An Efficient Program for Decoding the (255,223) Reed-Solomon Code Over *GF* (2^8) With Both Errors and Erasures, Using Transform Decoding

R. L. Miller and T. K. Truong
Communications Systems Research Section

I. S. Reed
University of Southern California

To decode a (255,223) Reed-Solomon code over GF(2^8), a fast Fourier-like transform over GF(2^8) has been developed to compute the syndromes and the error-erasure vectors of the transmitted code words. This new simplified transform decoder is implemented in a program on a digital computer. The (255,223) Reed-Solomon code over GF(2^8) is a NASA standard for concatenation with a convolutional code. In a simulation, random code words were corrupted by random error and erasure patterns, and decoded whenever theoretically possible. A matrix of execution times for this new transform decoder under varying sets of errors and erasure patterns is included in the paper. This matrix demonstrates that the speed of the new decoder is between 3 and 7 times faster than the standard R-S decoder, developed previously by NASA.

I. Introduction

Recently the authors developed (Ref. 2) a simplified algorithm for correcting erasures and errors of Reed-Solomon (R-S) codes over the finite field $GF(p^m)$, where p is a prime and m is an integer. For a space communication application it was shown (Ref. 3) that a 16-error-correcting R-S code of 255 eight-bit symbols, concatenated with a $k = 7$, rate = 1/2 or 1/3, Viterbi decoded convolutional code, can be used to reduce the signal-to-noise ratio required to meet a specified bit-error rate. Such a concatenated code is being considered for the Galileo project and the NASA End-to-End Data System.

In a concatenated code, the inner convolutional decoder is sometimes able to find only two or more equally probable symbols. For such a case, the best policy is to declare an erasure of the symbol. If the outer R-S code is able to utilize the additional information that certain erasures have occurred, then clearly the system performance will be enhanced.

The algorithm given in Ref. 2 is used here to correct patterns of t errors and s erasures of the words of the (255,223) R-S code, where $2t + s < 33$ and where the symbols belong to the finite field $GF(2^8)$. Define the following five vectors:

$(c_0, c_1, \ldots, c_{254}) = \mathbf{c}$, code vector

$(r_0, r_1, \ldots, r_{254}) = \mathbf{r}$, received vector

$(\mu_0, \mu_1, \ldots, \mu_{254}) = \boldsymbol{\mu}$, erasure vector

$(e_0, e_1, \ldots, e_{254}) = \mathbf{e}$, error vector

$(\mu_0, \mu_1, \ldots, \mu_{254}) = \tilde{\boldsymbol{\mu}}$, new erasure vector

These vectors are related by $\mathbf{r} = \mathbf{c} + \mu + \mathbf{e}$ and $\tilde{\mu} = \mathbf{e} + \mu$.

Suppose that t errors and s erasures occur in the received vector $\mathbf{r}$ of 255 symbols, and assume $2t + s < 33$. Then, following the algorithm in Ref. 2, the decoding procedure consists of these five steps:

(1) Compute the syndromes $S_k (1 \leq k \leq 32)$ of the received 255-tuple $(r_0, r_1, \ldots, r_{254})$, i.e.,

$$S_k = \sum_{i=0}^{254} r_i \alpha^{ik} \text{ for } k = 1, 2, \ldots, 32 \qquad (1)$$

where α is an element of order 255 in $GF(2^8)$. If $S_k = 0$ for $1 \leq k \leq 32$, then $\mathbf{r}$ is a code word and no further decoding is necessary. Otherwise,

(2) Compute τ_j for $j = 0,1,2, \ldots, s$ from the erasure locator polynomial

$$\tau(x) = \prod_{j=1}^{s} (x - Z_j) = \sum_{j=1}^{s} (-1)^j \tau_j x^{s-j} \qquad (2)$$

where s is the number of erasures in the received vector, and $Z_j (1 \leq j \leq s)$ are the known erasure locations. Next compute the Forney syndromes T_i for $1 \leq i \leq 32 - s$ from the equation

$$T_i = \sum_{j=0}^{s} (-1)^j \tau_j S_{i+s-j} \text{ for } 1 \leq i \leq 32 - s \qquad (3)$$

where $\tau_j (1 \leq j \leq s)$ and $S_j (1 \leq j \leq 32)$ are known.

(3) If $0 \leq s < 32$, then use continued fractions (see Ref. 4) to determine the error locator polynomial $\sigma(x)$ from the known T_i's $(1 \leq i < 32 - s)$. For the special case $s = 32$, it was shown (Refs. 4, 5) that it is impossible for any decoder to tell whether there are zero or more additional errors. Thus, for $s = 32$, the best policy is not to decode the message at all.

