Richard E. Blahut

A Universal Reed-Solomon Decoder

Two architectures for universal Reed-Solomon decoders are given. These decoders, called time-domain decoders, work directly on the raw data word as received without the usual syndrome calculation or power-sum-symmetric functions. Up to the limitations of the working registers, the decoders can decode any Reed-Solomon codeword or BCH codeword in the presence of both errors and erasures. Provision is also made for decoding extended codes and shortened codes.

Introduction

Reed-Solomon codes [1] and other Bose-Chaudhuri-Hocquenghem (BCH) [2, 3] codes have come into widespread use both in communication systems and in magnetic recording systems. Every particular application has its own distinct requirements usually satisfied by its own individual hardware design. It may be more efficient, instead, to develop a single universal decoder on a very-large-scale integrated-circuit chip. By a universal decoder we mean a decoder that can be used to decode any Reed-Solomon or BCH codeword up to the limits of the storage registers associated with the chip. Within these limits it should correct any number of errors and erasures, depending on the code, and for any code blocklength n and symbol alphabet size q. We limit q to be a power of two. A universal chip that could be "programmed" by a few discrete inputs to decode any code within its limits could find extensive applications in magnetic storage systems, in optical disc recording systems, in spread-spectrum packet radio for mobile communications, and in many other places.

This paper presents the architectures of two candidate universal decoders. The development of the paper is concerned only with the algorithmic aspects of the decoders; those aspects associated with logic circuit or chip design are not discussed.

The first universal decoder has a very simple structure and takes n^2 clocks to decode one codeword, where n is the blocklength of the code. The decoding time does not depend on the number of errors or erasures in the received word. The second universal decoder has a more complex structure but is faster. It takes $2tn$ clock intervals to decode one codeword, where $2t + 1$ is the minimum distance of the code. We refer to these as the n^2 decoder and the $2tn$ decoder, respectively. Other possibilities such as a $2t(n - 2t)$ decoder can be developed along the same lines but are not studied in any detail.

Both of the universal decoders decode primitive codes of blocklength equal to $q^m - 1$ for some integer m, codes whose blocklength n is a divisor of $q^m - 1$, shortened codes, and codes extended by a single symbol. The development of the theory and the notation is consistent with that given in [4]. Both of the decoders are of the kind we call "time-domain" decoders. By this we mean that the Berlekamp or Berlekamp-Massey iterations operate on the raw input data as they are received. There is no step that could be called a syndrome computation or a computation of power-sum symmetric functions. This is part of the reason why we feel that our algorithms are good candidates for universal decoders. Of course, the more conventional "frequency domain" algorithms can also be used to build a universal decoder. However, there are then more subsections of the algorithm that need to be reconfigured for every rate, blocklength, or field size. Further, the time-domain algorithms have a regular structure which is important for VLSI circuits.

We begin the paper with a development of Reed-Solomon codes using the suggestive language of the Fourier transform. The Berlekamp-Massey algorithm and the Berlekamp algorithm then are seen as algorithms for spectral estimation, albeit in a Galois field. We obtain our time domain algorithms by using standard properties of the Fourier transform to recast those algorithms. Next, we outline the architecture of universal decoders based on the algorithms. The paper ends with a summary and some typical performance calculations.

Reed-Solomon codes

We begin with a description of Reed-Solomon codes using the suggestive terminology of digital signal processing. The formulation of the decoders given later rests on well-known properties of the discrete Fourier transform.

The discrete Fourier transform

$$V_j = \sum_{i=0}^{n-1} \omega^{ij} v_i, \qquad j = 0, \cdots, n-1,$$

is familiar in digital signal processing. Usually one deals with a time-domain signal $\mathbf{v}$ and a frequency-domain transform $\mathbf{V}$ that are vectors of complex numbers, and the Fourier transform kernel is $\omega = e^{-\sqrt{-1}2\pi/n}$, an nth root of unity in the complex field.

The same definition of a Fourier transform also works in a Galois field. In this case, the time-domain signal $\mathbf{v}$ and the frequency-domain transform $\mathbf{V}$ are vectors of symbols from the Galois field $GF(q)$ and the Fourier transform kernel ω is an element of $GF(q)$, of order n. This formula looks exactly the same as before, but the additions and multiplications it expresses are in the Galois field $GF(q)$. The inverse Fourier transform and the convolution theorem hold because the proofs of these properties are based on only the formal structure of a field. There is one important difference here, however; in a Galois field an nth root of unity does not exist for every n, so a Fourier transform does not exist for every n. This is why error-control codes are usually limited in the choice of blocklength.

A Reed-Solomon code can be defined using the language of the Fourier transform. Let $\mathbf{c}$ be a vector of length n over the field $GF(q)$ with spectrum $\mathbf{C}$ in $GF(q)$. The t-error-correcting Reed-Solomon code of blocklength n with symbols in $GF(q)$ is the set of all vectors $\mathbf{c}$ whose spectrum satisfies $C_j = 0$ for $j = 1, \cdots, 2t$. One way to find these codewords is to encode in the frequency domain. This means setting $C_j = 0$ for $j = 1, \cdots, 2t$ and setting the remaining $n - 2t$ components of $\mathbf{C}$ equal to the $n - 2t$ information symbols. An inverse Fourier transform gives the codeword $\mathbf{c}$. Thus, the number of information symbols equals $n - 2t$ and there are two parity symbols for every error to be corrected. This is not the only way to encode the $n - 2t$ information symbols into codewords—others may yield a simpler implementation—but the frequency-domain method is the most convenient to deal with here. The decoder does not depend on how the codewords are used to store information except for the final step of reading the information out of the corrected codeword.

