A Software Simulation Study of a (255,223) Reed-Solomon Encoder/Decoder

Fabrizio Pollara

April 15, 1985

NASA
National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California
A Software Simulation Study of a (255,223) Reed-Solomon Encoder/Decoder

Fabrizio Pollara

April 15, 1985

NASA
National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California
The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.
TABLE OF CONTENTS

1. INTRODUCTION ........................................... 1

2. SIMULATION SET-UP ........................................ 4

3. RS ENCODER ................................................ 4

4. RS DECODER ................................................ 6
   4.1 DECODER ALGORITHM ................................... 6
   4.2 EUCLID ALGORITHM ..................................... 9
   4.3 INVERSE FFT ........................................ 10

5. USER GUIDE AND EXAMPLES ............................... 11

6. REFERENCES ............................................... 14

APPENDICES

A. NUMBER OF MULTIPLICATIONS REQUIRED TO COMPUTE THE SYNDROME
   OF A RS CODE ........................................... A-1

B. RS ENCODER SUBROUTINE ................................... B-1

C. RS DECODER SUBROUTINE ................................... C-1

D. EXAMPLE OF OUTPUT ....................................... D-1

E. EXAMPLE OF RANDOMLY CHOSEN CODEWORD ................. E-1
TABLE OF CONTENTS (cont.)

FIGURES

1. Simulation Block Diagram ..................... 5
2. RS Encoder ...................................... 7

TABLES

1. Decimal Representation of Elements of $\text{GF}(2^8)$ ............... 2
2. Coefficients of Generator Polynomial .................. 3
ABSTRACT

A set of software programs which simulates a (255,223) Reed-Solomon encoder/decoder pair is described. The transform decoder algorithm uses a modified Euclid algorithm, and closely follows the pipeline architecture proposed for the hardware decoder. Uncorrectable error patterns are detected by a simple test, and the inverse transform is computed by a finite field FFT.

Numerical examples of the decoder operation are given for some test codewords, with and without errors. The use of the software package is briefly described.
1. INTRODUCTION

A (255,223) Reed-Solomon (RS) code has been adopted as the standard outer code for concatenated coding systems by NASA and by the European Space Agency (ESA) [1]. This particular RS code is defined in GF(2^8) by the following parameters:

\[ N = 255 \] = number of 8-bit symbols in a codeword (block)
\[ K = 223 \] = number of information symbols in a block
\[ T = N-K \] = number of parity symbols.

Such a code is capable of correcting up to \( T/2 = 16 \) symbol errors in a block. The generator polynomial \( g(x) \) of the code is given by,

\[
g(x) = \prod_{i=M}^{M+T+1} (x - \alpha^{G_i}) = \sum_{j=0}^{T} g_j x^j
\]

where \( M = 112 \), \( G = 11 \), and \( \alpha \) is a root of the primitive polynomial over GF(2)

\[ x^8 + x^7 + x^2 + x + 1 \]

Every element of \( GF(2^8) \) can be represented as a polynomial in \( \alpha \) over GF(2) of degree less than 8, as shown in Table 1.

The constant \( M \) is chosen so that the polynomial has symmetrical coefficients, i.e.,

\[ g_j = g_{T-j}, \ j = 0,1,\ldots,T \]

It is shown in [2] that this is true if \( M = 2^{8-1} - (T/2) = 112 \).

The constant \( G = 11 \) is chosen to minimize the bit-serial implementation complexity of the encoder [3]. The polynomial coefficients are shown in Table 2.
Table I. Decimal Representation of Elements of GF($2^r$)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>13</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$z = \alpha^n; x = \alpha^y$
Table 2. Coefficients of Generator Polynomial

<table>
<thead>
<tr>
<th>$g_0$</th>
<th>$g_32$</th>
<th>$\alpha^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1$</td>
<td>$g_{31}$</td>
<td>$\alpha^{249}$</td>
</tr>
<tr>
<td>$g_2$</td>
<td>$g_{30}$</td>
<td>$\alpha^{59}$</td>
</tr>
<tr>
<td>$g_3$</td>
<td>$g_{29}$</td>
<td>$\alpha^{66}$</td>
</tr>
<tr>
<td>$g_4$</td>
<td>$g_{28}$</td>
<td>$\alpha^4$</td>
</tr>
<tr>
<td>$g_5$</td>
<td>$g_{27}$</td>
<td>$\alpha^{43}$</td>
</tr>
<tr>
<td>$g_6$</td>
<td>$g_{26}$</td>
<td>$\alpha^{126}$</td>
</tr>
<tr>
<td>$g_7$</td>
<td>$g_{25}$</td>
<td>$\alpha^{251}$</td>
</tr>
<tr>
<td>$g_8$</td>
<td>$g_{24}$</td>
<td>$\alpha^{97}$</td>
</tr>
<tr>
<td>$g_9$</td>
<td>$g_{23}$</td>
<td>$\alpha^{30}$</td>
</tr>
<tr>
<td>$g_{10}$</td>
<td>$g_{22}$</td>
<td>$\alpha^3$</td>
</tr>
<tr>
<td>$g_{11}$</td>
<td>$g_{21}$</td>
<td>$\alpha^{213}$</td>
</tr>
<tr>
<td>$g_{12}$</td>
<td>$g_{20}$</td>
<td>$\alpha^{50}$</td>
</tr>
<tr>
<td>$g_{13}$</td>
<td>$g_{19}$</td>
<td>$\alpha^{66}$</td>
</tr>
<tr>
<td>$g_{14}$</td>
<td>$g_{18}$</td>
<td>$\alpha^{170}$</td>
</tr>
<tr>
<td>$g_{15}$</td>
<td>$g_{17}$</td>
<td>$\alpha^5$</td>
</tr>
<tr>
<td>$g_{16}$</td>
<td></td>
<td>$\alpha^{24}$</td>
</tr>
</tbody>
</table>
The algorithm used is a transform decoder as described in [4], which is based on a modified Euclid algorithm to compute the error locator polynomial. Therefore, this simulation can be used to verify the performance of the proposed pipeline hardware decoder.

The only modifications consist in adapting the algorithm to symmetric generator polynomials, using a finite field FFT (Fast Fourier Transform) to compute the error pattern, and testing for uncorrectable error patterns.

2. SIMULATION SET-UP

The set of software subroutines includes a random generator (gen.c) of sequences taken from {0,1}, a RS encoder (rscod.c), a RS decoder (rsdec.c), and a block (error.c) which computes bit and symbol error probabilities. These subroutines are called by a main program named "universe.c".

