On-the-fly decoder for multiple byte errors

by Arvind M. Patel

Multiple-error-correcting Reed-Solomon or BCH codes in $GF(2^b)$ can be used for correction of multiple burst errors in binary data. However, the relatively long time required for decoding multiple errors has been among the main objections to applying these schemes to high-performance computer products. In this paper, a decoding procedure is developed for on-the-fly correction of multiple symbol (byte) errors in Reed-Solomon or BCH codes. A new decoder architecture expands the concept of Chien search of error locations into computation of error values as well, and creates a synchronous procedure for complete on-the-fly error correction of multiple byte errors. Forney's expression for error values is further simplified, which results in substantial economies in hardware and decoding time. All division operations are eliminated from the computation of the error-locator equation, and only one division operation is required in the computation of error values. The special cases of fewer errors are processed automatically, using the corresponding smaller set of syndromes through a single set of hardware. The resultant decoder implementation is well suited for LSI chip design with pipelined data flow. The implementation is illustrated with an example.

1. Introduction

Many tape and disk storage products in modern computers make use of error-correction coding to obtain a cost-effective design for high reliability and data integrity. In most of these applications, decoding for the error-correction code is a direct add-on penalty to the access time specification for a particular data-storage product. For this reason, an on-the-fly decoder with pipelined data flow is highly desirable.

Future storage products will be required to provide improved reliability and availability in spite of their greater packing densities and delivery rates. A relatively greater number of soft errors, including multiple burst errors, will need on-the-fly correction. Reed-Solomon [1] or BCH [2, 3] codes in Galois field $GF(2^b)$ can be used [4] for correction of burst errors by interleaving codewords. In these codes, each symbol is represented by a binary byte, and an error is a byte error. These codes are very efficient in terms of redundancy for correction of multiple errors; however, the relatively long time required for decoding multiple errors has been among the main objections to applying these schemes to high-performance computer products.

Since the appearance of the original works of Reed and Solomon [1], Bose and Chaudhuri [2], and Hocquenghem [3], the decoding problem has been studied by many. Peterson [5] and Gorenstein and Zierler [6] provided the basic key to the solution of the decoding problem through the concept of the error-locator polynomial. The coefficients of the error-locator polynomial are computed by solving a set of linear equations. Berlekamp [7] and Massey [8] provided an iterative method for computing the coefficients of the error-locator polynomial.

The roots of the error-locator polynomial represent the locations of the symbol in error. Chien [9] suggested a simple mechanized method for searching these roots, using a cyclic trial-and-error procedure. Forney [10] provided further simplifications in computations of the error values in the case of codes with nonbinary symbols. The iterative method [11] of Berlekamp and Massey provided yet another way of computing the error values. There are also direct decoding methods [12, 13] which avoid computing the coefficients of the error-locator polynomial. These methods, however, require more computations.

In this paper, we present decoder equations and architecture in which the location and value of each error are computed in a cyclic order without explicit information regarding the locations of other errors which are yet to be determined. This method expands the "Chien search" function (which normally finds only the error locations in a cyclic manner) into a complete mechanization of the error-correcting procedure. As a result, we can begin delivery of the decoded data symbols, one at a time, in synchrony with each cycle of the Chien search. Thus, access time is not impacted by the time required for the computation of the locations and values of *all* errors.

In this method, the hardware is highly simplified. In particular, the result of Chien search need not be stored, and one set of hardware computes the locations and values of all errors and corrects them at the appropriate cycle *during* the Chien search.

This paper also presents other specific improvements in the design and implementation of the on-the-fly decoder, as listed below:

a. All division operations are eliminated from the computations of the error-locator equation.
b. Lemmas 1, 2, and 3 provide a convenient closed-form expression for error values. This new expression requires a smaller total number of operations and only one division operation in on-the-fly computation of the error values.
c. The special cases of fewer errors are processed through a single set of hardware as a routine procedure.
d. The decoding equations, as well as the decoder design, possess a nested form of architecture which allows processing of fewer syndromes for fewer errors. Thus, the same hardware (LSI chip or chips) can be used in various applications with varying reductions in redundancy.

These improvements, with the new on-the-fly error-correction architecture, make the decoder implementation highly structured and well suited for LSI chip design with pipelined data flow. The decoder design is illustrated with details of this implementation for the case of three byte errors.

2. Error syndromes

In a general Reed-Solomon or BCH code, the codeword consists of n symbols in the Galois field $GF(q)$ which include r check symbols corresponding to the roots $\alpha^a, \alpha^{a+1}, \alpha^{a+2}, \cdots, \alpha^{a+r-1}$ of the generator polynomial, where α is an element of $GF(q)$. For convenience, the integer a is taken to be zero [14]. The r coding relations, then, can be written as

$$\sum_{i=0}^{n-1} \alpha^{ji} B_i = 0 \qquad \text{for } j = 0, 1, 2, \cdots, (r-1),$$

where $B_0, B_1, B_2, \cdots, B_{n-1}$ are the n symbols of the codeword. The corresponding syndromes S_j can be computed from the received codeword as

$$S_j = \sum_{i=0}^{n-1} \alpha^{ji} \hat{B}_i \qquad \text{for } j = 0, 1, 2, \cdots, (r-1), \tag{1}$$

where $\hat{B}_i$ represents the received symbol corresponding to B_i.

Let ν denote the actual number of symbols in error in a given codeword. The error values are $E_i = (\hat{B}_i - B_i)$, where i represents an error-location value from a set of different error locations given by $\{I\} = \{i_1, i_2, \cdots, i_\nu\}$. The relationships between the syndromes and the errors are then given by

$$S_j = \sum_{i \in \{I\}} \alpha^{ji} E_i \qquad \text{for } j = 0, 1, 2, \cdots, (r-1). \tag{2}$$

Any nonzero value of a syndrome indicates the presence of errors. The decoder processes these syndromes to determine the locations and values of the errors. Let t denote the maximum number of errors that can be decoded without ambiguity. A set of $r = 2t$ consecutive syndromes is sufficient to determine the locations of t errors. If the locations are already known, then a set of only t consecutive syndromes is sufficient to determine the values of t erasures (errors with known locations).

3. Equation for error locations

Consider the polynomial with roots at α^i, where $i \in \{I\}$. This is called the error-locator polynomial and is defined as

$$\prod_{i \in \{I\}} (1 - \alpha^{-i} x) = \sum_{m=0}^{\nu} \sigma_m x^m = \sum_{m=0}^{t} \sigma_m x^m. \tag{3}$$

In the case of erasures, the coefficient α^{-i} in each factor of the locator polynomial is known. Thus, the coefficients σ_m can easily be computed. In the case of errors, the coefficients σ_m can be determined from the syndromes. For a given received word, the decoder will proceed to determine σ_m as if there were t roots in the locator polynomial. If the actual number of errors ν is less than t, this will result in σ_m being zero for all $m > \nu$.