The proof that the preceding construction does indeed give a code that corrects t errors is the starting point for describing our decoders. The codeword $\mathbf{c}$ is transmitted and the channel makes errors described by the vector $\mathbf{e}$ which is nonzero in not more than t places. The received word $\mathbf{v}$ is written componentwise as

$$v_i = c_i + e_i, \qquad i = 0, \cdots, n-1.$$

The decoder must process the received word $\mathbf{v}$ so as to remove the error word $\mathbf{e}$; the information is then recovered from $\mathbf{c}$. The syndromes of this noisy codeword $\mathbf{v}$ are defined by the following set of $2t$ equations:

$$S_j = \sum_{i=0}^{n-1} \omega^{ij} v_i, \qquad j = 1, \cdots, 2t.$$

Obviously, the syndromes are computed as $2t$ components of a Fourier transform. The received noisy codeword has a Fourier transform given by $V_j = C_j + E_j$ for $j = 0, \cdots, n-1$, and the syndromes are the $2t$ components of this spectrum from 1 to $2t$. But, by construction of a Reed-Solomon code,

$$C_j = 0, \qquad j = 1, \cdots, 2t;$$

hence,

$$S_j = V_j = E_j, \qquad j = 1, \cdots, 2t.$$

The block of syndromes gives us a window through which we can look at $2t$ of the n components of the transform of the error pattern. The decoder must find the entire transform of the error pattern given a segment of length $2t$ of that transform and the additional information that at most t components of the time-domain error pattern are nonzero.

Suppose there are $\nu \le t$ errors at locations with index i_k for $k = 1, \cdots, \nu$. Define the polynomial

$$\Lambda(x) = \prod_{k=1}^{\nu} (1 - x\omega^{i_k}),$$

which is known as the *error-locator polynomial.* The vector $\mathbf{\Lambda}$ of length n whose components Λ_j are coefficients of the polynomial $\Lambda(x)$ has an inverse transform

$$\lambda_i = \frac{1}{n} \sum_{j=0}^{n-1} \Lambda_j \omega^{-ij}.$$

This can be obtained from $\Lambda(x)$ by evaluating $\Lambda(x)$ at $x = \omega^{-i}$. That is,

$$\lambda_i = \frac{1}{n} \Lambda(\omega^{-i}).$$

Therefore,

$$\lambda_i = \frac{1}{n} \prod_{k=1}^{\nu} (1 - \omega^{-i}\omega^{i_k}),$$

which is zero if $i = i_k$ where the i_k for $k = 1, \cdots, \nu$ index the error locations; and otherwise λ_i is nonzero. Hence $\lambda_i = 0$ if and only if $e_i \neq 0$. That is, in the time domain, $\lambda_i e_i = 0$; therefore, the convolution in the frequency domain is zero:

$$\mathbf{\Lambda} * \mathbf{E} = 0.$$

But $\Lambda_j = 0$ for $j > t$, and $\Lambda_0 = 1$, so this can be written

$$\sum_{j=1}^{t} \Lambda_j E_{k-j} = -E_k, \qquad k = 0, \cdots, n-1.$$

This convolution is a set of n equations in $n - t$ unknowns (t unknown components of $\mathbf{\Lambda}$ and $n - 2t$ unknown components of $\mathbf{E}$), and $2t$ known components of $\mathbf{E}$ given by the syndromes. This computation can be described as the operation of a linear feedback shift register with tap weights given by the coefficients of $\Lambda(x)$. It is an autoregressive filter. Of the n equations, the t equations

$$\sum_{j=1}^{t} \Lambda_j S_{k-j} = -S_k, \qquad k = 1 + t, \cdots, 2t,$$

involve only the known syndromes and the t unknown components of $\mathbf{\Lambda}$. These t equations are always solvable for the t unknown components of $\mathbf{\Lambda}$.

After the shift register $\mathbf{\Lambda}$ is computed, the remaining components of $\mathbf{E}$ can be obtained by recursive extension. That is, they can be sequentially computed from $\mathbf{\Lambda}$ using the preceding convolution equation written in the form

$$E_k = -\sum_{j=1}^{t} \Lambda_j E_{k-j}, \qquad k = 0, \cdots, n-1.$$

In this way all components of the vector $\mathbf{E}$ are computed. Then

$$C_j = V_j - E_j.$$

An inverse Fourier transform recovers the initial codeword with all errors corrected. The information symbols may then be read out in accordance with the method of encoding.

The computation of $\mathbf{E}$, the spectrum of the error pattern $\mathbf{e}$ that has least weight, is a problem of spectral estimation, albeit one in a Galois field instead of, as in the more conventional problem, in the real or complex field.