All programs are written in C-language on a VAX 750 computer. Figure 1(a) shows the block diagram of the simulation set-up. Channel errors are artificially introduced at the input of the RS decoder. If desired, the set-up may be modified to that of Fig. 1(b), where errors are produced by adding a sequence of random variables (for example Gaussian, if the subroutine "gauss.c" is used) to the encoded stream. Error bursts may be added with a separate subroutine, or by a concatenated, convolutional code and Viterbi decoding.

The modularity of the program allows the simulation of concatenated coding schemes to be described in a separate report.

3. RS ENCODER

Since we are considering a systematic RS code, the encoder will first output the K information symbols a_i. The T parity symbols are the coefficients b_i of the remainder polynomial \( b(x) = b_0 + b_1 x + \ldots + b_{T-1} x^{T-1} \), resulting from dividing the message polynomial \( x^T a(x) \) by the generator polynomial \( g(x) \), where \( a(x) = a_0 + a_1 x + \ldots + a_{K-1} x^{K-1} \).
(a) Fixed Error Pattern

(b) Random Error Pattern

Fig. 1. Simulation Block Diagram
This polynomial division can be easily implemented by a shift register divider, as shown in the logic diagram of Fig. 2, for the (255,223) RS code. Additions are to be interpreted modulo-two (exclusive-OR), multiplications in the field are performed by table look-up, where the table is automatically constructed during the first execution. The subroutine listing is shown in Appendix B.

The algorithm proceeds as follows:
(0) Initialize \(b_i = 0\), \(i=0,\ldots,T\)
(A) During the first 223 iterations \(0<j<223\):
   (1) get information symbol \(a_j\)
   (2) \(v = a_j + b_{T-1}\)
   (3) output \(z = a_j\)
(B) During last 32 iterations \(224<j<255\):
   (1) \(v = 0\)
   (2) output \(z = b_{T-1}\)
(C) For all \(j\)'s:
   (1) \(b_i = b_{i-1} + (g_i \times v), i=T-1,T-2,\ldots,1\)
   (2) \(b_0 = v\)

The encoder may be tested by forcing the generator to produce some given pattern whose corresponding codeword is known, and printing the output.

4. RS DECODER

4.1 DECODER ALGORITHM

The decoder performs the following basic operations:

- get received codeword
- compute syndrome
- obtain the error-locator polynomial by using the modified Euclid algorithm
- compute the remaining elements of the error sequence transform
- compute the inverse transform yielding the estimated error pattern.
Fig. 2. RS Encoder
Consider the generator polynomial in (1), and define:

\[ U = [u_0, u_1, \ldots, u_{N-1}] \]  = received codeword

where

\[ [u_0, u_1, \ldots, u_{N-T-1}] \]  = information symbols

\[ [u_{N-T}, \ldots, u_{N-1}] \]  = parity

\[ S = [s_0, s_1, \ldots, s_{T-1}] \]  = syndrome

\[ R = [r_0, r_1, \ldots, r_T] \]

\[ \lambda = [\lambda_0, \lambda_1, \ldots, \lambda_{T-1}] \]  (contains error-locator polynomial at last iteration)

\[ U = [u_0, u_1, \ldots, u_{T-1}] \]

\[ E = [E_0, E_1, \ldots, E_N] \]  = error pattern transform

\[ e = [e_0, e_1, \ldots, e_N] \]  = error pattern

\[ d(S) = \{ j : s_j \neq 0 \text{ and } s_{j+T} = 0, j < i \leq T \} \]

and similarly for \[ d(R) \]  and \[ d(\lambda) \].

Then the decoder algorithm can be described as follows:

1. get received codeword \[ U \]
2. compute the syndrome, (see Appendix A)

\[ S = 0 \]

\[ s_j = u_{N-1-j} + a^{G(j+1)} s_j ; j = 0, \ldots, T-1 ; i = 0, \ldots, N-1 \]

3. if \[ d(S) = 0 \] go to (11)
4. initialize,

\[ E_{j+1} = s_{T-1-j} ; j = 0, \ldots, T-1 \]

\[ R = 0 \]

\[ r_T = 1 \]

\[ \mu = 0 \]

\[ \mu_0 = 1 \]

\[ \lambda = 0 \]

\[ i = 1 \]

5. while \( i < T \) do:

\[ L = d(R) - d(S) \]

if \( d(R) < T/2 \) go to (6)

else if \( d(S) < T/2 \), \( \lambda = \mu \), go to (6)

else do EUCLID (see section 4.2)

\[ i = i + 1 \]
(6) compute normalized error-locator polynomial

$$-\lambda d(\lambda)$$

$$B = a.j \quad \lambda_j = B\lambda_j \quad j=0,\ldots,d(\lambda)$$

(7) compute remaining elements of error transform

$$E_{j+1} = 0$$

$$E_{j+1} = E_{j+1} + \lambda d(\lambda)_{-1-i} E_j \quad i=0,\ldots,d(\lambda)-1 \quad j=T,\ldots,N+T-1$$

$$E_0 = E_n$$

(8) test for uncorrectable error patterns,

if $$E_j \neq E_j+N$$ for some $$j=1,\ldots,T$$, go to (11)

(9) compute inverse transform $$e$$ (see section 4.3)

(10) compute corrected sequence,

$$U = U + e$$

(11) output $$U$$

A complete listing of the decoder subroutine is shown in Appendix C.

The test in step (8) is explained in [6]. It may also be observed that this RS code is effective in terms of undetected errors, since [7, 8], for independent symbol errors, the probability of undetected error $$P_u$$ is bounded by:

$$P_u < (N+1)^{-T} \sum_{i=0}^{T/2} \binom{N}{i} N^i < \frac{1}{(T/2)!}$$

For the (255,223) code, $$P_u < 4.8 \times 10^{-14}$$. But, for the (15,9) code considered in [4], $$P_u < 0.093$$.