Substituting $x = \alpha^i$ in Equation (3), we get

$$\sum_{m=0}^{t} \sigma_m \alpha^{mi} = 0 \qquad \text{for } i \in \{I\}. \tag{4}$$

By using Equations (2) and (4), it is easy to verify that the syndromes S_j and the coefficients σ_m of the error-locator polynomial satisfy the following set of relationships:

$$\sum_{m=0}^{t} \sigma_m S_{m+k} = 0 \qquad \text{for } k = 0, 1, \cdots, (t-1). \tag{5}$$

The set of equations (5) can be rewritten in matrix notation as

$$\begin{bmatrix} S_0 & S_1 & \cdots & S_t \\ S_1 & S_2 & \cdots & S_{t+1} \\ \vdots & \vdots & \vdots & \vdots \\ S_{t-1} & S_t & \cdots & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_{t-1} \\ \sigma_t \end{bmatrix} = 0. \tag{6}$$

Let M denote the t-by-$(t+1)$ syndrome matrix on the left side of Equation (6). Let M_t denote the square matrix obtained by eliminating the last column in matrix M. If M_t is nonsingular, then the above set of equations can be solved, using Cramer's rule, to obtain

$$\frac{\sigma_m}{\sigma_t} = \frac{\Delta_{tm}}{\Delta_{tt}} \qquad \text{for } m = 0, 1, \cdots, (t-1), \tag{7}$$

where Δ_{tt} is the nonzero determinant of matrix M_t, and Δ_{tm} denotes the determinant of the matrix obtained by replacing the mth column in the matrix M_t with the negative of the last column of the matrix M for each $m = 0, 1, \cdots, (t-1)$.

If matrix M_t is singular, that is, Δ_{tt} is zero, then the set of equations (5) is a dependent set. It can be shown that in the case of fewer than t errors, $\Delta_{tt} = \Delta_{tm} = 0$ for all m. Conversely, simultaneous occurrence of $\Delta_{tt} = 0$ and $\Delta_{tm} \neq 0$ indicates more than t errors. Thus, when Δ_{tt} is zero we assume fewer than t errors (and check for $\Delta_{tm} = 0$ for all m). In that case, σ_t is zero. We can delete σ_t and the last row and the last column of the syndrome matrix in Equation (6). The resulting matrix equation corresponds to that for $t-1$ errors. This process is repeated, if necessary, so that the final matrix equation corresponds to that for ν errors and M_ν is nonsingular. Then we need the set of determinants $\Delta_{\nu m}$, where $m = 0, 1, \cdots, \nu$.

It can easily be seen that $\Delta_{\nu m}$ for $\nu = t-1$ is a cofactor of Δ_{tt} corresponding to column $m-1$ and row t in matrix M_t. We can express Δ_{tt} in terms of these cofactors:

$$\Delta_{tt} = \sum_{m=0}^{t-1} S_{t-1+m} \Delta_{(t-1)m}. \tag{8}$$

Thus, the values $\Delta_{\nu m}$ for $\nu = t-1$ need not require separate computations. They are available as byproducts of the computation for Δ_{tt}. In fact, $\Delta_{\nu m}$ for subsequent smaller values of ν are all available as byproducts of the computation for Δ_{tt} through the hierarchial relationships of lower-order cofactors.

Thus, in the case of fewer errors, the decoder finds $\Delta_{tt} = 0$ and automatically backtracks through prior computations to the correct value of ν, and uses the previously computed cofactors $\Delta_{\nu m}$. This is illustrated later through hardware implementation of the case $t = 3$.

To accommodate the special cases of all fewer errors, we replace Equation (7) with a more convenient general form:

$$\frac{\sigma_m}{\sigma_\nu} = \frac{\Delta_m}{\Delta_\nu} \qquad \text{for } m = 0, 1, 2, \cdots, t, \tag{9}$$

where ν is determined from the fact that $\Delta_{mm} = 0$ for all $m > \nu$ and $\Delta_{mm} \neq 0$ for $m = \nu$. Then Δ_m is defined with the new notation as

$$\Delta_m = \begin{cases} \Delta_{mm} & \text{for } m > \nu, \\ \Delta_{\nu m} & \text{for } m \leq \nu. \end{cases} \tag{10}$$

Since $\sigma_0 = 1$, we can determine σ_m for all values of m, using Equation (9). However, we will see that the coefficients σ_m are not needed in the entire decoding process. To this end, we obtain a modified error-locator equation from Equations (4) and (9) as given by

$$\sum_{m=0}^{t} \Delta_m \alpha^{mi} = 0 \qquad \text{for } i \in \{I\}. \tag{11}$$

The error-location values $i \in \{I\}$ are the set of ν unique values of i which satisfy Equation (11).

4. Expression for error values

The error-locator polynomial, as defined by Equation (4), has ν roots corresponding to ν error-location values. Now consider a polynomial which has all roots of the error-locator polynomial except one corresponding to the location value $i = j$. This polynomial is defined as

$$\prod_{\substack{i \in \{I\} \\ i \neq j}} (1 - \alpha^{-i} x) = \sum_{m=0}^{\nu-1} \sigma_{j,m} x^m = \sum_{m=0}^{t-1} \sigma_{j,m} x^m. \tag{12}$$

When the actual number of error locations ν is less than t, the coefficients $\sigma_{j,m}$ are zero for $m = \nu, \cdots, (t-1)$. This is done to allow processing of any value of ν through the same set of hardware.

Substituting $x = \alpha^i$ in Equation (12), we get

$$\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mi} = 0 \qquad \text{for } i \in \{I\},\ i \neq j. \tag{13}$$

Now, taking a hint from Equation (5), we examine a similar expression involving the syndromes and the coefficients $\sigma_{j,m}$ of the new polynomial. Using Equation (2), we substitute for the syndrome S_m and get

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{m=0}^{t-1} \sigma_{j,m} \sum_{i \in \{I\}} \alpha^{mi} E_i. \tag{14}$$

Interchanging the order of summing parameters m and i in Equation (14), we get

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{i \in \{I\}} E_i \sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mi}. \tag{15}$$

Now, using Equations (13) and (15), we obtain

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = E_j \sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj}. \tag{16}$$

Thus, we have an expression for the error values

$$E_j = \frac{\sum_{m=0}^{t-1} \sigma_{j,m} S_m}{\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj}}. \tag{17}$$

This expression for error values is well known [10, 15]. Notice that with a known error-locator polynomial and the error-location value j, explicit values of other error locations are not required in computing the coefficients $\sigma_{j,m}$. Thus, we can eliminate $\sigma_{j,m}$ and reduce (17) further to obtain a more convenient form for on-the-fly processing. To this end, we prove Lemmas 1, 2, and 3 which follow.

In Lemma 1, we obtain a relation which expresses the coefficients $\sigma_{j,m}$ in terms of the known coefficients σ_k of the error-locator polynomial.