Spectral estimation

The system of equations

$$\sum_{j=1}^{t} \Lambda_j S_{k-j} = -S_k, \qquad k = 1 + t, \cdots, 2t,$$

must be solved for a vector $\mathbf{\Lambda}$. If there are exactly t errors, then it is well known that there is exactly one solution to this system of linear equations. If there are less than t errors, then the determinant of this system of equations will equal zero and there will be more than one solution for $\mathbf{\Lambda}$. Normally one solves for that $\mathbf{\Lambda}$ corresponding to a polynomial $\Lambda(x)$ of smallest degree.

The problem of solving for $\mathbf{\Lambda}$ is the problem of inverting a system of Toeplitz equations. There are many ways of dealing with a Toeplitz system of equations. This instance has an extra property in that the vector on the right side of the equation is related to elements of the Toeplitz matrix in a special way. The most popular algorithm for solving this system of equations for error-control decoders is the Berlekamp-Massey algorithm [5, 6] stated as follows.

Let $S_1, \cdots, S_{2t}$ be given. Let the following set of recursive equations be used to compute $\Lambda^{(2t)}(x)$:

$$\Delta_r = \sum_{j=0}^{n-1} \Lambda_j^{(r-1)} S_{r-j},$$

$$L_r = \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1},$$

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1 - \delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix},$$

for $r = 1, \cdots, 2t$. The initial conditions are $\Lambda^{(0)}(x) = 1$, $B^{(0)}(x) = 1$, $L_0 = 0$, and $\delta_r = 1$ if both $\Delta_r \neq 0$ and $2L_{r-1} \leq r - 1$, and otherwise $\delta_r = 0$. Then $\Lambda^{(2t)}(x)$ is the smallest-degree polynomial with the properties that $\Lambda_0^{(2t)} = 1$ and

$$S_k + \sum_{j=1}^{n-1} \Lambda_j^{(2t)} S_{k-j} = 0, \qquad k = L_{2t} + 1, \cdots, 2t.$$

The Berlekamp-Massey algorithm has $2t$ iterations and each iteration can have on the order of t operations, so the complexity is on the order of t^2. There are also several Fourier transforms to support it and these can have on the order of n^2 operations. After $\mathbf{\Lambda}$ is computed, the recursive extension

$$E_k = -\sum_{j=1}^{t} \Lambda_j E_{k-j}, \qquad k = 2t + 1, \cdots, n,$$

computes the unknown components of $\mathbf{E}$. This requires $n - 2t$ more iterations.

An alternative computation can be used to avoid the $n - 2t$ iterations of the recursive extension, but at the expense of increasing the complexity of the first $2t$ iterations by increasing the number of iterates. The algorithm, now known as the Berlekamp algorithm [5], is expanded to compute three (or sometimes two) polynomials $\Lambda(x)$, $\Lambda'(x)$, and $\Gamma(x)$. The error-locator polynomial $\Lambda(x)$ is as before, $\Lambda'(x)$ is its formal derivative, and $\Gamma(x)$, known as the error-evaluator polynomial, is defined by

$$\Gamma(x) = \Lambda(x)S(x), \qquad (\text{mod } x^{2t}).$$

The reason for computing these quantities is the formula known as the Forney algorithm [7], which is given by

$$e_i = -\frac{\Gamma(\omega^{-i})}{\Lambda'(\omega^{-i})}$$

whenever $\lambda_i = 0$. This expression can be used to compute the error magnitudes directly without the need for the $n - 2t$ extra iterations needed by the Berlekamp-Massey algorithm.

Both $\Lambda'(x)$ and $\Gamma(x)$ can be computed from $\Lambda(x)$ after the first $2t$ iterations are complete, but this procedure does not readily lend itself to the time-domain equations that we want. Instead, it is more convenient to include one or both of them as iterates. To include $\Lambda'(x)$ and $\Gamma(x)$ as iterates, we must also introduce the temporary iterates $B'(x)$ and $A(x)$. The iterations then become

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \\ \Lambda'^{(r)}(x) \\ B'^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x & 0 & 0 \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x & 0 & 0 \\ 0 & -\Delta_r & 1 & -\Delta_r x \\ 0 & (1-\delta_r) & \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \\ \Lambda'^{(r-1)}(x) \\ B'^{(r-1)}(x) \end{bmatrix},$$

$$\begin{bmatrix} \Gamma^{(r)}(x) \\ A^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Gamma^{(r-1)}(x) \\ A^{(r-1)}(x) \end{bmatrix},$$

where Δ_r, L_r, and δ_r are as previously defined, and the initial conditions are

$$\Lambda^{(0)}(x) = B^{(0)}(x) = \Gamma^{(0)}(x) = 1,$$

$$A^{(0)}(x) = \Lambda'^{(0)}(x) = B'^{(0)}(x) = 0.$$

Time-domain decoding

By recognizing the problem of decoding Reed-Solomon codes as a computation in the Fourier transform domain, we have opened other possibilities for the processing. The Berlekamp-Massey algorithm processes the transform of the received word. The Berlekamp-Massey algorithm is preceded by a Fourier transform and is followed by a Fourier transform in some form. However, instead of pushing the received word into the frequency domain, it is possible to push the Berlekamp-Massey algorithm into the time domain [8]. This makes the Fourier transforms simply vanish. On the other hand, the frequency-domain vectors of length t are replaced by time-domain vectors of length n; algorithms that in the frequency domain have complexity t^2 or nt become algorithms in the time domain that have complexity nt or n^2. The time-domain decoder is structurally simple and is useful in applications where structural simplicity is important and the number of iterations is not.