4.2 EUCLID ALGORITHM

This is a modified version of Euclid's algorithm for polynomials [5], which does not need the computation of inverse field elements. It operates on two polynomials,

$$A(x) = x^T \quad \text{and} \quad S(x) = \sum_{k=1}^{T} a_k x^{T-k}$$
and finds the $i^{th}$ remainder $R_i(x)$ of degree less than $T/2$, satisfying:

$$
\gamma_i(x) \lambda_i(x) A(x) + \lambda_i(x) S(x) = R_i(x)
$$

At the end, $\lambda_j(x)$ is the desired (unnormalized) locator polynomial. The algorithm is implemented as follows:

\[
\begin{aligned}
&\text{if } d(R) < d(S) \quad \begin{cases} 
R &\leftarrow S \\
\lambda &\leftarrow \mu \\
d(R) &\leftarrow d(S)
\end{cases} \\
&\text{if } s_d(S) = 0 \quad \begin{cases} 
\mu &\leftarrow \mu \\
d(S) &\leftarrow d(S) - 1 \\
\text{if } d(S) < T/2, \lambda = \mu, \text{return}
\end{cases}
\end{aligned}
\]

\[
\begin{aligned}
&\text{else} \quad \begin{cases} 
a = r_{d(R)}; b = s_{d(S)} \\
d(R) &\leftarrow d(R) - 1 \\
\hat{S} &\leftarrow D_{|L|}(S) \\
\hat{R} &\leftarrow b \hat{R} + a \hat{S} \\
\hat{\mu} &\leftarrow D_{|L|}(\mu) \\
\hat{\lambda} &\leftarrow b \hat{\lambda} + a \hat{\mu} \\
\text{if } d(R) < T/2, \text{return}
\end{cases}
\end{aligned}
\]

where $D_{|L|}(x)$ shifts right the components of a vector $x$ by $|L|$ positions, and fills with zeros.

4.3 INVERSE FFT

A direct computation of the inverse transform,

$$
e_j = \sum_{i=0}^{N-1} a^{G_{ij}} E_{N+1+i+j}; \quad i=0,\ldots,N-1; \quad j=0,\ldots,N-1
$$

requires $N^2 = 65025$ multiplications. The number of multiplications may be reduced by organizing the $N$-point one-dimensional array $E$ into a two-dimensional $n_1 \times n_2$ array, where $n_1 n_2 = N$, and $n_1$ and $n_2$ are
relatively prime. This algorithm (Good-Thomas FFT [6]) is based on the Chinese remainder theorem.

Let \( b = (a)_{N} \) denote the remainder of \( a \) modulo \( N \), and define

\[
i_1 = (i)_{n_1}, \quad i_2 = (i)_{n_2}, \quad j_1 = (\tilde{n}j)_{n_1}, \quad \text{and} \quad j = (\tilde{n}j)_{n_2}
\]

Then,

\[
i = (\tilde{N}(n_2 i_1 + n_1 i_2))_N \quad \text{and} \quad j = (n_2 j_1 + n_1 j_2)_N
\]

where, \( (\tilde{N}(n_1 + n_2)) = 1 \quad \Rightarrow \quad \tilde{N} = 8 \).

Now the inverse transform may be written in the following two steps

\[
D_{i_1,j_2} = \sum_{i_2=0}^{n_2-1} E_{N+1-N+1}^{n_1+i_2} G_{n_1}^{i_2} j_2 \quad 0 < i_1 < n_1, \quad 0 < j_2 < n_2
\]

\[
e_j = \sum_{i_1=0}^{n_1-1} D_{i_1,j_2} G_{n_2}^{i_1} j_1 \quad 0 < j_1 < n_1, \quad 0 < j_2 < n_2
\]

For \( N = 255 = 17 \cdot 15 = n_1 n_2 \), the number of multiplications is reduced from \( N^2 \) to \( N(n_1 + n_2) \). A further reduction may be obtained, if desired, by factoring \( N \) as \( N = 17 \cdot 5 \cdot 3 \).

5. USER GUIDE AND EXAMPLES

This software package may be run on any computer having a C-language compiler. The source code for the full set of subroutines is available by contacting the author. Subroutines required are:

(block management routines in object code form)

- sequencer.o
- block.o
- fifo.o
(include files)

star.h
dstar.h
type.h
alloc.h
para.h
param.h
dfifo.h

(simulation blocks)

gen.c
rscod.c
rsdec.c
add.c
gauss.c
error.c
universe.c

Subroutines are also available to simulate the (15,9) RS code considered in [4]. If the UNIX operating system is used, it is advisable to create a "makefile" to maintain (compile and link) the set of subroutines. In any case the subroutines must be compiled and linked to produce an executable image file.

In order to run the simulation it is not necessary to provide any external parameter or data file, since the information symbols are generated internally. If specific information sequences are of interest, the subroutine "gen.c" can be easily modified for this purpose. Non-real-time decoding of actual data could be accomplished by modifying "rsdec.c" so that it will read the data from a disk file in segments of a given number of blocks.

The output contains the number of channel symbol errors, and the number of corrected symbol and bit errors. If the number of channel errors is greater than 7/2, a warning message is printed. If real data needs to be
decoded, the decoded symbols can be displayed by adding a print statement in "rsdec.c". All output is normally displayed on the standard output (CRT), but it can be redirected to a disk file through the operating system. As an example, under UNIX, we could type:

```
sim > outfile
```

where "sim" is the executable program and the file "outfile" will contain the output.

By including the print statements provided in the subroutine rsdec.c, it is possible to display all the intermediate steps of the decoder. Such an example of output is shown in Appendix D for a given codeword and the error pattern \(e_7 = a^{202}, e_{120} = a^0, e_i = 0, i \neq 7, 120\). Elements in \(GF(256)\) are represented in decimal base.

If no errors were present, the output would show that \(S = 0\).

A randomly chosen codeword is shown in Appendix E.
REFERENCES


APPENDIX A

Number of Multiplications Required to Compute the Syndrome of a (255,223) RS Code with a Symmetric Generator Polynomial
Consider the generator polynomial in \((1)\), then the syndrome is defined as,

\[
s_j = \sum_{i=0}^{N-1} u_i^\alpha G_i(j+M), \quad j=0,\ldots,T
\]

Define the \(N\times N\) matrix \(J\),

\[
J = \begin{bmatrix}
0 & 1 \\
1 & \ddots & 1 \\
& \ddots & \ddots & 1 \\
& & 1 & 0
\end{bmatrix}
\]

such that,

\[
\tilde{u} = [\tilde{u}_0, \ldots, \tilde{u}_{N-1}] = [u_{N-1}, \ldots, u_0] = u J
\]

Then we can consider a new syndrome \(\tilde{s}\)

\[
\tilde{s}_m = \sum_{i=0}^{N-1} \tilde{u}_i G_i(m+M), \quad m=0,\ldots,T-1
\]

\[
= \sum_{i=0}^{N-1} u_{N-1-i} G_i(m+M), \quad m=0,\ldots,T-1
\]

Let \(k = N - i - 1\)

\[
\tilde{s}_m = \sum_{k=0}^{N-1} u_k G(N-1-k)(m+M) = \alpha^{-G(m+M)} \sum_{k=0}^{N-1} u_k \alpha^{-G(m+M)}
\]