Lemma 1

$$\sum_{k=0}^{m} \sigma_k \alpha^{kj} = \sigma_{j,m} \alpha^{mj} \qquad \text{for } 0 \le m < t. \tag{18}$$

Proof From the definition of polynomials in Equations (3) and (12), we have

$$\sum_{m=0}^{t} \sigma_m x^m = (1 - \alpha^{-j} x) \sum_{m=0}^{t-1} \sigma_{j,m} x^m. \tag{19}$$

Comparing the coefficients of each term in the polynomials on the two sides of Equation (19), we obtain

$$\sigma_m = \begin{cases} \sigma_{j,m} - \sigma_{j,m-1} \alpha^{-j} & \text{for } 0 < m < t, \\ \sigma_{j,m} & \text{for } m = 0. \end{cases} \tag{20}$$

Using Equation (20), we can substitute for σ_k and obtain

$$\sum_{k=0}^{m} \sigma_k \alpha^{kj} = \sigma_{j,0} + \sum_{k=1}^{m} [\sigma_{j,k} \alpha^{kj} - \sigma_{j,k-1} \alpha^{(k-1)j}]. \tag{21}$$

On eliminating the canceling terms from Equation (21), we get

$$\sum_{k=0}^{m} \sigma_k \alpha^{kj} = \sigma_{j,m} \alpha^{mj} \qquad \text{for } 0 \le m < t. \tag{22}$$

This completes the proof of Lemma 1.

Next, we rewrite the denominator of the expression in Equation (17), using the result of Lemma 1.

Lemma 2

$$\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj} = \sum_{k=0}^{t} -k \sigma_k \alpha^{kj}. \tag{23}$$

Proof Using Lemma 1, we first obtain

$$\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj} = \sum_{m=0}^{t-1} \sum_{k=0}^{m} \sigma_k \alpha^{kj}. \tag{24}$$

Collecting all terms with the same values of k in Equation (24), we then get

$$\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj} = \sum_{k=0}^{t-1} (t - k) \sigma_k \alpha^{kj}. \tag{25}$$

Using Equation (4), we can rewrite Equation (25) as

$$\sum_{m=0}^{t-1} \sigma_{j,m} \alpha^{mj} = -t \sigma_t \alpha^{tj} + \sum_{k=0}^{t-1} -k \sigma_k \alpha^{kj}, \tag{26}$$

which is the same as Equation (23). This completes the proof of Lemma 2.

We again use the result of Lemma 1 and obtain a more convenient expression for the numerator in Equation (17) in the following lemma.

Lemma 3

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{m=0}^{t-1} \xi_m \alpha^{-mj}, \tag{27}$$

where

$$\xi_m = \sum_{i=0}^{t-1-m} \sigma_i S_{m+i}. \tag{28}$$

Proof Using Lemma 1, we can express $\sigma_{j,m}$ in terms of σ_k, obtaining

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{m=0}^{t-1} S_m \alpha^{-mj} \sum_{k=0}^{m} \sigma_k \alpha^{kj}. \tag{29}$$

Substituting $(m - h)$ for k on the right-hand side of Equation (29), we get

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{m=0}^{t-1} S_m \sum_{h=0}^{m} \sigma_{m-h} \alpha^{-hj}. \tag{30}$$

Now, interchanging the order of summing parameters m and h in Equation (30) gives

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{h=0}^{t-1} \alpha^{-hj} \sum_{m=h}^{t-1} \sigma_{m-h} S_m. \tag{31}$$

Substituting $(k + h)$ for m in Equation (31), we have

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{h=0}^{t-1} \alpha^{-hj} \sum_{k=0}^{t-1-h} \sigma_k S_{k+h}. \tag{32}$$

This completes the proof of Lemma 3.

The computation of ξ_m for $m = 0$ to $(t - 1)$ in Equation (28) of Lemma 3 requires $t(t + 1)/2$ multiplications of the type $\sigma_i S_{m+i}$. The number of multiplications can be reduced in case of ξ_m for $m < (t - 1)/2$ by using Equation (5), which yields the following alternate expression:

$$\xi_m = \sum_{i=t-m}^{t} -\sigma_i S_{m+i}. \tag{33}$$

Equation (28) requires the syndromes $S_0, S_1, \cdots, S_{t-1}$, and Equation (33) requires the syndromes $S_t, S_{t+1}, \cdots, S_{2t-1}$. When we use Equation (28) for $m \ge (t - 1)/2$ and Equation

(33) for $m < (t-1)/2$, the number of multiplications required is the minimum and is equal to the integer closest to $(t+1)^2/4$.

Note that Equation (28) does not satisfy the requirement that the same hardware process the case of fewer errors with fewer syndromes, since the high-order terms do not vanish automatically. Furthermore, Equation (33) cannot be used if the decoder is designed for t erasures where the syndromes $S_t, S_{t+1}, \cdots, S_{2t-1}$ may not be available. The following corollary of Lemma 3 provides an alternate equivalent expression which removes the difficulty mentioned above.

Corollary

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \alpha^{-\mu j} \sum_{m=0}^{t-1} \beta_m \alpha^{mj}, \tag{34}$$

where $0 \le \mu < t$ and the coefficients β_m are given by

$$\beta_m = \begin{cases} \displaystyle\sum_{k=0}^{m} \sigma_k S_{k-m+\mu} & \text{for } m \le \mu, \\ \displaystyle\sum_{k=m+1}^{t} -\sigma_k S_{k-m+\mu} & \text{for } m > \mu. \end{cases} \tag{35}$$

Proof Using the result from Equation (4), we can rewrite the terms with $m > \mu$ in the right-hand side of Equation (29) as follows:

$$\sum_{m=0}^{t-1} \sigma_{j,m} S_m = \sum_{m=0}^{\mu} S_m \alpha^{-mj} \sum_{k=0}^{m} \sigma_k \alpha^{kj} + \sum_{m=\mu+1}^{t-1} S_m \alpha^{-mj} \sum_{k=m+1}^{t} -\sigma_k \alpha^{kj}. \tag{36}$$

The proof of the corollary, then, follows steps similar to those in Equations (30)–(32) of Lemma 3. First, substitute $(m-h)$ for k, and rewrite the right-hand side of (36) with m and h as summing parameters. Then, interchange the order of summing parameters m and h. Next, substitute $(k+h)$ for m, and rewrite the right-hand side with h and k as summing parameters. Finally, substitute $(\mu - m)$ for h. This completes the proof of the corollary.

The important feature of the expression in the corollary is that the high-order terms vanish automatically in the case of fewer errors, and the resultant computation involves a reduced set of syndromes $S_0, S_1, \cdots, S_{\nu-1}$, where ν is the actual number of errors or erasures, and $\mu < \nu \le t$.

The number of multiplications in computing β_m of Equation (35) depends on the choice of μ. The minimum number of multiplications is, again, the integer closest to $(t+1)^2/4$ when $\mu = \underline{\mu}$, where $\underline{\mu}$ is the largest integer under $(t-1)/2$.