(4) Compute the combined erasure and error locator polynomial from the equation

$$\tilde{\tau}(x) = \sigma(x)\tau(x) = \sum_{k=0}^{s+t} (-1)^k \tilde{\tau}_k x^{s+t-k} \qquad (4)$$

where $\sigma(x)$ and $\tau(x)$ are now known. Then compute the rest of the transform of the erasure and error vector from the equation

$$S_l = \sum_{k=1}^{s+t} (-1)^k \tilde{\tau}_k S_{l-k} \text{ for } l > 32.$$

Note that $S_{255} = S_0$.

(5) Compute the inverse transform of the syndrome vector $(S_0, S_1, \ldots, S_{254})$ to obtain the error-erasure vector. That is, calculate

$$\tilde{\mu}_i = e_i + \mu_i = \sum_{k=0}^{254} S_k \alpha^{-ik} \text{ for } i = 0, 1, 2, \ldots, 254 \qquad (5)$$

Finally, subtract the error-erasure vector $\tilde{\mu}$ from the received vector to correct it.

Previously the authors (Ref. 1) implemented the above decoding procedure on the UNIVAC 1108 computer. The disadvantage of this decoding program was that the syndromes in Step 1 were computed directly instead of using FFT-like techniques. Also, the slower Chien-type search was used to find the roots of the error polynomial instead of Steps (4) and (5), above. Finally, another direct inverse transform of the syndrome vector points was used to obtain the error and erasure magnitudes.

It was shown (Ref. 6) that a combination of the Chinese remainder theorem and Winograd's algorithm (Refs. 7 and 8) could be used to develop a fast algorithm for computing the syndromes needed for decoding an R-S or BCH code. Such a method requires only a small fraction of the number of multiplications and additions that are required in a direct computation. More generally, it was shown also in Ref. 9 that a modification of Winograd's method could be used to compute $(2^m - 1)$-point transforms over $GF(2^m)$, where $m = 4,5,6,8$.

In this article, the methods developed in Refs. 6 and 9 are applied to compute the syndromes in Step 1 and the error-erasure vector discussed in Step 5. An important advantage of this new transform decoder over the previous methods (Refs. 1 and 10) is that the complexity of the syndrome calculation is substantially reduced. Furthermore, the Chien-type search (Ref. 8) for the roots of the error locator polynomial is completely eliminated. These are replaced by the computation of a 255-point transform using FTT-like techniques. The result is a simpler and faster decoder than can be obtained by conventional means.

The simplified transform decoder was written in FORTRAN V and was implemented on the UNIVAC 1108 computer. The matrix of decoding times for correcting errors and erasures of the (255,223) in this simplified decoder is given in

Table 1. Also, the matrix of decoding times of the conventional algorithm, described in Ref. 1, is provided in Table 2. A comparison of Tables 1 and 2 shows that the new algorithm is faster by a factor of from 3 to 7.

II. A Fast Technique for Computing the Error and Erasure Magnitudes

The computation of the error-erasure vector in Step 5 of the transform decoder is based on the methods in Refs. 8 and 9. The key idea is the use of FFT-like techniques over the the finite field $GF(2^8)$. These concepts are used to compute efficiently the expression

$$A_j = \sum_{i=0}^{255-1} a_i \alpha^{ij} \text{ for } 0 \leqslant j \leqslant 254 \qquad (6)$$

where α is an element of order 255 in $GF(2^8)$. To begin with, since $n = 255 = 3 \times 5 \times 17$, by Refs. 7 and 8, (6) can be decomposed into the following 3 stages:

Stage 1

$$A^1_{(i_1, i_2, j_3)} = \sum_{i_3=0}^{3-1} a_{(i_1, i_2, i_3)} \alpha_3^{i_3 j_3} \text{ for } 0 \leqslant j_3 \leqslant 2$$

Stage 2

$$A^2_{(i_1, j_2, j_3)} = \sum_{i_2=0}^{5-1} A^1_{(i_1, i_2, j_3)} \alpha_2^{i_2 j_2} \text{ for } 0 \leqslant j_2 \leqslant 4$$