But,

\[
\alpha^{-G(m+M)} = \alpha^{G(h+M)}, \quad \text{where} \quad m = T - 1 - h, \quad h=0,\ldots,T-1
\]

\[
\tilde{s}_{T-1-h} = \alpha^{G(h+M)} \sum_{k=0}^{N-1} u_k \alpha^{G(h+M)}, \quad h=0,\ldots,T-1
\]
And finally, \( s_{T-1-h} = \alpha^{G(h+M)} s_h \), \( h=0,...,T-1 \)

\[
\tilde{S} = S \tilde{\Gamma}, \text{ where } \tilde{\Gamma} = \begin{bmatrix}
\alpha^G & 0 \\
\alpha^G & \alpha^G(M+1) \\
\vdots & \vdots \\
0 & \alpha^G(M+T-1)
\end{bmatrix}
\]

\( S = uA \),

where \( A = [a_{i,j}] \), \( a_{i,j} = a_{i,j}^G(M+1) \), \( i=0,...,N-1 \), \( j=0,...,T-1 \)

\[
\tilde{S} = uJ \Sigma
\]

\( S(J+\Gamma) = uAJ + uJAJ = u(AJ + JAJ) d \)

Let

\[
B = AJ + JAJ = \begin{bmatrix}
b_0 & \cdots & b_j & \cdots & b_{T-1}
\end{bmatrix} A
\begin{bmatrix}
\ldots & \begin{bmatrix}
b_j^* \\
0 \\
\vdots \\
\begin{bmatrix}b_{j+1}^* \\
Jb_{j+1}^*
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\]

be a partition of \( B \) into the column vectors \( b_j \), and

\( u = [u_1, u_c, u_d] \)

Then

\[
d_j = u_1 b_j^* + u_c b_j^* + u_d Jb_{j+1}^* = (u_1 + u_d) b_j^* + u_c b_{j+1}^*
\]

The computation of \( d_j \) requires only \( 254/2 + 1 = 128 \) multiplications (instead of 255).

\[
S = d (J+\Gamma)^{-1}
\]

Note that \( (J+\Gamma)^{-1} \) has the form:

\[
(J+\Gamma)^{-1} = \begin{bmatrix}
\beta_0 & 0 & \cdots & 0 \\
0 & \beta_1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\epsilon_0 & 0 & \cdots & \beta_{T-1}
\end{bmatrix}
\]

Therefore,

\[
S_j = \beta_j d_j + \epsilon_j d_{T-1-j} \quad j=0,...,T-1
\]

So that each \( S_j \) can be computed with \( 128 + 2 = 130 \) multiplications.
APPENDIX B

RS Encoder Subroutine
/* Reed-Solomon Encoder (CCSDS Doc. #1, Sept 1983) */

#include <stdio.h>
#include "../type.h"
#include "../star.h"
#define TT 32
#define N 255
#define K 223
#define CNT pstate->cnt
#define V pstate->vv
#define G pstate->g
#define B pstate->b
#define H pstate->h
#define F pstate->f

typedef struct {
    int non;
} PARAM, *PARAMPTR;
typedef struct {
    unsigned char b[TT], g[TT+1], cnt, h[N], f[N+1], vv;
} STATE, *STATEPTR;

rscode (pparam, size, pstate, psstar)
PARAMPTR pparam;
STATEPTR pstate;
STARPTR pstar;
int size;

{ SAMPLE x;
  int i, j;
  
  if (pstate == NULL) {
    pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
    if (no_input_fifos() || no_output_fifos() ! = 2)
      return(3);
  }

  /* H[ ] and F[ ] compute the power and log in GF(256) */

  H[0]=1;
  for(i = 0; i < 8; i++) H[i+1] = 2*H[i];
  for(i = 8; i < W; i++) H[i] = H[i-1]*H[i-6]^H[i-7]^H[i-8];

  for(j = 1; j < W+1; j++) {
    for(i = 0; i < W; i++) {
      if(H[i]==j) F[j]=1;
    }
  }

  CNT=0;
  V=0;
  F[0]=0;
/** G[ ] are the coefficients of the generating polynomial ***/

G[0]=H[0];
G[1]=H[2];
G[2]=H[9];
G[3]=H[66];
G[4]=H[4];
G[5]=H[43];
G[6]=H[126];
G[7]=H[251];
G[8]=H[97];
G[9]=H[30];
G[10]=H[3];
G[12]=H[50];
G[13]=H[66];
G[14]=H[170];
G[15]=H[5];
G[16]=H[24];

for(i=0;i<TT/2;i++) G[TT-i]=G[i];

if(length_output_fifo(0) != length_output_fifo(1)) return(7);
if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);

while(length_input_fifo(0) >0 || CNT> emphasis 5) {
    if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    if(length_output_fifo(1)==maxlength_output_fifo(1)) return(0);
    if(CNT=0) for(i=0;i<TT;i++) B[i]=0;
    if(CNT<K) //*** information bits ***/
    {
        get(0,&x);
        V=((int)x)^B[TT-1];
    }
    else //*** parity bits ***/
    {
        V=0;
        x=(SAMPLE)^B[TT-1];
    }
    for(i=TT-1;i>0;i--)
    B[i]=B[i-1]^((V1=x)^(H([G[i]]+F[V])%N));
    B[0]=((V1=x)^(H([G[0]]+F[V])%N));
    //***
    put(0,x);
    put(1,x);
    CNT=(CNT+1)%H;
}
return(0);
APPENDIX C

RS Decoder Subroutine
#include <stdio.h>
#include "../type.h"
#include "../star.h"
#define TT 32
#define N 255
#define M 112
#define G 11
#define H pstate->h
#define F pstate->f
#define MUL(A, B) ((B1=0) && ((H[(A+F[(B)])%N]))
typedef struct {
    int non;
} PARAM, *PARAMPTR;
typedef struct {
    unsigned char h[N], f[N+1];
} STAT, *STATEPTR;
rseec (pparam, size, pstate, pstar)
PARAMPTR pparam;
STATEPTR pstate;
STATPTR pstar;
int size;
{
    SAMPLE x;
    unsigned char et[17], ex[N], e[N], E[N+TT+1], S[TT+1], degR, degS;
    unsigned char R[TT+1], mu[TT+1], lam[TT+1], REC[N], tem, fa, fb;
    unsigned char *PR, *PS, *PT, *Plam, *Pmu, a, b;
    int i, j, L, CL, TH, ix, Jx, i1, i2, j1, j2;

