We can obtain logarithms of P, Q, and R from the log table of Appendix A:

$$p = \log_\alpha P, \tag{C2}$$

$$q = \log_\alpha Q, \tag{C3}$$

$$r = \log_\alpha R. \tag{C4}$$

Then the matrix equation (C1) can be rewritten as a relation among field elements in GF(2^8) as follows:

$$\alpha^{-2i}\alpha^p \oplus \alpha^{-i}\alpha^q = \alpha^r. \tag{C5}$$

Multiplying both sides of Equation (C5) by α^{p-2q}, we obtain

$$\alpha^{(2p-2q-2i)} \oplus \alpha^{(p-q-i)} = \alpha^{(p-2q+r)}. \tag{C6}$$

Substituting t for $(p - q - i)$ in Equation (C6), we get

$$\alpha^{2t} \oplus \alpha^t = \alpha^{(p-2q+r)} \tag{C7}$$

and

$$i = (p - q) - t \text{ modulo } 255. \tag{C8}$$

The right-hand side of Equation (C7) is a known field element α^d, where

$$d = (p - q) + (r - q) \text{ modulo } 255. \tag{C9}$$

Next we present a simple table-lookup solution for Equation (C7), which can be rewritten as

$$\alpha^t(\alpha^t \oplus \alpha^0) = \alpha^d. \tag{C10}$$

Using this expression, we can relate each value of t (from 0 to 254) to a value of d. We note that some values of d are absent in this relationship, and that each valid value of d corresponds to two values of t. For a given value of d, if $t = t_1$ is one of the solutions of Equation (C10), it is easy to see that $t = t_2$ is also a solution, where

$$\alpha^{t_2} = \alpha^{t_1} \oplus \alpha^0. \tag{C11}$$

Table 1 Table of t vs. d in $\alpha^d = \alpha^t(\alpha^t \oplus \alpha^0)$.

d	t	d	t	d	t	d	t	d	t
000	01010101	051	10001010	102	00010101	153	01000101	204	00101010
001	------	052	------	103	10001011	154	10100101	205	11100101
002	------	053	01001011	104	------	155	11001011	206	00010111
003	00111000	054	------	105	00011111	156	10101011	207	------
004	------	055	10010111	106	10010110	157	00101110	208	------
005	------	056	00000100	107	10011100	158	00100001	209	------
006	01110000	057	01010111	108	------	159	------	210	00111110
007	10000000	058	------	109	10010011	160	------	211	00100100
008	------	059	01011100	110	00101111	161	------	212	00101101
009	------	060	------	111	------	162	00110101	213	------
010	------	061	01000010	112	00001000	163	------	214	00111001
011	00001100	062	------	113	------	164	------	215	------
012	00101011	063	------	114	10101110	165	00101001	216	------
013	------	064	------	115	01111001	166	00111101	217	11100010
014	00000001	065	------	116	------	167	01001000	218	00100111
015	------	066	------	117	------	168	01001101	219	------
016	------	067	------	118	10111000	169	01001111	220	01011110
017	01110111	068	01100110	119	00010110	170	00100010	221	01011000
018	------	069	01101010	120	------	171	------	222	------
019	01000001	070	------	121	------	172	00011011	223	00100110
020	------	071	------	122	10000100	173	00111011	224	00010000
021	01101011	072	------	123	------	174	------	225	------
022	00011000	073	------	124	------	175	------	226	------
023	------	074	------	125	------	176	11000000	227	------
024	01010110	075	01010010	126	------	177	------	228	01011101
025	------	076	00000101	127	10011000	178	01000110	229	------
026	------	077	01111010	128	------	179	11000101	230	11110010
027	------	078	01111000	129	00011100	180	00100101	231	------
028	00000010	079	10010000	130	------	181	01001110	232	------
029	------	080	------	131	01000000	182	11001001	233	00010010
030	------	081	10011010	132	------	183	------	234	------
031	------	082	------	133	00000110	184	------	235	------
032	------	083	10011110	134	------	185	10111100	236	01110001
033	------	084	10100110	135	------	186	------	237	------
034	00110011	085	00010001	136	10111011	187	00001011	238	00101100
035	------	086	10001101	137	10100000	188	------	239	00010011
036	------	087	------	138	10110101	189	------	240	------
037	------	088	01100000	139	------	190	------	241	------
038	10000010	089	00100011	140	------	191	01001100	242	------
039	00111100	090	10010010	141	------	192	00001110	243	------
040	------	091	01110110	142	------	193	00100000	244	00001001
041	------	092	------	143	------	194	00000011	245	------
042	01010011	093	------	144	------	195	------	246	------
043	01100100	094	------	145	------	196	01010000	247	01101110
044	00110000	095	------	146	------	197	------	248	------
045	01001001	096	00000111	147	00011110	198	------	249	------
046	------	097	10000001	148	------	199	------	250	------
047	------	098	00101000	149	00110010	200	------	251	00110111
048	10000011	099	------	150	10100100	201	00001111	252	------
049	00010100	100	------	151	------	202	00011001	253	01100010
050	------	101	10001100	152	00001010	203	------	254	00110001