Stage 3

$$S_{(j_1, j_2, j_3)} = \sum_{i_1=0}^{17-1} A^2_{(i_1, j_2, j_3)} \alpha_1^{i_1 j_1} \text{ for } 0 \leqslant j_1 \leqslant 16 \qquad (7)$$

where $\alpha_3 = \alpha^{85}$, $\alpha_2 = \alpha^{51}$, and $\alpha_1 = \alpha^{120}$. In (7), Stage 1, Stage 2, and Stage 3 are 3-, 5-, and 17-point transforms, respectively. The detailed algorithms for computing the 3-, 5-, and 17-point transform over $GF(2^8)$ are reproduced in the appendix. It follows from these algorithms that the number of multiplications needed to compute a 3-, 5-, and 17-point transform is 1, 5, and 53, respectively. In similar fashion, it can be shown that the number of additions needed to compute a 3-, 5-, and 17-point transform is 5, 17, and 173, respectively. Thus, the total number of multiplications and additions needed to compute the A_j for $0 \leqslant j \leqslant 254$ is $17 \times 5 \times 1 + 17 \times 5 \times 3 + 53 \times 5 \times 3 = 1135$ and $17 \times 5 \times 5 + 17 \times 5 \times 17 + 173 \times 5 \times 3 = 4465$, respectively. In contrast, using a Chien-type search algorithm, $254(s + t)$ multiplications and $254(s + t)$ additions are required for a direct computation of the inverse transform at the $s + t$ points corresponding to the s erasure locations and the t error locations.

III. Program Design and Implementation

The decoding procedure described in the previous section was implemented on the UNIVAC 1108 computer using FORTRAN V. This program is used to correct any combination of t errors and s erasures occurring in a 255-symbol R-S code word, where $2t + s < 33$. The overall basic structure of the program is given in Fig. 1. It is divided into a main program and five major subroutines.

The Main Program. This is the main driver of the rest of the program. It initializes the decoding process and keeps track of the elapsed CPU time.

Input. This subroutine generates a random code vector (polynomial) $R(x)$ for the R-S decoder and then adds errors and erasures $E(x)$ to it.

Step 1. The first 32 syndromes of the received vectors as well as the corrected vectors are calculated in this subroutine using a combination of the Chinese remainder theorem and Winograd's algorithm as described in Ref. 5. In case the corrected received word is not an R-S code word, the subroutine will output the message, "The corrected received vector is not a codeword." This helps to confirm the correctness of the program, as well as to indicate that the number of errors and erasures have exceeded the limits allowable by the decoder.

Step 2. This subroutine computes the Forney syndrome vector **T** (Eq. 3) from the erasure vector **Z**. The erasure locator polynomial $\tau(x)$ (Eq. 2) is also calculated in this step.

Step 3. The error locator polynomial $\sigma(x)$ is calculated from the Forney syndrome vector **T** using the continued fraction algorithm. The product of the error locator polynomial $\sigma(x)$ and the erasure locator polynomial $\tau(x)$ is next computed (Eq. 4). The coefficients of this erasure and error locator polynomial are used to compute the remaining terms $S_{33}, \ldots, S_{255}$.

Step 4. This step directly computes the inverse Fourier transform of the vector $(S_1, S_2, \ldots, S_{255})$ to obtain the error and erasure vector. Finally, the received vector is corrected to provide an estimate of the transmitted code word.

IV. Simulation Results

The computation times of this new algorithm and the conventional method described in Ref. 1 for decoding code words which were corrupted by many different error and erasure patterns are given in Tables 1 and 2. These results were obtained by performing five trials for each entry in the tables and then averaging. Along any row or column, the computation times tend to increase with the row or column indices until decoding failures occur due to an excess of allowable errors and erasures. An examination of the decoding times in these two tables indicates that the new decoder operates considerably faster than the conventional decoder described in Ref. 1. If the received word is the same as the originally transmitted code word (i.e., if no errors or erasures occurred), the new decoder will be seven times faster. If the received word contains s erasures and t errors, where $2t + s < 33$, the new decoder will operate about three times faster. Finally, if $2t + s \geqslant 33$, then the new decoder will operate about twice as fast.