Let λ and $\mathbf{b}$ denote respectively the inverse Fourier transforms of the vectors $\mathbf{\Lambda}$ and $\mathbf{B}$. To push the Berlekamp-Massey equations into the time domain, simply replace the frequency-domain variables Λ_j and B_j with the time-domain variables λ_i and b_i, replace the delay operator x with ω^{-i}, and replace product terms with convolution terms. Replacement of the delay operator with ω^{-i} is justified by the translation property of Fourier transforms; replacement of a product with a convolution is justified by the convolution theorem. Then, as is proved in [8], the time-domain algorithm is as follows.

Let $\mathbf{v}$ be the received noisy Reed-Solomon codeword and let the following set of recursive equations be used to compute $\lambda_i^{(2t)}$ for $i = 0, \cdots, n-1$:

$$\Delta_r = \sum_{i=0}^{n-1} \omega^{ir}[\lambda_i^{(r-1)} v_i],$$

$$L_r = \delta_r(r - L_{r-1}) + (1-\delta_r)L_{r-1},$$

$$\begin{bmatrix} \lambda_i^{(r)} \\ b_i^{(r)} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r\omega^{-i} \\ \Delta_r^{-1}\delta_r & (1-\delta_r)\omega^{-i} \end{bmatrix} \begin{bmatrix} \lambda_i^{(r-1)} \\ b_i^{(r-1)} \end{bmatrix},$$

for $i = 0, \cdots, n-1$ and for $r = 1, \cdots, 2t$. The initial conditions are $\lambda_i^{(0)} = 1$ for all i, $b_i^{(0)} = 1$ for all i, $L_0 = 0$, and $\delta_r = 1$ if both $\Delta_r \neq 0$ and $2L_{r-1} \leq r-1$, and otherwise $\delta_r = 0$. Then $\lambda_i^{(2t)} = 0$ if and only if $e_i \neq 0$.

For nonbinary codes we must also compute the error magnitudes in the frequency domain. These are computed by the following recursion:

$$E_k = -\sum_{j=1}^{t} -\Lambda_j E_{k-j}, \qquad k = 2t+1, \cdots, n-1.$$

It is not possible to just write the Fourier transform of this equation; some restructuring is necessary. The following equivalent set of recursive equations for $r = 2t+1, \cdots, n$ is suitably restructured:

$$\Delta_r = \sum_{i=0}^{n-1} \omega^{ir} v_i^{(r-1)}\lambda_i,$$

$$v_i^{(r)} = v_i^{(r-1)} - \Delta_r\omega^{-ri}.$$

Starting with $v_i^{(2t)} = v_i$, and $\lambda_i = \lambda_i^{(2t)}$ for $i = 0, \cdots, n-1$, the last iteration results in

$$v_i^{(n)} = e_i, \qquad i = 0, \cdots, n-1.$$

The reason this works is that $E_j = V_j$ for $j = 1, \cdots, 2t$, and the new equations, though written in the time domain, are actually sequentially changing V_j to E_j for $j = 2t+1, \cdots, n$.

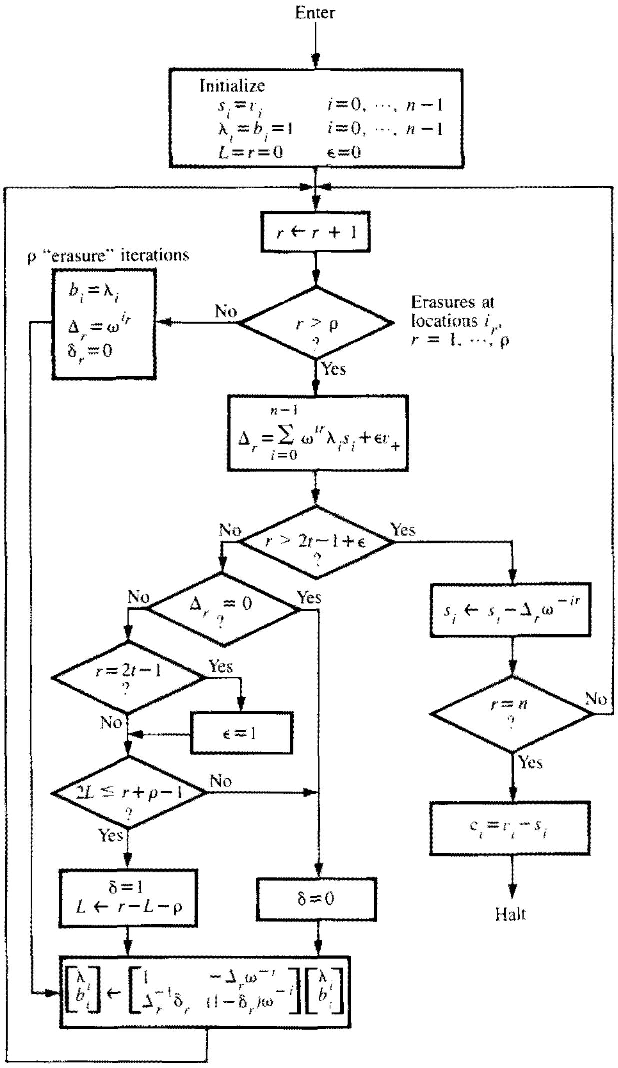

Figure 1 Time-domain decoder algorithm.