The same hardware can also be used in applications with a smaller set of available syndromes $S_0, S_1, \cdots, S_{t'-1}$ and up to t' errors or erasures, provided that $\mu < t' \le t$. From this point of view, a lower value of μ is desirable. The choice of $\mu = 0$ offers the maximum flexibility in terms of the applicability of the same hardware for processing fewer errors with fewer syndromes. The number of multiplications required in computing β_m in Equation (35) with $\mu = 0$ is $(t^2 - t + 2)/2$, which is not the lowest value possible but is still lower than the maximum value required in Lemma 3.

In view of the above observations, we use the expression in the corollary of Lemma 3 for decoder implementation. For large values of t, it is advisable to use $\mu = \underline{\mu}$ for greatest economy in hardware. In the case of small values of t, $\mu = 0$ provides greater flexibility in adapting other smaller values of t later without modifying the already fabricated hardware.

Now we can rewrite Equation (17), using the results of Lemma 2 and the corollary of Lemma 3. As a result, any error value E_i can be expressed as

$$E_i = \alpha^{-\mu i} \frac{\displaystyle\sum_{m=0}^{t-1} \beta_m \alpha^{mi}}{\displaystyle\sum_{m=0}^{t} -m\sigma_m \alpha^{mi}}, \tag{37}$$

where

$$\beta_m = \begin{cases} \displaystyle\sum_{k=0}^{m} \sigma_k S_{k-m+\mu} & \text{for } m \le \mu, \\ \displaystyle\sum_{k=m+1}^{t} -\sigma_k S_{k-m+\mu} & \text{for } m > \mu. \end{cases} \tag{38}$$

In view of Equation (9), the coefficients σ_m can be eliminated to obtain error values in terms of Δ_m:

$$E_i = \alpha^{-\mu i} \frac{\displaystyle\sum_{m=0}^{t-1} \Phi_m \alpha^{mi}}{\displaystyle\sum_{m=0}^{t} -m\Delta_m \alpha^{mi}}, \tag{39}$$

where

$$\Phi_m = \begin{cases} \displaystyle\sum_{k=0}^{m} \Delta_k S_{k-m+\mu} & \text{for } m \le \mu, \\ \displaystyle\sum_{k=m+1}^{t} -\Delta_k S_{k-m+\mu} & \text{for } m > \mu. \end{cases} \tag{40}$$

In the case of the binary base field, the denominator of Equation (39) simplifies further since the terms with even values of m ($m = 0 \bmod 2$) vanish. The resultant expression for E_i for the binary base field is

$$E_i = \alpha^{-\mu i} \frac{\displaystyle\sum_{m=0}^{t-1} \Phi_m \alpha^{mi}}{\displaystyle\sum_{\substack{m=0 \\ m \text{ odd}}}^{t} \Delta_m \alpha^{mi}}, \tag{41}$$

where

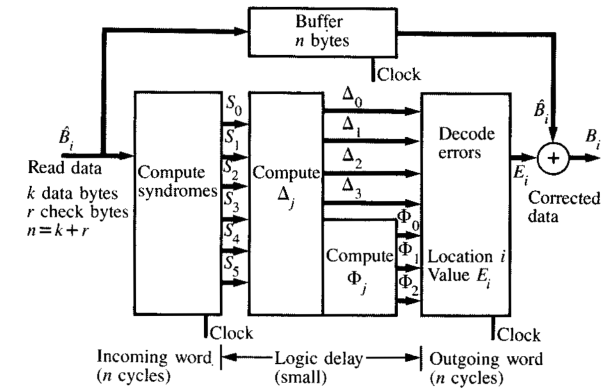

Figure 1

Decoder block diagram.

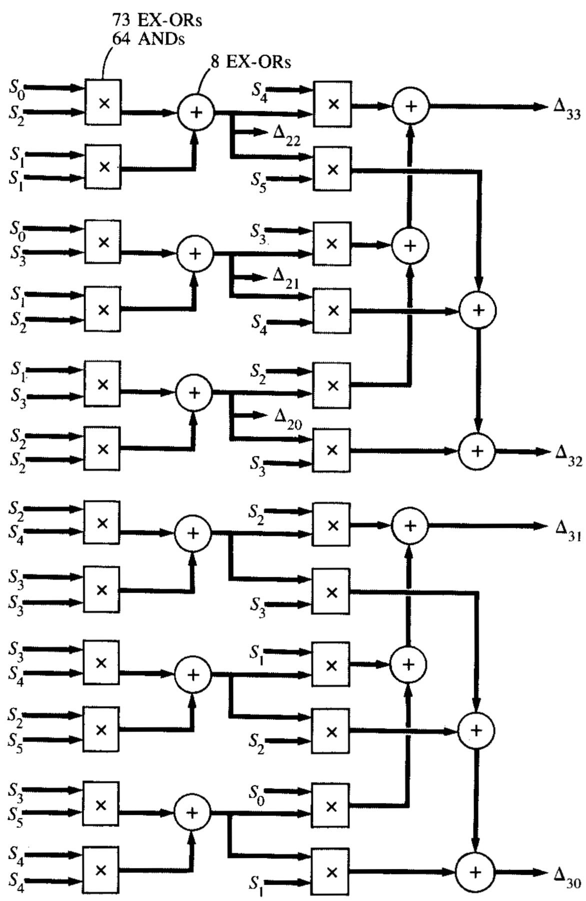

Figure 2

Coefficients of the error-locator polynomial.

$$\Phi_m = \begin{cases} \sum_{k=0}^{m} \Delta_k S_{k-m+\mu} & \text{for } m \le \mu, \\ \sum_{k=m+1}^{t} \Delta_k S_{k-m+\mu} & \text{for } m > \mu. \end{cases} \qquad (42)$$

Note that the computation for the denominator in Equation (41) is already available as the sum of all odd (or even) terms in the computations of Equation (11). For each value of i, the numerator can be computed and multiplied by the inverse of the denominator in synchrony with the search for error locations. The resultant E_i is used for correcting the outgoing ith symbol $\hat{B}_i$ whenever the error-locator equation (11) is satisfied.

5. Decoder implementation

Figure 1 is a block diagram of the on-the-fly decoder. The decoding process is continuous in an uninterrupted stream of data arriving in the form of a chain of n-symbol codewords. The decoder computes syndromes for the incoming codeword as it decodes and corrects errors in the (previously received) outgoing codeword.

Each clock cycle corresponds to an input of one data symbol of the incoming codeword concurrent with an output of one corrected data symbol of the outgoing codeword. A buffer holds at least n symbols of the uncorrected data between the incoming and outgoing symbols.

We use the three-error-correcting Reed-Solomon code in $GF(2^8)$ as an example of special interest for application in computer products. Six check symbols correspond to the six roots α^0, α^1, α^2, α^3, α^4, α^5 of the generator polynomial. The corresponding syndromes are denoted by S_0, S_1, S_2, S_3, S_4, and S_5, respectively.

These syndromes are computed from the received word in the conventional manner in accordance with Equation (1). The implementation for this step is well known, using EX-OR circuits and shift registers. Here, we present the hardware implementation for the remaining steps of the decoding procedure, using the equations developed in the previous sections.