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) != 1 || no_output_fifos( ) != 1)
            return(3);
    }

    /* H[ ] and F[ ] compute the power and log in GF(256) */

    H[0]=1;
    for(i=0; i<8; i++) H[i+1] = 2*H[i];
    for(i=8; i<N; i++) H[i] = (H[i-1] & H[i-6] & H[i-7] & H[i-8]);

    for(j=1; j<N+1; j++) {
        for(i=0; i<N; i++) {
            if(H[i]==j) F[j]=i;
        }
    }

    F[0]=0;
}
if (length_output_fifo(0) == max_length_output_fifo(0)) return(0);
        while (length_input_fifo(0) > 0 )
        {
        if (length_output_fifo(0) == max_length_output_fifo(0)) return(0);
    
  /***********************************************************/
  for(j=0; j<N; j++)
  {
  get(0, &x);
  REC[j] = (char)x;
  e[j] = 0;
  }

  PR=R;
  PS=S;
  Plam=lam;
  Pmu=mu;
  for(j=0; j<=TT; j++) { R[j] = 0; S[j] = 0; lam[j] = 0; mu[j] = 0; }
  /***********************************************************/
  for(i=0; i<N; i++)
  {
  ix=N-1-i;
  for(j=0; j<TT; j++) S[j] = REC[ix]^MUL(G*(j+M), S[j]);
  }

  degS=TT;
  while((degS+PS)==0 && degS>0) --degS;
  if (degS>0)
  {
  /****** Modified Euclid Algorithm  *************
  for(j=0; j<TT; j++) E[j+1] = (PS+TT-1-j);
  *(PR+TT)=1;
  #mu=1;
  degR=TT;
  degS=TT;
  i=1;
  TH=TT/2;
  
  while(i<=TT)
  {
  while((degR+PR)==0 && degR>0) --degR;
  while((degS+PS)==0 && degS>0) --degS;
  
  L=degR-degS;
  CL=L;
  if(L<0) CL = -L;
  if(degR<TH || degS<TH)
  {
  if(degR==TH) Plam=Pmu;
  break;
  }
  else
  
  C-3
{ if(degR < degS) {
  PT=PR;
  PR=PS;
  PS=PT;
  PT=Plam;
  Plam=Pmu;
  Pmu=PT;
  tem=degR;
  degR=degS;
  degS=tem;
}
if((PS+degS)==0) {
  degS--;
  if(degS<TH) {
    Plam=Pmu;
    break;
  }
}
else {
  /* compute R lam */
  a= *(PR+degR);
  b= *(PS+degS);
  fa=F[a];
  fb=F[b];
  degR--;
  for(j=0; j<=TT; j++) {
    tem=(j>CL)*(*(PS+j-CL));
    PT=PR+j;
    #PT=(b!=0)*MUL(fb, #PT)*(a!=0)*MUL(fa, tem);
    tem=(j>CL)*(*(Pmu+j-CL));
    PT=Plam+j;
    #PT=(b!=0)*MUL(fb, #PT)*(a!=0)*MUL(fa, tem);
  }
  if(degR<TH) break;
}
}
i++;
} /************ Error locator polynomial *************/
degR=TT;
while(*(degR+Plam)==0 && degR>0) --degR;

tem=N-F[*(Plam+degR)];
for(j=0; j<=degR; j++) {
  PT=Plam+j;
  *PT=MUL(tem, *PT);
}
for(j=TT; j<N+TT; j++)
{
    E[j+1]=0;
    for(i=0; i<degr; i++)
    {
        tem = *(Plam+degr-i-1);
        jx = j-i;
        if(temi=0) E[j+1] ^= MUL(F[tem], E[jx]);
    }
}
E[0]=E[N];
for(j=1; j<=TT; j++)
    if(E[j] != E[j+N]) {printf("1n * TEST FAILED ****"); j=0; break;}
if(j1=0)
{
    /*****************************************************/
    for(j2=0; j2<15; j2++)
    {
        jx=G*17*j2;
        for(i1=0; i1<17; i1++)
        {
            et[i1]=0;
            for(i2=0; i2<15; i2++)
            {
                i=(N+1-M+8*(15*i1+17*i2))%N;
                et[i1] ^= MUL(jx*i2,E[i]);
            }
        }
        for(j1=0; j1<17; j1++)
        {
            ix=G*15*j1;
            j=(15*j1+17*j2)%N;
            e[j]=0;
            for(i1=0; i1<17; i1++)
            {
                e[j] ^= MUL(ix*i1,et[i1]);
            }
        }
    }
    for(j=0; j<N; j++) REC[j] ^= e[j];
}
}
for(j=0; j<N; j++)
{
    x=(SAMPLE)REC[j];
    put(0,x);
}
return (0);
APPENDIX D