The time-domain decoder has no Fourier transforms (no syndrome computation nor Chien search); it has only one major computational block which is easily designed into digital logic. It does, however, always deal with vectors of length n rather than with vectors of length t used by the frequency-domain decoder. Hence, there are hardware/speed tradeoffs.

To get a faster time-domain decoder, we can start with the Berlekamp algorithm. Transformed into the time domain, these equations become the following:

$$\begin{bmatrix} \lambda_i^{(r)} \\ b_i^{(r)} \\ \lambda_i'^{(r)} \\ b_i'^{(r)} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r\omega^{-i} & 0 & 0 \\ \Delta_r^{-1}\delta_r & (1-\delta_r)\omega^{-i} & 0 & 0 \\ 0 & -\Delta_r & 1 & -\Delta_r\omega^{-i} \\ 0 & (1-\delta_r) & \Delta_r^{-1}\delta_r & (1-\delta_r)\omega^{-i} \end{bmatrix} \begin{bmatrix} \lambda_i^{(r-1)} \\ b_i^{(r-1)} \\ \lambda_i'^{(r-1)} \\ b_i'^{(r-1)} \end{bmatrix},$$

$$\begin{bmatrix} \gamma_i^{(r)} \\ a_i^{(r)} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r\omega^{-i} \\ \Delta_r^{-1}\delta_r & (1-\delta_r)\omega^{-i} \end{bmatrix} \begin{bmatrix} \gamma_i^{(r-1)} \\ a_i^{(r-1)} \end{bmatrix},$$

for $i = 0, \cdots, n-1$, and for $r = 1, \cdots, 2t$. The initial conditions are $\lambda_i^{(0)} = b_i^{(0)} = \gamma_i^{(0)} = 1$ for all i; $\lambda_i'^{(0)} = b_i'^{(0)} = a_i^{(0)} = 0$ for all i; $L_0 = 0$; and $\delta_r = 1$ if both $\Delta_r \neq 0$ and $2L_{r-1} \leq r-1$, and otherwise $\delta_r = 0$.

Architecture of the decoders

Now we are ready for the central section of the paper, the architecture of the universal decoders. A flow diagram for the basic n^2 time-domain decoder, which was developed in the previous section, is shown in **Figure 1.** Notice that the initialization is trivial, starting with a syndrome vector equal to the raw data vector just as it is received. At the end of n iterations, this syndrome vector has been changed into the error vector. The decoder decodes both errors and erasures, and can be used for Reed-Solomon codes, BCH codes, and singly extended versions of these codes. Discussion of the algorithmic theory associated with these enhancements is deferred to the next section, although the enhancements themselves are included in the figures of this section.

Most of the clutter in Fig. 1 is concerned with logical tests and the setting of switches, and is quite trivial in a hardware implementation of a decoder. The index r counts out the n iterations, and the flow diagram is best understood by following the r index. During the first ρ iterations, with ρ equal to the number of erased symbols, the basic Berlekamp-Massey iteration is tricked into initializing itself for ρ erasures as is described in the next section. This is done with the same computations as would be done if there were no erasures, except that different variables are switched into the input of the computations. There is virtually no increase in complexity to fill erasures.

Next the Berlekamp-Massey algorithm proceeds through $2t - \rho$ iterations to compute the time-domain error-locator vector. The next-to-last of these iterations is special when decoding singly extended codes. A special test determines whether an extra syndrome is needed, and if so, sets the switch position denoted by ϵ to a one. Then iteration $2t$ can be completed. Otherwise only $2t - 1$ iterations are needed by the Berlekamp-Massey algorithm. The last $n - 2t$ (or $n - 2t + 1$) iterations update s_i to compute the error vector.

The flow can be simplified a little more than is shown in Fig. 1. After the block that updates r, one inserts $s_i \leftarrow \omega^i s_i$. Then the equation for Δ_r loses the term in ω, as does the equation for s_i on the right. Because ω has order n, and there are n iterations, the final s_i is multiplied by ω^{ni}, which equals one. Hence, the final result is not affected.

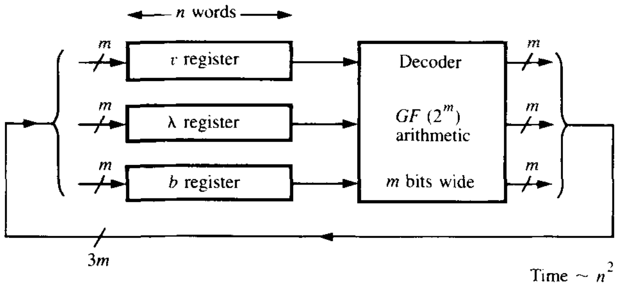

Figure 2 Architecture of a universal Reed-Solomon decoder.

The architecture of a universal decoder is shown in **Figure 2**. The largest symbol field is $GF(2^m)$, which consists of m-bit symbols. In particular, we can take m equal to 8 so the code symbols are up to 8-bit bytes. Then there are 24 bits moving through the decoder in parallel and recirculating back to the shift registers. The contents of the shift registers are shifted into the decoder n times and the shift registers are of length n, so the decoding time is n^2 clock intervals.