Table 1. Decoder execution times in seconds (new method)

Erasures \ Errors	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.046	0.119	0.132	0.176	0.161	0.177	0.186	0.201	0.214	0.272	0.240	0.252	0.254	0.328	0.344	0.325	0.319	0.325
1	0.121	0.129	0.144	0.189	0.170	0.185	0.196	0.210	0.223	0.270	0.252	0.261	0.273	0.355	0.360	0.366	0.105	
2	0.131	0.143	0.155	0.200	0.180	0.201	0.208	0.219	0.236	0.247	0.262	0.270	0.232	0.352	0.366	0.346	0.324	
3	0.141	0.151	0.165	0.204	0.192	0.207	0.220	0.232	0.246	0.258	0.270	0.279	0.293	0.388	0.365	0.103		
4	0.150	0.164	0.175	0.188	0.201	0.216	0.230	0.242	0.257	0.265	0.278	0.303	0.301	0.373	0.328	0.339		
5	0.158	0.174	0.186	0.198	0.211	0.228	0.240	0.250	0.260	0.287	0.290	0.298	0.322	0.384	0.097			
6	0.170	0.187	0.196	0.211	0.222	0.237	0.250	0.261	0.274	0.296	0.297	0.307	0.341	0.393	0.337			
7	0.181	0.199	0.207	0.219	0.285	0.245	0.262	0.269	0.283	0.294	0.305	0.356	0.330	0.112				
8	0.192	0.207	0.225	0.229	0.286	0.254	0.267	0.278	0.299	0.304	0.314	0.359	0.337	0.408				
9	0.207	0.215	0.227	0.246	0.278	0.265	0.279	0.289	0.305	0.315	0.325	0.334	0.113					
10	0.211	0.227	0.239	0.248	0.262	0.273	0.289	0.297	0.309	0.322	0.333	0.343	0.442					
11	0.235	0.235	0.247	0.258	0.271	0.284	0.297	0.307	0.320	0.348	0.342	0.088						
12	0.230	0.245	0.256	0.267	0.287	0.293	0.308	0.315	0.327	0.363	0.351	0.356						
13	0.239	0.254	0.265	0.279	0.290	0.302	0.316	0.327	0.338	0.350	0.084							
14	0.250	0.264	0.277	0.287	0.300	0.312	0.325	0.335	0.348	0.366	0.361							
15	0.260	0.274	0.293	0.299	0.311	0.323	0.332	0.344	0.355	0.083								
16	0.270	0.326	0.299	0.305	0.320	0.330	0.342	0.353	0.370	0.377								
17	0.278	0.361	0.334	0.315	0.331	0.340	0.351	0.362	0.081									
18	0.289	0.354	0.350	0.337	0.338	0.349	0.360	0.374	0.389									
19	0.306	0.337	0.325	0.334	0.346	0.357	0.367	0.077										
20	0.315	0.331	0.331	0.344	0.389	0.367	0.379	0.380										
21	0.324	0.342	0.342	0.352	0.417	0.376	0.075											
22	0.324	0.358	0.351	0.360	0.374	0.386	0.388											
23	0.338	0.371	0.361	0.368	0.392	0.074												
24	0.344	0.391	0.370	0.380	0.390	0.393												
25	0.356	0.391	0.401	0.387	0.072													
26	0.363	0.379	0.465	0.398	0.401													
27	0.373	0.393	0.523	0.070														
28	0.381	0.392	0.545	0.407														
29	0.390	0.401	0.092															
30	0.399	0.410	0.504															
31	0.410	0.069																

Table 2. Decoder execution times in seconds (conventional method)