- *Computation of coefficients*

For the three-error case, the matrix equation (6) for the coefficients σ_m of the error-locator polynomial can be written as

$$\begin{bmatrix} S_0 & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \end{bmatrix} = \sigma_3 \begin{bmatrix} S_3 \\ S_4 \\ S_5 \end{bmatrix}. \qquad (43)$$

The corresponding determinants Δ_{33}, Δ_{32}, Δ_{31}, Δ_{30} of Equation (7) are given by the following expressions, where $\oplus$ denotes the modulo-2 vector sum of 8-bit bytes:

$$\Delta_{33} = S_2(S_1S_3 \oplus S_2S_2) \oplus S_3(S_0S_3 \oplus S_1S_2) \oplus S_4(S_1S_1 \oplus S_2S_0), \quad (44)$$

$$\Delta_{32} = S_3(S_1S_3 \oplus S_2S_2) \oplus S_4(S_0S_3 \oplus S_1S_2) \oplus S_5(S_1S_1 \oplus S_2S_0), \quad (45)$$

$$\Delta_{31} = S_0(S_4S_4 \oplus S_3S_5) \oplus S_1(S_3S_4 \oplus S_2S_5) \oplus S_2(S_3S_3 \oplus S_2S_4), \quad (46)$$

$$\Delta_{30} = S_1(S_4S_4 \oplus S_3S_5) \oplus S_2(S_3S_4 \oplus S_2S_5) \oplus S_3(S_3S_3 \oplus S_2S_4). \quad (47)$$

These four expressions can be implemented through combinational logic circuits as shown in **Figure 2**. These circuits require 24 product operations and 14 sum operations in $GF(2^8)$. A typical product operation in $GF(2^8)$ requires, at the most, 73 EX-OR gates and 64 AND gates, as shown in Appendix A. A sum operation in $GF(2^8)$ is a modulo-2 vector sum which requires 8 EX-OR gates. The hardware of Figure 2 can be further reduced, if desired, by time-sharing some of the repetitive functions.

The determinants Δ_{22}, Δ_{21}, and Δ_{20} for the two-error case are cofactors in the expression (44) for Δ_{33}, as was expressed in Equation (8) for the general case of t errors. These cofactors are

$$\Delta_{22} = S_1S_1 \oplus S_2S_0, \quad (48)$$

$$\Delta_{21} = S_0S_3 \oplus S_1S_2, \quad (49)$$

$$\Delta_{20} = S_1S_3 \oplus S_2S_2. \quad (50)$$

In Figure 2, the computations for Δ_{22}, Δ_{21}, and Δ_{20} are shown as the interim by-products within the computations for Δ_{33}. Similarly, Δ_{11} and Δ_{10}, which are readily available syndromes, are cofactors in the expression (48) for Δ_{22}:

$$\Delta_{11} = S_0, \quad (51)$$

$$\Delta_{10} = S_1. \quad (52)$$

Figure 3 shows the hardware implementation of Equation (10), where the decoder identifies the correct number (ν) of errors and selects appropriate values for Δ_m. When Δ_{33} is nonzero, the parameters Δ_3, Δ_2, Δ_1, and Δ_0 take the values Δ_{33}, Δ_{32}, Δ_{31}, and Δ_{30}, respectively. When Δ_{33} is zero, then Δ_2, Δ_1, and Δ_0 take the values Δ_{22}, Δ_{21}, and Δ_{20}, respectively, which corresponds to the syndrome equations for two symbol errors. Similarly, if Δ_{22} is also zero, then Δ_1 and Δ_0 take the values Δ_{11} and Δ_{10}, respectively, which corresponds to the syndrome equations for one symbol error. Thus, Figure 3 produces the appropriate values of the coefficients Δ_3, Δ_2, Δ_1, and Δ_0 for the error-locator equation (11), rewritten for the case of $t = 3$ as

$$\Delta_3\alpha^{3i} \oplus \Delta_2\alpha^{2i} \oplus \Delta_1\alpha^{i} \oplus \Delta_0 = 0 \qquad \text{for } i \in \{I\}. \quad (53)$$

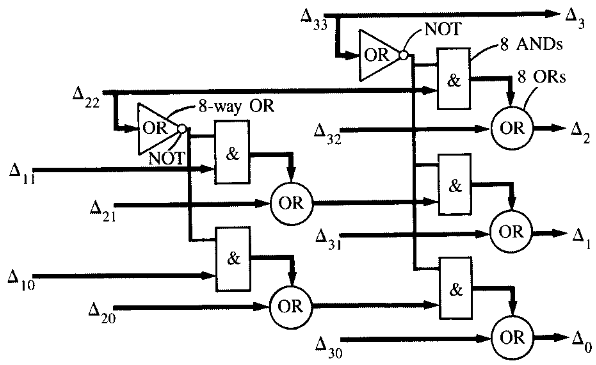

Figure 3

Selection of coefficients for correct number of errors.

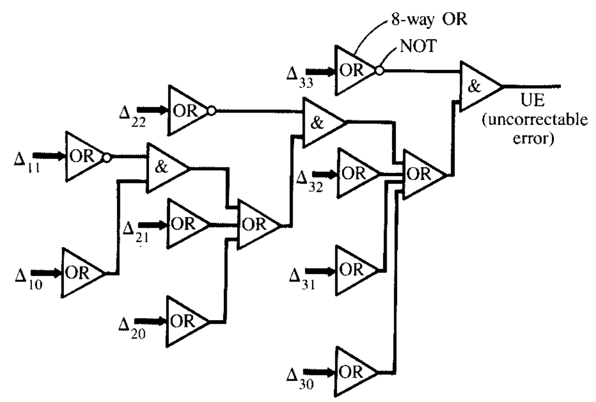

Figure 4

Check for consistency of coefficients.

Figure 4 provides a check for consistency of the coefficients in the case of fewer than three errors. In particular, when $\Delta_{33} = 0$, we must have $\Delta_{32} = \Delta_{31} = \Delta_{30} = 0$. Also, when $\Delta_{33} = \Delta_{22} = 0$ we must have $\Delta_{21} = \Delta_{20} = 0$, and when $\Delta_{33} = \Delta_{22} = \Delta_{11} = 0$ we must have $\Delta_{10} = 0$. Violation of any of these conditions implies the presence of more than three errors, resulting in an uncorrectable-error (UE) signal at the output of the circuit in Figure 4. With some additional hardware (not shown in Figure 4), we can obtain ν as a two-digit binary number a_1a_0. This number is the largest value of m for which $\Delta_{mm} \neq 0$ ($1 \leq m \leq 3$), and is given by the following logic functions:

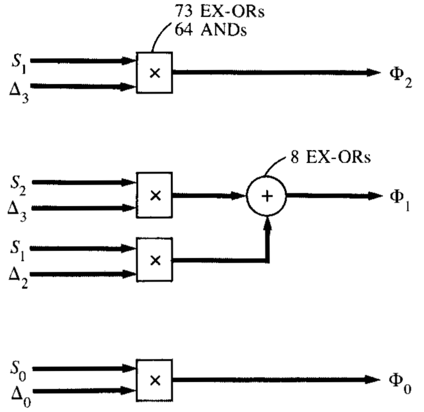

Figure 5

Coefficients for the error-value expression.