The computations within the decoder are just those shown in Fig. 1. There are five hard-wired m-bit by m-bit multiplications running concurrently, an ω^{-i} generator, and an inverse computation. The multiplier structure and the inversion change with the field size. Multipliers are special 8-bit by 8-bit multipliers without carry and with the high-order output bits folded back by the rules of Galois field multiplication. There are also some adders which are bit-by-bit exclusive-or circuits. The accumulation of the sum defining Δ_r proceeds as the variables λ_i, b_i, and s_i are shifted into the decoder arithmetic section. Meanwhile, at the same time, λ_i and b_i are updated and returned to the shift registers to be ready for the next iteration. After Δ_r is computed, it is inverted, perhaps by a table look-up or by discrete logic. In this way, the discrepancy for one value of the index r is being computed concurrently with the update of λ_i and b_i.

The remainder of the decoder consists of switches and minor logic that control the routing of data. The speed of the decoder depends on the number of logic levels between the input and output of the decoder logic. The worst path through the decoder has two multiplications and a bitwise-modulo-two addition. If there are three logic levels in a multiplication, then in one iteration a word drops through seven levels of logic. A clock interval is determined by the time it takes a signal to pass through seven levels of logic. Clearly, one can expect very high decoder speeds.

A flow diagram for the more complex $2tn$ time-domain decoder is shown in **Figure 3** and a decoder is shown in **Figure 4**. Now there are seven words flowing through the decoder in parallel. This requires a 56-bit-wide data path to handle Reed-

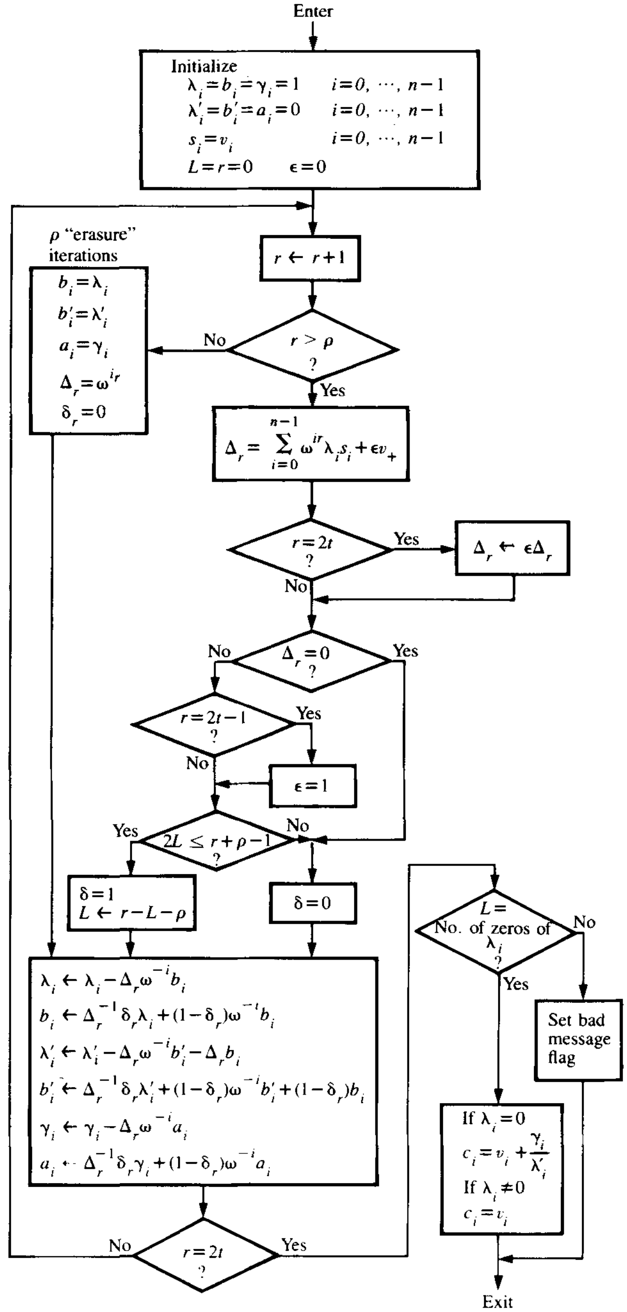

Figure 3 Another time-domain decoder algorithm.

Solomon codes on 8-bit bytes. The decoder terminates with the Forney algorithm. Many of the other features of the decoder are carried over from the n^2 decoder.

One way to compare the timing of Figs. 1 and 3 is with the grid shown in **Figure 5**. Each cell in the grid represents one clock time and there are up to n^2 grid cells. The cells in each column represent the vector components, and each column represents one iteration during which that vector is processed.

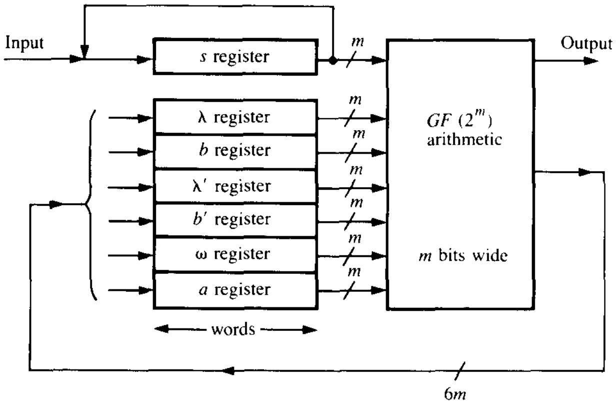

Figure 4 Architecture of another universal Reed-Solomon decoder.