Erasures \ Errors	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.314	0.354	0.397	0.463	0.529	0.483	0.515	0.564	0.608	0.681	0.750	0.760	0.734	0.812	0.777	0.940	0.878	0.592
1	0.342	0.375	0.466	0.472	0.483	0.495	0.549	0.584	0.625	0.661	0.709	0.771	0.746	0.792	0.887	0.867	0.414	
2	0.377	0.427	0.439	0.455	0.485	0.557	0.561	0.616	0.609	0.737	0.813	0.703	0.811	0.899	0.871	0.837	0.577	
3	0.382	0.421	0.455	0.481	0.511	0.571	0.574	0.607	0.744	0.757	0.697	0.779	0.814	0.840	0.836	0.429		
4	0.439	0.456	0.466	0.524	0.594	0.634	0.588	0.629	0.761	0.679	0.710	0.778	0.848	0.943	0.957	0.577		
5	0.437	0.471	0.483	0.582	0.583	0.596	0.609	0.638	0.763	0.759	0.782	0.810	0.868	1.054	0.409			
6	0.439	0.472	0.552	0.572	0.574	0.631	0.646	0.656	0.697	0.724	0.808	0.794	0.930	0.997	0.521			
7	0.474	0.492	0.625	0.612	0.588	0.660	0.782	0.679	0.770	0.817	0.790	0.815	0.943	0.401				
8	0.492	0.610	0.647	0.661	0.640	0.648	0.699	0.692	0.820	0.897	0.784	0.864	0.906	0.521				
9	0.583	0.595	0.665	0.626	0.625	0.664	0.720	0.741	0,781	0.840	0.812	0.918	0.432					
10	0.659	0.607	0.668	0.683	0.730	0.724	0.746	0.767	0.769	0.823	0.936	1.034	0.485					
11	0.561	0.695	0.682	0.661	0.741	0.763	0.733	0.770	0.818	0.811	0.849	0.405						
12	0.606	0.661	0.681	0.756	0.691	0.741	0.753	0.783	0.919	0.830	0.912	0.462						
13	0.656	0.638	0.687	0.737	0.708	0.740	0.800	0.816	0.901	0.858	0.356							
14	0.620	0.760	0.716	0.740	0.735	0.779	0.785	0.831	0.924	0.949	0.449							
15	0.693	0.778	0.719	0.749	0.840	0.863	0.824	0.981	0.883	0.424								
16	0.662	0.735	0.715	0.766	0.758	0.860	0.833	0.907	0.912	0.481								
17	0.802	0.744	0.712	0.800	0.789	0.852	0.841	0.864	0.402									
18	0.858	0.720	0.756	0.825	0.806	0.980	0.859	0.879	0.506									
19	0.852	0.747	0.771	0.791	0.802	0.963	0.897	0.347										
20	0.756	0.740	0.841	0.935	0.831	0.878	0.972	0.428										
21	0.758	0.763	0.954	0.911	0.918	0.899	0.386											
22	0.750	0.805	0.956	0.898	0.875	1.049	0.406											
23	0.769	0.807	0.825	0.858	0.910	0.429												
24	0.811	0.902	0.893	0.892	0.907	0.439												
25	0.829	0.857	0.906	0.945	0.342													
26	0.872	0.920	0.907	0.928	0.372													
27	0.960	0.960	0.935	0.340														
28	0.914	0.908	0.935	0.507														
29	1.028	0.930	0.374															
30	0.938	0.945	0.826															
31	0.960	0.374																

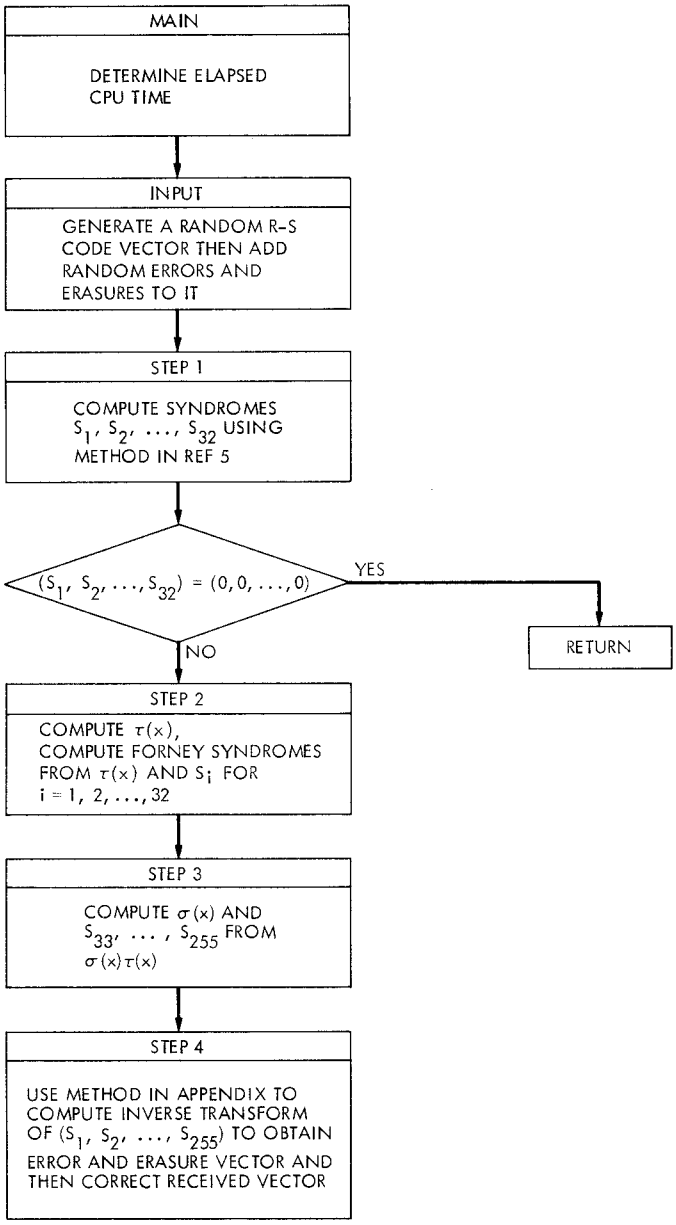

Fig. 1. Basic functional structure of R-S decoding program using transform over *GF* (2^8) and continued fractions