Example of Output
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td>37</td>
<td>0</td>
<td>38</td>
<td>0</td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>42</td>
<td>0</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
<td>45</td>
<td>0</td>
<td>46</td>
<td>0</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>49</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>0</td>
<td>53</td>
<td>0</td>
<td>54</td>
<td>0</td>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>57</td>
<td>0</td>
<td>58</td>
<td>0</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>62</td>
<td>0</td>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>65</td>
<td>0</td>
<td>66</td>
<td>0</td>
<td>67</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>0</td>
<td>69</td>
<td>0</td>
<td>70</td>
<td>0</td>
<td>71</td>
<td>0</td>
</tr>
<tr>
<td>72</td>
<td>0</td>
<td>73</td>
<td>0</td>
<td>74</td>
<td>0</td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>0</td>
<td>77</td>
<td>0</td>
<td>78</td>
<td>0</td>
<td>79</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>81</td>
<td>0</td>
<td>82</td>
<td>0</td>
<td>83</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>0</td>
<td>85</td>
<td>0</td>
<td>86</td>
<td>0</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>88</td>
<td>0</td>
<td>89</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>0</td>
<td>93</td>
<td>0</td>
<td>94</td>
<td>0</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>96</td>
<td>0</td>
<td>97</td>
<td>0</td>
<td>98</td>
<td>0</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>101</td>
<td>0</td>
<td>102</td>
<td>0</td>
<td>103</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>0</td>
<td>105</td>
<td>0</td>
<td>106</td>
<td>0</td>
<td>107</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>0</td>
<td>109</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>111</td>
<td>0</td>
</tr>
<tr>
<td>112</td>
<td>0</td>
<td>113</td>
<td>0</td>
<td>114</td>
<td>0</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>0</td>
<td>117</td>
<td>0</td>
<td>118</td>
<td>0</td>
<td>119</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>121</td>
<td>0</td>
<td>122</td>
<td>0</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>0</td>
<td>125</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>127</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>129</td>
<td>0</td>
<td>130</td>
<td>0</td>
<td>131</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>0</td>
<td>133</td>
<td>0</td>
<td>134</td>
<td>0</td>
<td>135</td>
<td>0</td>
</tr>
<tr>
<td>136</td>
<td>0</td>
<td>137</td>
<td>0</td>
<td>138</td>
<td>0</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>141</td>
<td>0</td>
<td>142</td>
<td>0</td>
<td>143</td>
<td>0</td>
</tr>
<tr>
<td>144</td>
<td>0</td>
<td>145</td>
<td>0</td>
<td>146</td>
<td>0</td>
<td>147</td>
<td>0</td>
</tr>
<tr>
<td>148</td>
<td>0</td>
<td>149</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>151</td>
<td>0</td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>153</td>
<td>0</td>
<td>154</td>
<td>0</td>
<td>155</td>
<td>0</td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>157</td>
<td>0</td>
<td>158</td>
<td>0</td>
<td>159</td>
<td>0</td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>161</td>
<td>0</td>
<td>162</td>
<td>0</td>
<td>163</td>
<td>0</td>
</tr>
<tr>
<td>164</td>
<td>0</td>
<td>165</td>
<td>0</td>
<td>166</td>
<td>0</td>
<td>167</td>
<td>0</td>
</tr>
<tr>
<td>168</td>
<td>0</td>
<td>169</td>
<td>0</td>
<td>170</td>
<td>0</td>
<td>171</td>
<td>0</td>
</tr>
<tr>
<td>172</td>
<td>0</td>
<td>173</td>
<td>0</td>
<td>174</td>
<td>0</td>
<td>175</td>
<td>0</td>
</tr>
<tr>
<td>176</td>
<td>0</td>
<td>177</td>
<td>0</td>
<td>178</td>
<td>0</td>
<td>179</td>
<td>0</td>
</tr>
<tr>
<td>180</td>
<td>0</td>
<td>181</td>
<td>0</td>
<td>182</td>
<td>0</td>
<td>183</td>
<td>0</td>
</tr>
<tr>
<td>184</td>
<td>0</td>
<td>185</td>
<td>0</td>
<td>186</td>
<td>0</td>
<td>187</td>
<td>0</td>
</tr>
<tr>
<td>188</td>
<td>0</td>
<td>189</td>
<td>0</td>
<td>190</td>
<td>0</td>
<td>191</td>
<td>0</td>
</tr>
<tr>
<td>192</td>
<td>0</td>
<td>193</td>
<td>0</td>
<td>194</td>
<td>0</td>
<td>195</td>
<td>0</td>
</tr>
<tr>
<td>196</td>
<td>0</td>
<td>197</td>
<td>0</td>
<td>198</td>
<td>0</td>
<td>199</td>
<td>0</td>
</tr>
</tbody>
</table>

(Codeword)
No. of input errors = 2

\[ i \quad s \quad \log() \]

0  16  4  (Syndrome)
1  117  136
2  121  184
3  191  134
4  113  24
5  55  53
6  187  74
7  243  96
8  248  246
9  218  239
10  100  223
11  66  88
12  233  83
13  30  43
14  170  85
15  151  145
16  58  33
17  190  22
18  238  248
19  33  87
20  130  156
21  254  60
22  209  171
23  133  159
24  165  30
25  188  182
26  21  212
27  107  63
28  241  75
29  74  173
30  37  172
31  16  4

\[ i = 1 \]
\[ d(R) = 32 \quad d(S) = 31 \quad L = 1 \]
\[ a = 0 \quad b = 4 \]

D-3
<table>
<thead>
<tr>
<th>i</th>
<th>s</th>
<th>log()</th>
<th>r</th>
<th>log()</th>
<th>log()</th>
<th>u</th>
<th>log()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>4</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>117</td>
<td>136</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td>121</td>
<td>184</td>
<td>117</td>
<td>136</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>191</td>
<td>134</td>
<td>121</td>
<td>184</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>113</td>
<td>24</td>
<td>191</td>
<td>134</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>55</td>
<td>53</td>
<td>113</td>
<td>24</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>187</td>
<td>74</td>
<td>55</td>
<td>53</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>243</td>
<td>96</td>
<td>187</td>
<td>74</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>248</td>
<td>246</td>
<td>243</td>
<td>96</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>218</td>
<td>239</td>
<td>248</td>
<td>246</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>223</td>
<td>218</td>
<td>239</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>66</td>
<td>88</td>
<td>100</td>
<td>223</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>233</td>
<td>83</td>
<td>66</td>
<td>88</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>43</td>
<td>233</td>
<td>83</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>170</td>
<td>85</td>
<td>30</td>
<td>43</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>151</td>
<td>145</td>
<td>170</td>
<td>85</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>58</td>
<td>33</td>
<td>151</td>
<td>145</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>190</td>
<td>22</td>
<td>58</td>
<td>33</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>238</td>
<td>248</td>
<td>190</td>
<td>22</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>33</td>
<td>87</td>
<td>238</td>
<td>248</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>130</td>
<td>156</td>
<td>33</td>
<td>87</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>254</td>
<td>60</td>
<td>130</td>
<td>156</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>209</td>
<td>11</td>
<td>254</td>
<td>60</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>133</td>
<td>159</td>
<td>209</td>
<td>11</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>165</td>
<td>30</td>
<td>133</td>
<td>159</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>188</td>
<td>182</td>
<td>165</td>
<td>30</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>21</td>
<td>212</td>
<td>188</td>
<td>182</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>107</td>
<td>63</td>
<td>21</td>
<td>212</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>241</td>
<td>75</td>
<td>107</td>
<td>63</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>74</td>
<td>173</td>
<td>241</td>
<td>75</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>37</td>
<td>172</td>
<td>74</td>
<td>173</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>16</td>
<td>4</td>
<td>37</td>
<td>172</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
</tbody>
</table>