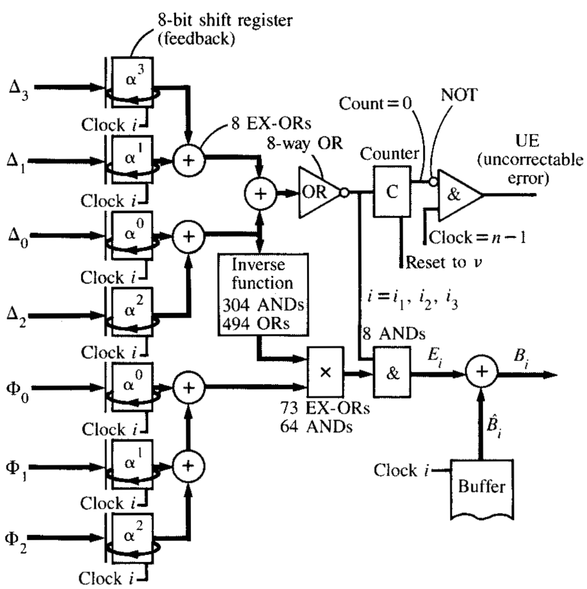

Figure 6

Cyclic decoder circuit. The shift register for Φ_m ($m = 0, 1, 2$) has the multiplier α^{m-a} when a is nonzero (see [14]).

$a_1 = (\Delta_{33} \neq 0)$ OR $(\Delta_{22} \neq 0)$,

$a_0 = (\Delta_{33} \neq 0)$ OR $[(\Delta_{22} = 0)$ AND $(\Delta_{11} \neq 0)]$.

This value of ν will be used by the decoder in Figures 6 and 7, shown later.

Next, we show the hardware implementation for obtaining the values of coefficients Φ_m in the numerator of the error-value equation (41). For the case of three errors, Equation (42) with $\mu = 0$ can be rewritten for Φ_0, Φ_1, and Φ_2 as follows:

$$\Phi_0 = \Delta_0 S_0, \tag{54}$$

$$\Phi_1 = \Delta_2 S_1 \oplus \Delta_3 S_2, \tag{55}$$

$$\Phi_2 = \Delta_3 S_1. \tag{56}$$

Figure 5 shows the implementation of Equations (54), (55), and (56), which requires four product operations and one sum operation in $GF(2^8)$. The error-value equation (41), for the case of three errors, can be rewritten as

$$E_i = \frac{\Phi_2\alpha^{2i} \oplus \Phi_1\alpha^{i} \oplus \Phi_0}{\Delta_3\alpha^{3i} \oplus \Delta_1\alpha^{i}}. \tag{57}$$

- *On-the-fly error correction*

Figure 6 shows the mechanized shift-register circuits for determining error locations and error values in accordance with Equations (53) and (57), respectively. The computed values of the coefficients Δ_3, Δ_2, Δ_1, Δ_0 and Φ_2, Φ_1, Φ_0 are entered into appropriate shift registers at clock zero. Each clock cycle generates a shifting operation of these registers. A shifting operation multiplies the contents of each register by a specific constant, namely α^3, α^2, and α in the case of the registers for Δ_3, Δ_2, and Δ_1, respectively; and α^2 and α in the case of the registers for Φ_2 and Φ_1, respectively. Multiplication by a constant requires a small number of EX-OR gates, as explained in Appendix A.

At the ith clock cycle ($0 \leq i < n$), the upper set of summing circuits in Figure 6 at the output of the shift registers are presented with all the terms of Equation (53). If the sum is zero, then Equation (53) is satisfied and we have captured the error location. Similarly, at the ith clock cycle, the lower set of summing circuits at the output of the shift registers are presented with all the terms of the numerator in Equation (57). The denominator for (57) is already available from the upper set of summing circuits.

Subsequent networks for an inverse operation and then a product operation compute the error value E_i for each i in accordance with Equation (57). The algebraic inverse in $GF(2^8)$ can be obtained through combinational logic, which maps each 8-digit binary sequence into its inverse—a specific 8-digit binary sequence, as shown in Appendix B. This requires, at the most, 304 AND gates and 494 OR gates.

When the error location is captured, the outgoing word symbol $\hat{B}_i$ is modified by E_i through the output sum network. For all other values of i, the computed value of E_i is ignored. When all bytes B_0 through B_{n-1} of the codeword are delivered (at the final clock cycle $n - 1$), if ν error locations were not captured, then the errors exceed the correction capability of the decoder. This condition is detected by means of a counter which counts down from a preset value ν. If the count does not reach zero at the final clock cycle, then the decoder has not corrected the errors properly. The decoder indicates this condition by giving an uncorrectable-error (UE) signal.

- *Delivery order of corrected bytes*

The corrected bytes in the decoder of Figure 6 are delivered in the order $B_0, B_1, B_2, \cdots, B_{n-1}$. This is the reverse order compared to that in the encoding operation, since the check bytes correspond to the low-order positions. The reversal can easily be removed by introducing a reversal relationship between clock-cycle count j and byte-location number i. We substitute $n - j$ for i in the decoding equations (53) and (57) and rewrite them as

$$(\Delta_3\alpha^{3n})\alpha^{-3j} \oplus (\Delta_2\alpha^{2n})\alpha^{-2j} \oplus (\Delta_1\alpha^{n})\alpha^{-j} \oplus \Delta_0 = 0, \tag{58}$$

$$E_{(n-j)} = \frac{(\Phi_2\alpha^{2n})\alpha^{-2j} \oplus (\Phi_1\alpha^{n})\alpha^{-j} \oplus \Phi_0}{(\Delta_3\alpha^{3n})\alpha^{-3j} \oplus (\Delta_1\alpha^{n})\alpha^{-j}}. \tag{59}$$

In these equations, j represents the clock-cycle count, where $j = 1$ to n, successively, correspond to the byte-position values $i = (n - 1)$ to 0. This provides delivery of bytes in the order $B_{n-1}, \cdots, B_1, B_0$, which is the same order as that in the encoding process.

To accomplish the modifications mentioned above, the following changes are made in the decoder hardware of Figure 6: (1) The shift-register multipliers α^3, α^2, and α are replaced by α^{-3}, α^{-2}, and α^{-1}, respectively; (2) The coefficients Δ_3, Δ_2, and Δ_1 are premultiplied by α^{3n}, α^{2n}, and α^{n}, respectively, and the coefficients Φ_2 and Φ_1 are premultiplied by α^{2n} and α^{n}, respectively.

In the case of shortened code, the premultiplication circuits depend on the value of n, and each circuit requires a small number of EX-OR gates. In the case of full-length code, α^{mn} is unity for all values of m; hence these premultiplication circuits are not needed. The decoder, with the two modifications discussed above, appears in **Figure 7**.