Appendix

This appendix contains the algorithms for computing 3-, 5- and 17-point transforms over fields of characteristic two containing the necessary roots of unity. Let α be an element of order 255 in $GF(2^8)$.

The 3-point transform is given by

$$A_k = \sum_{n=0}^{3-1} a_n \alpha_3^{nk} \quad \text{for} \quad 0 \leqslant k \leqslant 2$$

where $\alpha_3 = \alpha^{85}$ is a primitive cube root of unity.

Algorithm for 3-Point Transform

$$s_1 = a_1 + a_2, \quad A_0 = s_1 + a_0, \quad m_1 = \alpha_3 s_1, \quad s_2 = A_0 + m_1,$$
$$A_1 = s_2 + a_1, \quad A_2 = s_2 + a_2$$

Thus, the 3-point transform requires only one multiplication and five additions.

The 5-point transform is given by

$$A_k = \sum_{n=0}^{5-1} a_n \alpha_2^{nk} \quad \text{for} \quad 0 \leqslant k \leqslant 4$$

where $\alpha_2 = \alpha^{51}$ is a primitive fifth root of unity.

Algorithm for 5-Point Transform

$$s_1 = a_2 + a_3, \quad s_2 = a_1 + a_4, \quad s_3 = a_1 + a_3, \quad s_4 = a_2 + a_4,$$
$$s_5 = s_1 + s_2, \quad A_0 = s_5 + a_0, \quad m_1 = (1 + \alpha_2^3) s_5,$$
$$m_2 = (\alpha_2^3 + \alpha_2^4) s_1, \quad m_3 = (\alpha_2 + \alpha_2^3) s_2, \quad m_4 = (\alpha_2 + \alpha_2^4) s_3,$$
$$m_5 = (\alpha_2 + \alpha_2^5) s_4, \quad s_6 = A_0 + m_1, \quad s_7 = s_6 + m_2,$$
$$s_8 = s_6 + m_3, \quad s_9 = m_5 + a_2, \quad s_{10} = m_4 + a_1,$$
$$s_{11} = m_5 + a_4, \quad s_{12} = m_4 + a_3, \quad A_1 = s_8 + s_9,$$
$$A_2 = s_7 + s_{10}, \quad A_3 = s_7 + s_{11}, \quad A_4 = s_8 + s_{12}$$

Thus, the 5-point transform requires only 5 multiplications and 17 additions.

The 17-point transform is given by

$$A_k = \sum_{n=0}^{17-1} a_n \alpha_1^{nk} \text{ for } 0 \leqslant k \leqslant 16$$

where $\alpha_1 = \alpha^{120}$ is a primitive seventeenth root of unity.

Algorithm for 17-Point Transform

$$S_1 = Y_2 + Y_3, \quad S_2 = Y_1 + Y_4, \quad S_3 = Y_1 + Y_3,$$
$$S_4 = Y_2 + Y_4, \quad S_5 = S_3 + S_4, \quad N_1 = BS_5,$$
$$N_2 = (A + B) S_1, \quad N_3 = (C + B) S_2, \quad N_4 = (C + A) S_3,$$
$$N_5 = (C + A) S_4, \quad N_6 = E \cdot Y_1, \quad N_7 = E \cdot Y_3,$$
$$N_8 = E \cdot Y_4, \quad N_9 = E \cdot Y_2, \quad S_6 = N_1 + N_2,$$
$$S_7 = N_1 + N_3, \quad S_8 = S_6 + N_4, \quad S_9 = S_7 + N_4,$$
$$S_{10} = N_6 + N_5, \quad S_{11} = S_7 + N_5, \quad X_1 = S_{10} + N_9,$$
$$X_2 = S_8 + N_6, \quad X_3 = S_{10} + N_8, \quad X_4 = S_9 + N_7 \qquad \text{(A-1)}$$