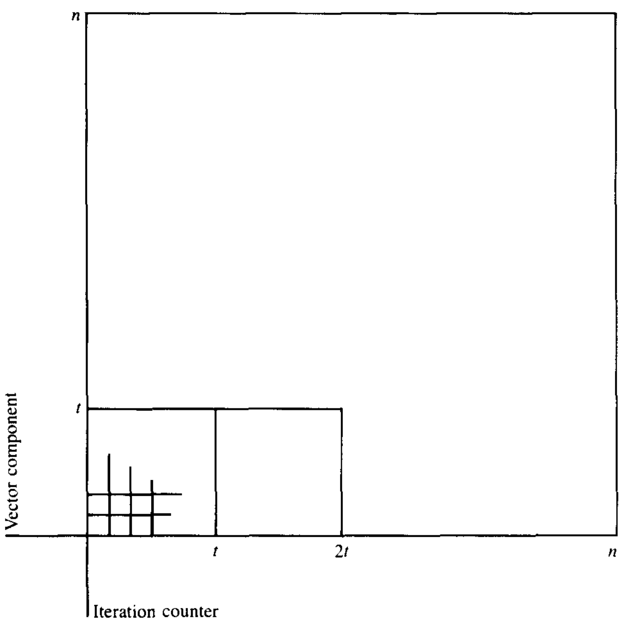

Figure 5 Timing chart.

The speed of an algorithm is, in part, determined by the length of the vectors and the number of iterations. The first algorithm has n iterations and processes vectors of length n. The computation may be thought of as stepping through the squares of the grid column-by-column. There are n^2 computational steps, each of which is rather simple. The second algorithm described has fewer steps because there are only $2t$ iterations. There are $2tn$ computational steps, but these are each about three times as difficult as before.

One can also use Fig. 5 to contemplate the number of clocks of the more conventional frequency-domain Berlekamp-Massey algorithm. Then one has something like $2t^2$ clocks because there are $2t$ iterations involving vectors of length t. That decoder, however, includes other processing steps such as a Fourier transform.

Enhancements

The universal decoders include provision for decoding BCH codes, extended codes, and shortened codes, and for filling erasures as well as correcting errors. This section provides the theory behind these enhancements.

To see that the decoder can correct BCH codes, it is only necessary to realize that every BCH code is a subfield-subcode of a Reed-Solomon code. That is, the BCH code is a subset of a Reed-Solomon code. Hence, since the decoder can correct every codeword in a Reed-Solomon code, it can correct every codeword in a BCH code.

The filling of erasures makes use of a time-domain version of the procedure described in [9]. It is shown there that it suffices to initialize the Berlekamp-Massey algorithm with the erasure-locator polynomial

$$\Omega(x) = \prod_{k=1}^{\rho} (1 - x\omega^{i_k}),$$

where i_k is the location of the kth of ρ erasures. The initialization $\Lambda(x) = B(x) = \Omega(x)$ can be put in the form of the other iterations by using the index r in place of k, and in each iteration multiplying in one factor $(1 - x\omega^{i_r})$. This we put in the form

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\omega^{i_r}x \\ 1 & -\omega^{i_r}x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix},$$

for $r = 1, \cdots, \rho$, and now with the initialization $\Lambda(x) = B(x) = 1$. This is very nearly the form of the Berlekamp-Massey iteration. All that is needed in a circuit is a few switches to replace the discrepancy Δ_r with the erasure location ω^{i_r}, and to update $B^{(r)}(x)$ in the same manner as $\Lambda^{(r)}(x)$. These computations are readily transformed into the time domain. Hence, the cost of including the ability to correct erasures is virtually zero.

Extended Reed-Solomon codes are described by Wolf [10]. The decoding of a singly extended Reed-Solomon code is described in [4]. Singly extended codes are important because they allow one to use codes such as a (32,16) or a (256,240) Reed-Solomon code in which the blocklength is a power of two. Often, a power-of-two blocklength will fit into an application more snugly than a blocklength that is not a power of two.

An (n,k) singly extended Reed-Solomon code with $n = 2^m$ can be described as an $(n - 1,k)$ Reed-Solomon code appended by an additional parity check denoted c_+ and defined by

$$c_+ = \sum_{i=0}^{n-2} c_i \omega^{i2t},$$

provided $C_1, C_2, \cdots, C_{2t-1}$ have been chosen as "parity frequencies." With these parity frequencies the $2t - 1$ spectral components of the received word $V_1, V_2, \cdots, V_{2t-1}$ give enough syndromes to correct $t - 1$ errors and detect t errors in the first $n - 1$ symbols of the received word. If there are t errors detected, then the appended symbol c_+ is error-free. Hence, $V_{2t} - c_+$ gives one more valid syndrome and the t errors can be corrected.