i = 2
d(R)= 31  d(S)= 31  L= 0
a=172  b= 4

<p>| 0  | 16 | 4     | 217| 176  | 37 | 172 | 1  | 0  |
| 1  | 117| 136   | 176| 192  | 16 | 4   | 0  | *  |
| 2  | 121| 184   | 199| 204  | 0  | *   | 0  | *  |
| 3  | 191| 134   | 103| 47   | 0  | *   | 0  | *  |
| 4  | 113| 24    | 240| 46   | 0  | *   | 0  | *  |
| 5  | 55 | 53    | 156| 133  | 0  | *   | 0  | *  |
| 6  | 187| 74    | 134| 27   | 0  | *   | 0  | *  |
| 7  | 243| 96    | 46 | 226  | 0  | *   | 0  | *  |
| 8  | 248| 246   | 251| 23   | 0  | *   | 0  | *  |
| 9  | 218| 239   | 52 | 128  | 0  | *   | 0  | *  |
| 10 | 100| 223   | 212| 179  | 0  | *   | 0  | *  |
| 11 | 66 | 88    | 124| 245  | 0  | *   | 0  | *  |
| 12 | 233| 83    | 180| 150  | 0  | *   | 0  | *  |
| 13 | 30 | 43    | 137| 9    | 0  | *   | 0  | *  |</p>
<table>
<thead>
<tr>
<th>i</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>170.85</td>
<td>151.145</td>
<td>58.33</td>
<td>190.22</td>
<td>238.248</td>
<td>33.87</td>
<td>130.156</td>
<td>254.60</td>
<td>209.171</td>
<td>173.159</td>
<td>115.30</td>
<td>188.182</td>
<td>21.212</td>
<td>107.63</td>
<td>241.75</td>
<td>74.173</td>
<td>37.172</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ i=3 \]
\[ d(R)=30 \]
\[ d(S)=31 \]
\[ L=-1 \]
\[ a=4 \]
\[ b=115 \]

<table>
<thead>
<tr>
<th>0</th>
<th>179.119</th>
<th>217.176</th>
<th>37.172</th>
<th>177.115</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>196.41</td>
<td>176.192</td>
<td>16.4</td>
<td>1217.176</td>
</tr>
<tr>
<td>2</td>
<td>159.235</td>
<td>199.204</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>19.78</td>
<td>103.47</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>202.77</td>
<td>240.46</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>125.164</td>
<td>156.133</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>252.58</td>
<td>134.27</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4.2</td>
<td>46.226</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>110.54</td>
<td>251.23</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>133.159</td>
<td>52.128</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>167.210</td>
<td>212.179</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>95.21</td>
<td>124.245</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>94.181</td>
<td>180.150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>98.40</td>
<td>137.9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>41.217</td>
<td>99.186</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>12.101</td>
<td>114.70</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>150.169</td>
<td>83.138</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>200.224</td>
<td>231.193</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>221.12</td>
<td>185.236</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>99.186</td>
<td>65.155</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>234.137</td>
<td>7.106</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>223.68</td>
<td>174.37</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>9.205</td>
<td>148.174</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>28.108</td>
<td>202.77</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>15.42</td>
<td>173.11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>251.23</td>
<td>119.247</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>164.219</td>
<td>11.188</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>218.23</td>
<td>72.208</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>57.69</td>
<td>219.38</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>53.177</td>
<td>169.146</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>169.146</td>
<td>177.115</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ D-5 \]
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

i = 4

d(R) = 30  d(S) = 30  L = 0

a = 146  b = 115

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32</td>
<td>5</td>
<td>217</td>
<td>176</td>
</tr>
<tr>
<td>1</td>
<td>107</td>
<td>63</td>
<td>176</td>
<td>192</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>*</td>
<td>199</td>
<td>204</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>*</td>
<td>103</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>*</td>
<td>240</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>*</td>
<td>156</td>
<td>133</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>*</td>
<td>134</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>*</td>
<td>46</td>
<td>226</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>*</td>
<td>251</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>*</td>
<td>52</td>
<td>128</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>*</td>
<td>212</td>
<td>179</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>*</td>
<td>124</td>
<td>245</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>*</td>
<td>180</td>
<td>150</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>*</td>
<td>137</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>*</td>
<td>99</td>
<td>186</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>*</td>
<td>114</td>
<td>70</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>*</td>
<td>83</td>
<td>138</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>*</td>
<td>231</td>
<td>193</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>*</td>
<td>185</td>
<td>236</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>*</td>
<td>65</td>
<td>155</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>*</td>
<td>7</td>
<td>106</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>*</td>
<td>174</td>
<td>37</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>*</td>
<td>148</td>
<td>174</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>*</td>
<td>202</td>
<td>77</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>*</td>
<td>173</td>
<td>11</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>*</td>
<td>119</td>
<td>247</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>*</td>
<td>11</td>
<td>188</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>*</td>
<td>72</td>
<td>208</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>*</td>
<td>219</td>
<td>38</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>*</td>
<td>169</td>
<td>146</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>*</td>
<td>177</td>
<td>115</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>*</td>
</tr>
</tbody>
</table>

i = 5

d(R) = 1  d(S) = 30  L = -29

1  o log()

0 37 172  (Error-locator polynomial)

D-6
\[
\begin{array}{cccccccccc}
1 & e log & 1 & e log & 1 & e log & 1 & e log & 1 & e log \\
0 & 0 & 0 & 1 & 0 & 2 & 0 & 3 & 0 & (Error) \\
4 & 0 & 5 & 0 & 6 & 0 & 7 & 0 & 80 & 202 \\
8 & 0 & 9 & 0 & 10 & 0 & 11 & 0 & \\
12 & 0 & 13 & 0 & 14 & 0 & 15 & 0 & \\
16 & 0 & 17 & 0 & 18 & 0 & 19 & 0 & \\
20 & 0 & 21 & 0 & 22 & 0 & 23 & 0 & \\
24 & 0 & 25 & 0 & 26 & 0 & 27 & 0 & \\
28 & 0 & 29 & 0 & 30 & 0 & 31 & 0 & \\
32 & 0 & 33 & 0 & 34 & 0 & 35 & 0 & \\
36 & 0 & 37 & 0 & 38 & 0 & 39 & 0 & \\
40 & 0 & 41 & 0 & 42 & 0 & 43 & 0 & \\
44 & 0 & 45 & 0 & 46 & 0 & 47 & 0 & \\
48 & 0 & 49 & 0 & 50 & 0 & 51 & 0 & \\
52 & 0 & 53 & 0 & 54 & 0 & 55 & 0 & \\
56 & 0 & 57 & 0 & 58 & 0 & 59 & 0 & \\
60 & 0 & 61 & 0 & 62 & 0 & 63 & 0 & \\
64 & 0 & 65 & 0 & 66 & 0 & 67 & 0 & \\
68 & 0 & 69 & 0 & 70 & 0 & 71 & 0 & \\
72 & 0 & 73 & 0 & 74 & 0 & 75 & 0 & \\
76 & 0 & 77 & 0 & 78 & 0 & 79 & 0 & \\
80 & 0 & 81 & 0 & 82 & 0 & 83 & 0 & \\
84 & 0 & 85 & 0 & 86 & 0 & 87 & 0 & \\
88 & 0 & 89 & 0 & 90 & 0 & 91 & 0 & \\
92 & 0 & 93 & 0 & 94 & 0 & 95 & 0 & \\
96 & 0 & 97 & 0 & 98 & 0 & 99 & 0 & \\
100 & 0 & 101 & 0 & 102 & 0 & 103 & 0 & \\
104 & 0 & 105 & 0 & 106 & 0 & 107 & 0 & \\
108 & 0 & 109 & 0 & 110 & 0 & 111 & 0 & \\
\end{array}
\]
APPENDIX E