where

$$A = \begin{pmatrix} \alpha_1^8 & \alpha_1^6 & \alpha_1^{13} & \alpha_1^{14} \\ \alpha_1^6 & \alpha_1^{13} & \alpha_1^{14} & \alpha_1^2 \\ \alpha_1^{13} & \alpha_1^{14} & \alpha_1^2 & \alpha_1^{10} \\ \alpha_1^{14} & \alpha_1^2 & \alpha_1^{10} & \alpha_1^{16} \end{pmatrix}$$

$$B = \begin{pmatrix} \alpha_1^2 & \alpha_1^{10} & \alpha_1^{16} & \alpha_1^{12} \\ \alpha_1^{10} & \alpha_1^{16} & \alpha_1^{12} & \alpha_1^9 \\ \alpha_1^{16} & \alpha_1^{12} & \alpha_1^9 & \alpha_1^{11} \\ \alpha_1^{12} & \alpha_1^9 & \alpha_1^{11} & \alpha_1^4 \end{pmatrix}$$

$$C = \begin{pmatrix} \alpha_1^9 & \alpha_1^{11} & \alpha_1^4 & \alpha_1^3 \\ \alpha_1^{11} & \alpha_1^4 & \alpha_1^3 & \alpha_1^{15} \\ \alpha_1^4 & \alpha_1^3 & \alpha_1^{15} & \alpha_1^7 \\ \alpha_1^3 & \alpha_1^{15} & \alpha_1^7 & \alpha_1^1 \end{pmatrix}$$

$$D = \begin{pmatrix} \alpha_1^{15} & \alpha_1^7 & \alpha_1^1 & \alpha_1^5 \\ \alpha_1^7 & \alpha_1^1 & \alpha_1^5 & \alpha_1^8 \\ \alpha_1^1 & \alpha_1^5 & \alpha_1^8 & \alpha_1^6 \\ \alpha_1^5 & \alpha_1^8 & \alpha_1^6 & \alpha_1^{13} \end{pmatrix}$$

$X_1 = [A_3, A_{15}, A_7, A_1]^T$, $X_2 = [A_5, A_8, A_6, A_{13}]^T$,

$X_3 = [A_{12}, A_9, A_{11}, A_4]^T$,

$X_4 = [A_{14}, A_2, A_{10}, A_{16}]^T$,

$E = A + B + C + D$

and Y_1 through Y_4 are obtained from the expressions for X_1 through X_4 on replacing each A_i by a_i. Note that (A-1) requires nine (4 X 4) matrix multiplications.

For convenience,

$$\sum_{i=0}^{7} a_i \alpha^i$$

will be represented as the integer

$$\sum_{i=0}^{7} a_i 2^i$$

Observe that N_1 in (A-1) can be put in the form

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 44 & 150 & 169 & 195 \\ 150 & 169 & 145 & 185 \\ 169 & 145 & 185 & 193 \\ 145 & 185 & 193 & 36 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} \quad \text{(A-2)}$$

where

$c_1 = a_3 + a_5 + a_{12} + a_{14}$, $c_2 = a_{15} + a_8 + a_9 + a_2$,

$c_3 = a_7 + a_6 + a_{11} + a_{10}$, $c_4 = a_1 + a_{13} + a_4 + a_{16}$

From (B-8) in Ref. 9, one obtains the algorithm for computing (A-2). That is,

$s_1 = c_1 + c_3$, $s_2 = c_1 + c_2$, $s_3 = c_2 + c_4$, $s_4 = c_3 + c_4$,

$s_5 = s_2 + s_4$, $M_1 = 145 \times s_5$, $M_2 = 56 \times s_1$,

$M_3 = 40 \times s_3$, $M_4 = 7 \times s_2$, $M_5 = 80 \times s_4$,

$M_6 = 130 \times c_1$, $M_7 = 23 \times c_2$, $M_8 = 64 \times c_7$,

$M_9 = 205 \times c_4$, $s_6 = a_0 \times M_1$, $s_7 = s_6 + M_2$,

$s_8 = s_7 + M_4$, $s_9 = s_6 + M_3$, $s_{10} = s_9 + M_4$,

$s_{11} = s_7 + M_5$, $s_{12} = S_9 + M_5$, $b_1 = s_8 + M_6$,

$b_2 = s_{10} + M_7$, $b_3 = s_{10} + M_8$, $b_4 = s_{11} + M_9$,

$A_0 + s_5$

Thus,

$b_1 = s_8 + M_6$, $b_2 = s_{10} + M_7$, $b_3 = s_{10} + M_8$,

$b_4 = s_{11} + M_9$

Hence, the total number of Galois field multiplications and additions needed to compute N_1 is 9 and 17, respectively.