To enhance our decoder so that it can decode a singly extended Reed-Solomon code only requires a test of the discrepancy for zero at iteration $2t - 1$ followed by the trivial equation $S_{2t} = V_{2t} - v_+$ in the frequency domain. It is trivial to include the equivalent of this equation in the time-domain decoder.

The decoders also work for nonprimitive Reed-Solomon codes. These are codes whose blocklength is a divisor of $q - 1$. One merely uses an element of order n for ω. For example, 51 divides 255, so one has a nonprimitive Reed-Solomon code of blocklength 51 with 8-bit symbols. This could be extended to blocklength 52.

One also has the option of using shortened Reed-Solomon codes, that is, codes terminated by fixing some of the information places at zero. The n^2 decoder must carry these zero places through the calculation, so the decoding speed is determined by the blocklength of the unshortened code. The $2nt$ decoder need not process the zero components. Its performance is determined by the blocklength of the shortened code.

Finally, we mention other variations that might be used. The method of Burton [11] could be incorporated to trade a division for multiplications. If a circuit for dividing by the generator polynomial is available to use without additional cost, then one might wish to divide out the generator polynomial to change the input vector to length $n - 2t$, thereby obtaining some speed advantage [but at the cost of using the divide by $g(x)$ circuit]. One also can incorporate schemes that kick out after fewer than $2t$ iterations if there are fewer than t errors in the received word. Simple logical tests suffice for this test.

Summary

Typical performance parameters for a universal decoder are given in **Table 1.** Of course, one might also choose to enlarge these numbers to handle larger codes; the parameters given were chosen to cover most potential applications.

Table 1 Typical design parameters of the universal decoder.

$n \leq 256$ bytes
$q = 2, 4, 8, 16, 32, 64, 128, 256$ (1-bit to 8-bit bytes)
BCH and Reed-Solomon codes
Errors and erasures decoding
Decode time (clocks): n^2 or $2tn$
Logic delay: 7 gates/clock

Table 2 Performance calculations: (a) for the n^2 decoder; (b) for the $2tn$ decoder.

(a) *n^2 decoder*

Symbol field	*Block length*	*Decode time per symbol**	*Bit rate per decoder***
2^{12}	4096	4095	87.9 Kbps
2^8	256	255	941.2 Kbps
2^8	52	51	4.7 Mbps
2^8	18	17	14.1 Mbps

(b) *$2tn$ decoder*

Symbol field	*Block length*	*$2t$*	*Decode time per symbol**	*Bit rate per decoder***
2^{12}	4096	400	400	899.9 Kbps
2^8	256	40	40	6.0 Mbps
2^8	52	10	10	24.0 Mbps
2^8	18	6	6	40.0 Mbps

*In clock times.
** At a 30-MHz clock.

The attraction of the universal decoders is their structural simplicity, which makes it feasible to build the decoder on a single chip, and their versatility, which enables a single chip to handle many applications. Of course, one can always improve performance for a single application by building a more complex decoder or one that is tailored to the application.

Performance calculations for the n^2 and $2tn$ decoders are listed in **Table 2.** These calculations pertain to a decoder that corrects the worst-case error pattern in every received word. We have not considered alternative configurations whose decoding time varies with the number of errors.

References

1. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Indust. Appl. Math.* **8,** 300–304 (1960).
2. A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres* **2,** 147–156 (1959).
3. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Group Codes," *Info. Control* **3,** 68–79 (1960).
4. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Co., Reading, MA, 1983.

5. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968.
6. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Info. Theory* **IT-15**, 122–127 (1969).
7. G. D. Forney, Jr., "On Decoding BCH Codes," *IEEE Trans. Info. Theory* **IT-11**, 549–557 (1965).
8. R. E. Blahut, "Transform Decoding without Transforms," presented at the Tenth IEEE Communication Theory Workshop, Cypress Gardens, FL, 1980.
9. R. E. Blahut, "Transform Techniques for Error Control Codes," *IBM J. Res. Develop.* **23**, 299–315 (1979).
10. J. K. Wolf, "Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications," *Bell Syst. Tech. J.* **48**, 2405–2424 (1969).
11. H. O. Burton, "Inversionless Decoding of Binary BCH Codes," *IEEE Trans. Info. Theory* **IT-17**, 464–466 (1971).

Received August 4, 1983; revised October 11, 1983

Richard E. Blahut *IBM Federal Systems Division, Owego, New York 13827.* Dr. Blahut is an IBM Fellow at the Owego facility, where he joined IBM in 1964. He is involved in analyzing and designing jam-protected digital communications systems, statistical information processing systems, and radar imaging systems. He was awarded an IBM Outstanding Contribution Award in 1976 for developing coherent passive location theory. He is also the recipient of the IEEE Information Theory Group 1974 paper award for his paper, "Computation of Channel Capacity and Rate Distortion Functions." Dr. Blahut is writing several textbooks, including one on error-control codes, which has recently appeared. He was the 1982 president of the IEEE Information Theory Group and is an IEEE Fellow and a Courtesy Professor of Electrical Engineering at Cornell University, Ithaca, New York. His education includes a Ph.D. in electrical engineering from Cornell University in 1972, an M.S. in physics from Stevens Institute of Technology, Hoboken, New Jersey, in 1964, and a B.S. in electrical engineering from the Massachusetts Institute of Technology in 1960.