The field element α is represented by matrix T of Equation (4).
The absence of a value for t implies an invalid value of d.

Substituting $t = t_1$ in Equation (C10) and then using (C11), we see that

$$\alpha^{t_1}\alpha^{t_2} = \alpha^d. \tag{C12}$$

This expression provides a simple way to obtain the second value of t from the first:

$$t_2 = (d - t_1) \text{ modulo } 255. \tag{C13}$$

Table 1 relates each valid value of d to one of the two values of t. Values of d are listed in ascending order for easy reference as addresses of an 8×256-bit memory. The corresponding value of t is stored in the memory as an 8-bit binary number. The all-zeros vector (invalid value) is stored at addresses corresponding to the invalid values of d, and is so interpreted.

In the case of two errors, the computed value of d fetches one of the two values for t from Table 1. The second value for t is obtained from the first by using Equation (C13). With these two values of t, Equations (C8) and (C9) provide the two values of i as the error locations y and z given by

$$y = (p - q) - t \qquad \text{modulo } 255, \tag{C14}$$

$$z = t - (r - q) \qquad \text{modulo } 255. \tag{C15}$$

An invalid value of d fetches $t = 0$ from Table 1, which is interpreted as an uncorrectable error involving three or more bytes of the code word.

References

1. I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *J. Soc. Indust. Appl. Math.* **8,** 300–304 (1960).
2. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Group Codes," *Info. Control* **3,** 68–79 (1960).
3. A. M. Patel, U.S. Patents 4,525,838 and 4,706,250, dated June 25, 1985 and November 10, 1987, respectively.
4. A. M. Patel, "On-The-Fly Decoder for Multiple Byte Errors," *IBM J. Res. Develop.* **30,** 259–269 (1986).
5. G. D. Forney, *Concatenated Codes*, M.I.T. Press, Cambridge, MA, 1966.
6. P. Elias, "Error-Free Coding," *IEEE Trans. Info. Theory* **IT-4,** 29–37 (1954).
7. A. M. Patel, "Error-Recovery Scheme for the IBM 3850 Mass Storage System," *IBM J. Res. Develop.* **24,** 32–42 (1980).
8. J. K. Wolf, "Adding Two Information Symbols in Certain Nonbinary BCH Codes, and Some Applications," *Bell Syst. Tech. J.* **48,** 2408–2424 (1969).
9. R. T. Chien, "Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Info. Theory* **IT-10,** 357–363 (1964).

Received August 31, 1988; accepted for publication January 11, 1989

Arvind M. Patel *IBM General Products Division, 5600 Cottle Road, San Jose, California 95193.*

Dr. Patel is an IBM Fellow at the GPD Laboratory in San Jose. He is currently involved with the development of advanced signal processing techniques for computer storage products. He received his B.E. from Sardar Vallabh-Bhai Vidyapeeth, India, in 1959, his M.S. from the University of Illinois, Urbana, in 1961, and his Ph.D. from the University of Colorado at Boulder in 1969, all in electrical engineering. Dr. Patel joined IBM at the Poughkeepsie, New York, laboratory in 1962. Since then he has participated in various aspects of magnetic recording technology and product development projects in the Poughkeepsie, Boulder, and San Jose laboratories. His main theoretical interest has been in exploring the area of information theory and coding for computer applications. His work on data encoding and error-correcting codes has won him five Outstanding Invention Awards from IBM in 1972, 1973, 1983, 1985, and 1987, and an Outstanding Technical Paper Award from the American Federation of Information Processing Societies in 1970. Dr. Patel has been elected a Fellow of the Institute of Electrical and Electronics Engineers, with the citation: "For contributions to data encoding/decoding and error correction and their application to magnetic storage devices."