The quantities N_i for i = 2, 3, 4, 5, defined in (A-1), are computed by the same procedure as indicated in Ref. 9. The number of multiplications and additions needed to compute N_i for i = 2, 3, 4, 5 is 9 and 15, respectively.

To compute N_i for i = 6, 7, 8, 9, for example, consider $N_6 = E \cdot Y_1$, i.e.,

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} \alpha^{170} & \alpha^0 & \alpha^{85} & \alpha^0 \\ \alpha^0 & \alpha^{85} & \alpha^0 & \alpha^{170} \\ \alpha^{85} & \alpha^0 & \alpha^{170} & \alpha^0 \\ \alpha^0 & \alpha^{170} & \alpha^0 & \alpha^{85} \end{pmatrix} \begin{pmatrix} a_3 \\ a_{15} \\ a_7 \\ a_1 \end{pmatrix} \quad \text{(A-3)}$$

By a procedure similar to that used to compute the matrix (B-8) in Ref. 9, one obtains the algorithm for (A-3). That is,

$$s_1 = a_3 + a_7, \quad s_2 = a_{15} + a_1, \quad s_3 = s_1 + s_2, \quad M_1 = 1 \times s_5,$$
$$M_2 = 215 \times s_1, \quad M_3 = 214 \times s_2, \quad s_4 = M_1 + M_2,$$
$$s_5 = M_1 + M_3, \quad b_1 = s_4 + a_3, \quad b_2 = s_5 + a_{15},$$
$$b_3 = s_4 + a_7, \quad b_4 = s_5 + a_1 \qquad \text{(A-4)}$$

Thus, from (A-4), the number of multiplications and additions needed to compute N_6 is 2 and 9, respectively. Similarly, the matrix N_i for i = 7, 8, 9 requires 2 multiplications and 9 additions. After combining the above results, the total number of multiplications and additions needed to compute a 17-point transform in $GF(2^8)$ is 53 and 173, respectively.

References

1. Miller, R. L., Truong, T. K., Benjauthrit, B., and Reed, I. S., "A Reed-Solomon Decoding Program for Correcting Both Errors and Erasures," in *The Deep Space Network Progress Report 42-53*, pp. 102-107, Jet Propulsion Laboratory, Pasadena, Calif., Oct. 15, 1979.

2. Reed, I. S., and Truong, T. K., "A Simplified Algorithm for Correcting Both Errors and Erasures of R-S Codes," in *The Deep Space Network Progress Report 42-48*, Jet Propulsion Laboratory, Pasadena, Calif., Sept. 1978.

3. Odenwalder, J., et al., "Hybrid Coding Systems Study Final Report," Linkabit Corp., NASA CR114, 486, Sept. 1972.

4. Reed, I. S., Scholtz, R. A., Truong, T. K., and Welch, L. R., "The Fast Decoding of Reed-Solomon Codes Using Fermont Theoretic Transforms and Continued Fractions," *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 1, pp. 100-106, Jan. 1978.

5. Berlekamp, E. R., and Ramsey, J. L., "Readable Erasures Improve the Performance of Reed-Solomon Codes," *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 5, Sept. 1978.

6. Reed, I. S., Truong, T. K., and Miller, R. L., "A Fast Technique for Computing Syndromes of BCH and RS Codes," in *The Deep Space Network Progress Report 42-52*, pp. 67-70, Jet Propulsion Laboratory, Pasadena, Calif., Aug. 15, 1979.

7. Winograd, S., "On Computing the Discrete Fourier Transform," *Proc. Nat. Acad. Sci. USA*, Vol. 73, pp. 1005-1006, 1976.

8. Reed, I. S., and Truong, T. K., "Fast Mersenne-Prime Transforms for Digital Filtering," *Proc. IEEE*, Vol. 125, No. 5, pp. 433-440, May 1978.

9. Reed, I. S., Truong, T. K., Miller, R. L., and Benjauthrit, B., "Further Results on Fast Transforms for Decoding Reed-Solomon Codes over $GF(2^m)$, for m = 4, 5, 6, 8," in *The Deep Space Network Progress Report 42-50*, pp. 132-154, Jet Propulsion Laboratory, Pasadena, Calif., Jan. 15, 1979.

10. Berlekamp, E. R., *Algebraic Coding Theory*, McGraw Hill, New York, 1968.