Example of Output for Randomly Chosen Codeword
\[
\begin{array}{cccccccc}
\text{i} & \text{u} & \text{i} & \text{u} & \text{i} & \text{u} & \text{i} & \text{u} \\
0 & 2 & 143 & 1 & 104 & 3 & 10 & \text{(Codeword)} \\
4 & 157 & 5 & 95 & 6 & 32 & 7 & 68 \\
8 & 139 & 9 & 200 & 10 & 143 & 11 & 186 \\
12 & 130 & 13 & 130 & 14 & 240 & 15 & 26 \\
16 & 124 & 17 & 179 & 18 & 133 & 19 & 176 \\
20 & 222 & 21 & 164 & 22 & 4 & 23 & 211 \\
24 & 42 & 25 & 248 & 26 & 131 & 27 & 219 \\
28 & 37 & 29 & 92 & 30 & 227 & 31 & 151 \\
32 & 135 & 33 & 187 & 34 & 189 & 35 & 61 \\
36 & 36 & 37 & 246 & 38 & 205 & 39 & 227 \\
40 & 155 & 41 & 19 & 42 & 223 & 43 & 111 \\
44 & 130 & 45 & 245 & 46 & 58 & 47 & 12 \\
48 & 13 & 49 & 141 & 50 & 183 & 51 & 31 \\
52 & 65 & 53 & 144 & 54 & 96 & 55 & 187 \\
56 & 154 & 57 & 172 & 58 & 239 & 59 & 148 \\
60 & 59 & 61 & 55 & 62 & 172 & 63 & 113 \\
64 & 154 & 65 & 106 & 66 & 26 & 67 & 38 \\
68 & 167 & 69 & 14 & 70 & 69 & 71 & 3 \\
72 & 128 & 73 & 182 & 74 & 165 & 75 & 201 \\
76 & 148 & 77 & 111 & 78 & 142 & 79 & 30 \\
80 & 88 & 81 & 246 & 82 & 29 & 83 & 130 \\
84 & 107 & 85 & 106 & 86 & 162 & 87 & 197 \\
88 & 72 & 89 & 133 & 90 & 141 & 91 & 236 \\
92 & 111 & 93 & 71 & 94 & 216 & 95 & 201 \\
96 & 17 & 97 & 52 & 98 & 246 & 99 & 192 \\
100 & 62 & 101 & 1 & 102 & 62 & 102 & 95 \\
104 & 141 & 105 & 203 & 106 & 215 & 107 & 66 \\
108 & 79 & 109 & 203 & 110 & 32 & 111 & 138 \\
112 & 49 & 113 & 136 & 114 & 165 & 115 & 209 \\
116 & 115 & 117 & 141 & 118 & 129 & 119 & 162 \\
120 & 139 & 121 & 157 & 122 & 81 & 123 & 100 \\
124 & 86 & 125 & 101 & 126 & 158 & 127 & 215 \\
128 & 195 & 129 & 180 & 130 & 199 & 131 & 5 \\
132 & 253 & 133 & 41 & 134 & 111 & 135 & 177 \\
136 & 27 & 137 & 109 & 138 & 107 & 139 & 84 \\
140 & 72 & 141 & 226 & 142 & 39 & 143 & 138 \\
144 & 112 & 145 & 220 & 146 & 156 & 147 & 9 \\
148 & 111 & 149 & 61 & 150 & 177 & 151 & 20 \\
152 & 165 & 153 & 14 & 154 & 50 & 155 & 112 \\
156 & 135 & 157 & 108 & 158 & 52 & 159 & 216 \\
160 & 133 & 161 & 132 & 162 & 2 & 163 & 237 \\
164 & 252 & 165 & 224 & 166 & 142 & 167 & 178 \\
168 & 126 & 169 & 180 & 170 & 87 & 171 & 121 \\
172 & 21 & 173 & 143 & 174 & 217 & 175 & 88 \\
176 & 234 & 177 & 142 & 178 & 120 & 179 & 33 \\
180 & 117 & 181 & 17 & 182 & 235 & 183 & 211 \\
184 & 39 & 185 & 243 & 186 & 39 & 187 & 140 \\
188 & 151 & 189 & 54 & 190 & 207 & 191 & 3 \\
192 & 45 & 193 & 61 & 194 & 30 & 195 & 116 \\
196 & 79 & 197 & 128 & 198 & 79 & 199 & 29 \\
200 & 13 & 201 & 189 & 202 & 145 & 203 & 43 \\
\end{array}
\]
| 204 77 | 205 172 | 206 108 | 207 47 |
| 208 118 | 209 54 | 210 177 | 211 22 |
| 212 155 | 213 40 | 214 226 | 215 153 |
| 216 44 | 217 101 | 218 37 | 219 51 |
| 220 28 | 221 156 | 222 167 | 223 175 |
| 224 80 | 225 108 | 226 20 | 227 122 |
| 228 202 | 229 251 | 230 31 | 231 81 |
| 232 29 | 233 89 | 234 159 | 235 134 |
| 236 60 | 237 217 | 238 91 | 239 13 |
| 240 243 | 241 103 | 242 83 | 243 142 |
| 244 162 | 245 69 | 246 174 | 247 177 |
| 248 55 | 249 20 | 250 163 | 251 108 |
| 252 191 | 253 74 | 254 141 |
A set of software programs which simulates a (255,223) Reed-Solomon encoder/decoder pair is described. The transform decoder algorithm uses a modified Euclid algorithm, and closely follows the pipeline architecture proposed for the hardware decoder. Uncorrectable error patterns are detected by a simple test, and the inverse transform is computed by a finite field FFT.

Numerical examples of the decoder operation are given for some test codewords, with and without errors. The use of the software package is briefly described.