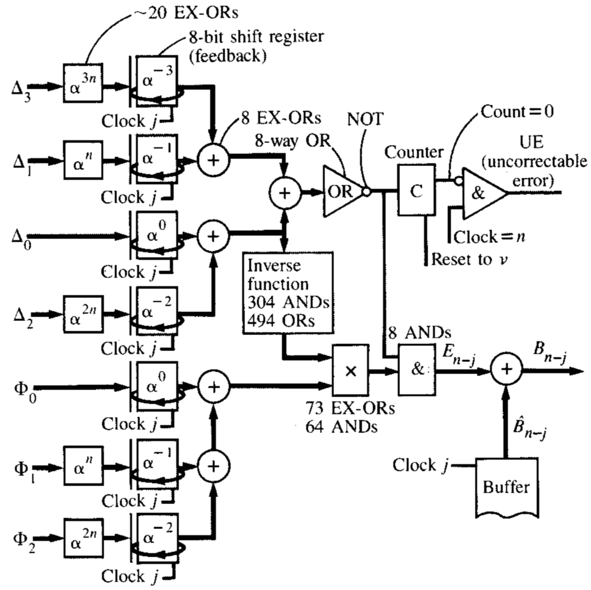

Figure 7

Modified cyclic decoder circuit. The shift register for Φ_m ($m=0$, 1, 2) has the multiplier α^{a-m} and a premultiplier $\alpha^{(m-a)n}$ when a is nonzero (see [14]).

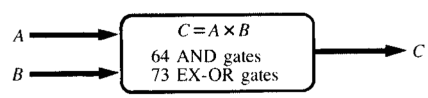

Figure 8

Product function.

6. Conclusions

A decoding procedure is developed for on-the-fly correction of multisymbol errors in Reed-Solomon or BCH codes in nonbinary or extended binary fields. In particular, the decoding equations are formulated in a manner which expands the concept of Chien search of error locations into a search for error values as well, and creates a synchronous procedure for complete on-the-fly correction of multisymbol errors.

Lemmas 1, 2, and 3 and the corollary are new results that provide a more convenient form of the error-value expression for on-the-fly decoding. This expression is key to substantial economies in hardware and decoding time. All division operations are eliminated from the computation of the error-locator equation, and only one division operation is required in the computation of error values.

Further, the decoding equations are organized such that the computations for the special cases of fewer errors than the maximum are realized as byproducts of the main computations. Also, the same decoding hardware can be used for search and correction of errors in all cases of fewer errors, including applications where correspondingly fewer syndromes are available. The details of the decoding hardware are given for the case of the three-byte-correcting Reed-Solomon code.

Appendix A—Product function in $GF(2^8)$

Figure 8 represents the estimated hardware for implementing the product function in $GF(2^8)$. A, B, and C are elements of $GF(2^8)$ and are represented by 8-bit binary vectors, where $C = A \times B$:

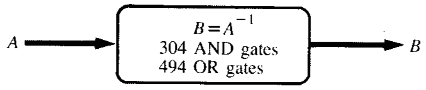

Figure 9

Inverse function.

$$A = [a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0],$$
$$B = [b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0],$$
$$C = [c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0].$$

The product $A \times B$ is obtained through a two-step process. First, we compute the coefficients f_i of the product polynomial F, where $F = A \times B$ modulo 2. Computation of the coefficients $f_i (i = 0, \cdots, 14)$ requires 64 AND gates and 49 EX-OR gates:

$$\begin{aligned}
f_0 &= a_0 b_0, \\
f_1 &= a_0 b_1 \oplus a_1 b_0, \\
f_2 &= a_0 b_2 \oplus a_1 b_1 \oplus a_2 b_0, \\
f_3 &= a_0 b_3 \oplus a_1 b_2 \oplus a_2 b_1 \oplus a_3 b_0, \\
&\vdots \\
f_7 &= a_0 b_7 \oplus a_1 b_6 \oplus a_2 b_5 \oplus \cdots \oplus a_6 b_1 \oplus a_7 b_0, \\
f_8 &= a_1 b_7 \oplus a_2 b_6 \oplus a_3 b_5 \oplus \cdots \oplus a_7 b_1, \\
&\vdots \\
f_{13} &= a_6 b_7 \oplus a_7 b_6, \\
f_{14} &= a_7 b_7.
\end{aligned}$$

Next, we reduce the polynomial F modulo $p(x)$, where $p(x)$ is a primitive binary polynomial of degree 8. We use $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. The reduction of f_i modulo $p(x)$ requires, at the most, 24 EX-OR gates:

$$C_0 = f_0 \oplus f_8 \oplus f_{12} \oplus f_{13} \oplus f_{14},$$
$$C_1 = f_1 \oplus f_9 \oplus f_{13} \oplus f_{14},$$
$$C_2 = f_2 \oplus f_8 \oplus f_{10} \oplus f_{12} \oplus f_{13},$$
$$C_3 = f_3 \oplus f_8 \oplus f_9 \oplus f_{11} \oplus f_{12},$$
$$C_4 = f_4 \oplus f_8 \oplus f_9 \oplus f_{10} \oplus f_{14},$$
$$C_5 = f_5 \oplus f_9 \oplus f_{10} \oplus f_{11},$$
$$C_6 = f_6 \oplus f_{10} \oplus f_{11} \oplus f_{12},$$
$$C_7 = f_7 \oplus f_{11} \oplus f_{12} \oplus f_{13}.$$

The logic for the entire product function requires one level of AND circuits and five levels of EX-OR circuits, which in turn require 64 AND gates and a maximum of 73 EX-OR gates. Note that when one of the multiplicands, say A, is a known constant, then the expression for each component C_i of C will reduce to EX-OR of selected components of the second multiplicand, namely B. The resultant product function requires no AND gates and only a small number of EX-OR gates (maximum 25), depending on the constant A.

Appendix B—Inverse function in $GF(2^8)$

Figure 9 represents the estimated hardware for the inverse function in $GF(2^8)$. A and B are elements of $GF(2^8)$ and are represented by 8-bit binary vectors, where $B = A^{-1}$.

Figure 10 lists 255 nonzero elements of $GF(2^8)$, each with the corresponding inverse element. These elements were generated by the following primitive polynomial:

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1.$$

The elements are represented by 8-digit binary vectors which, in polynomial notation, have the coefficient for the high-order term on the left. The inverse function of this table can be implemented through combinational logic, using the conventional 8-bit encode-decode function. This requires three levels of AND circuits and seven levels of OR circuits, which in turn require a maximum of 304 AND gates and 494 OR gates.

References and note

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. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres* (Paris) **2**, 147–156 (1959).
4. A. M. Patel, "Error-Recovery Scheme for the IBM 3850 Mass Storage System," *IBM J. Res. Develop.* **24**, 32–42 (1980).
5. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IEEE Trans. Info. Theory* **IT-6**, 459–470 (1960).
6. D. C. Gorenstein and N. Zierler, "A Class of Error-Correcting Codes in p^m Symbols," *J. Soc. Indust. Appl. Math.* **9**, 207–214 (1961).
7. E. R. Berlekamp, "On Decoding Binary Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Info. Theory* **IT-11**, 577–579 (1965).
8. J. L. Massey, "Step-by-Step Decoding of the Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Info. Theory* **IT-11**, 580–585 (1965).
9. R. T. Chien, "Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Info. Theory* **IT-10**, 357–363 (1964).
10. G. D. Forney, Jr., "On Decoding BCH Codes," *IEEE Trans. Info. Theory* **IT-11**, 549–557 (1965).
11. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968.
12. Richard E. Blahut, "A Universal Reed-Solomon Decoder," *IBM J. Res. Develop.* **28**, 150–158 (1984).
13. T. Horiguchi and Y. Sato, "A Decoding Method for Reed-Solomon Codes Over $GF(2^m)$," *Trans. IECE* (Jpn.), pp. 97–98 (1983).
14. A scale-up factor α^a in all the roots of the generator polynomial results in a corresponding scale-up factor α^{ai} attached to each error value E_i. [See Equation (2).] The effect of this on the syndrome decoder is that each term of the numerator in the expression for error values [such as Equation (57)] will have a scale-down factor α^{-ai}.
15. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes* (2nd ed.), MIT Press, Cambridge, MA, 1972.

S	S^{-1}	S	S^{-1}	S	S^{-1}	S	S^{-1}	S	S^{-1}
00000001	00000001	00110100	10100100	01100111	01001101	10011010	10111101	11001101	01111101
00000010	10001110	00110101	11000011	01101000	01010010	10011011	10010100	11001110	10101000
00000011	11110100	00110110	01000000	01101001	10001101	10011100	10101100	11001111	00111010
00000100	01000111	00110111	01011110	01101010	11101111	10011101	00001001	11010000	00101001
00000101	10100111	00111000	01010000	01101011	10110011	10011110	11000111	11010001	01110001
00000110	01111010	00111001	00100010	01101100	00100000	10011111	10100010	11010010	11001000
00000111	10111010	00111010	11001111	01101101	11101100	10100000	00011100	11010011	11110110
00001000	10101101	00111011	10101001	01101110	00101111	10100001	10000010	11010100	11111001
00001001	10011101	00111100	10101011	01101111	00110010	10100010	10011111	11010101	01000011
00001010	11011101	00111101	00001100	01110000	00101000	10100011	11000110	11010110	11010111
00001011	10011000	00111110	00010101	01110001	11010001	10100100	00110100	11010111	11010110
00001100	00111101	00111111	11100001	01110010	00010001	10100101	11000010	11011000	00010000
00001101	10101010	01000000	00110110	01110011	11011001	10100110	01000110	11011001	01110011
00001110	01011101	01000001	01011111	01110100	11101001	10100111	00000101	11011010	01110110
00001111	10010110	01000010	11111000	01110101	11111011	10101000	11001110	11011011	01111000
00010000	11011000	01000011	11010101	01110110	11011010	10101001	00111011	11011100	10011001
00010001	01110010	01000100	10010010	01110111	01111001	10101010	00001101	11011101	00001010
00010010	11000000	01000101	01001110	01111000	11011011	10101011	00111100	11011110	00011001
00010011	01011000	01000110	10100110	01111001	01110111	10101100	10011100	11011111	10010001
00010100	11100000	01000111	00000100	01111010	00000110	10101101	00001000	11100000	00010100
00010101	00111110	01001000	00110000	01111011	10111011	10101110	10111110	11100001	00111111
00010110	01001100	01001001	10001000	01111100	10000100	10101111	10110111	11100010	11100110
00010111	01100110	01001010	00101011	01111101	11001101	10110000	10000111	11100011	11110000
00011000	10010000	01001011	00011110	01111110	11111110	10110001	11100101	11100100	10000110
00011001	11011110	01001100	00010110	01111111	11111100	10110010	11101110	11100101	10110001
00011010	01010101	01001101	01100111	10000000	00011011	10110011	01101011	11100110	11100010
00011011	10000000	01001110	01000101	10000001	01010100	10110100	11101011	11100111	11110001
00011100	10100000	01001111	10010011	10000010	10100001	10110101	11110010	11101000	11111010
00011101	10000011	01010000	00111000	10000011	00011101	10110110	10111111	11101001	01110100
00011110	01001011	01010001	00100011	10000100	01111100	10110111	10101111	11101010	11110011
00011111	00101010	01010010	01101000	10000101	11001100	10111000	11000101	11101011	10110100
00100000	01101100	01010011	10001100	10000110	11100100	10111001	01100100	11101100	01101101
00100001	11101101	01010100	10000001	10000111	10110000	10111010	00000111	11101101	00100001
00100010	00111001	01010101	00011010	10001000	01001001	10111011	01111011	11101110	10110010
00100011	01010001	01010110	00100101	10001001	00110001	10111100	10010101	11101111	01101010
00100100	01100000	01010111	01100001	10001010	00100111	10111101	10011010	11110000	11100011
00100101	01010110	01011000	00010011	10001011	00101101	10111110	10101110	11110001	11100111
00100110	00101100	01011001	11000001	10001100	01010011	10111111	10110110	11110010	10110101
00100111	10001010	01011010	11001011	10001101	01101001	11000000	00010010	11110011	11101010
00101000	01110000	01011011	01100011	10001110	00000010	11000001	01011001	11110100	00000011
00101001	11010000	01011100	10010111	10001111	11110101	11000010	10100101	11110101	10001111
00101010	00011111	01011101	00001110	10010000	00011000	11000011	00110101	11110110	11010011
00101011	01001010	01011110	00110111	10010001	11011111	11000100	01100101	11110111	11001001
00101100	00100110	01011111	01000001	10010010	01000100	11000101	10111000	11111000	01000010
00101101	10001011	01100000	00100100	10010011	01001111	11000110	10100011	11111001	11010100
00101110	00110011	01100001	01010111	10010100	10011011	11000111	10011110	11111010	11101000
00101111	01101110	01100010	11001010	10010101	10111100	11001000	11010010	11111011	01110101
00110000	01001000	01100011	01011011	10010110	00001111	11001001	11110111	11111100	01111111
00110001	10001001	01100100	10111001	10010111	01011100	11001010	01100010	11111101	11111111
00110010	01101111	01100101	11000100	10011000	00001011	11001011	01011010	11111110	01111110
00110011	00101110	01100110	00010111	10011001	11011100	11001100	10000101	11111111	11111101

Figure 10

Inverse of field elements in $GF(2^8)$. S is an element of $GF(2^8)$. S as a polynomial has the high-order term on the left and is defined by a residue class, modulo $p(x)$, where $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

Received May 22, 1985; accepted for publication November 25, 1985

Arvind M. Patel *IBM General Products Division, 5600 Cottle Road, San Jose, California 95193.* Dr. Patel is a senior technical staff member in the Magnetic Recording Institute, a GPD/Research organization at San Jose. He is currently involved with the development of computer storage products using magnetic recording technology. 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 worked on 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 four Outstanding Invention Awards from IBM in 1972, 1973, 1983, and 1985, 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."