JPL PUBLICATION 82-84

The VLSI Design of a Single Chip Reed-Solomon Encoder

T.K. Truong
L.J. Deutsch
Jet Propulsion Laboratory

I.S. Reed
University of Southern California

November 15, 1982

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 82-84

The VLSI Design of a Single Chip Reed-Solomon Encoder

T.K. Truong
L.J. Deutsch
Jet Propulsion Laboratory

I.S. Reed
University of Southern California

November 15, 1982

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 contract with the National Aeronautics and Space Administration.

ACKNOWLEDGMENTS

The authors wish to acknowledge I. S. Hsu, K. Wang, and C. S. Yeh of the University of Southern California VLSI laboratory for their support in design work and in preparation of this manuscript. We also wish to acknowledge J. W. Heller of JPL for his guidance and advice during the course of this effort and M. Perlman who brought the Berlekamp algorithm to our attention and worked with Berlekamp to produce the prototype discrete IC decoder upon which this chip is modeled.

ABSTRACT

This document presents a design for a single chip implementation of a Reed-Solomon encoder. The code used is the NASA and ESA (Europaan Space Agency) standard (255, 223) code. The architecture that leads to this single VLSI chip design makes use of a bit-serial finite field multiplication algorithm due to E. R. Berlekamp.

CONTENTS

APPENDICES

TABLES

CONTENTS (continued)

FIGURES

I. INTRODUCTION

A concatenated coding system consisting of a convolutional inner code and a Reed-Solomon outer code has been adopted as a guideline for future space missions by both the European Space Agency (ESA) and NASA [1]. The convolutional inner code is the same (7, 1/2) code used by the Voyager project. The outer Reed-Solomon code is a (255, 223) block code on 8-bit symbols and it is capable of correcting up to 16 symbol errors. The performance of such schemes is investigated in [2] where it is shown that this concatenated channel provides a coding gain of almost 2 dB over the convolutional-only channel at a decoded bit error rate of 10^{-5}. Hardware simulations of the concatenated channel were reported in [3]. One of the benefits of concatenated coding, and one of the main motivations for its acceptance as a standard system, is that it provides for a nearly error free communications link at fairly low power levels. This means that source data compression techniques[4] can be used to help increase channel throughput without a substantial change in overall error rate.

It is the purpose of this report to present a design for a single chip encoder for the outer Reed-Solomon code. This encoder would represent a considerable space, weight, and power savings over the smallest encoder (about 30 chips) currently available.

A Reed-Solomon encoder is basically a circuit that performs polynomial division in a finite field. Such circuits are well known [5] and their implementation is straightforward. The major problem in designing a small encoder is the large quantity of hardware that is necessary to perform the finite field multiplications. A conventional encoder as described in [5] for the (255, 223) code requires 32 finite field multipliers. These multipliers are usually implemented as either full parallel multipliers or table look-up multipliers. The use of either of these multiplication algorithms would prohibit the implementation of the encoder on a single medium density VLSI chip.

Fortunately, E. R. Berlekamp has developed a serial algorithm for this finite field multiplication [6]. This multiplication algorithm has enabled the design of a workable VLSI architecture consisting of only about 3,000 transistors.

Section II of this report contains a brief overview of Reed-Solomon coding and some important concepts of finite field arithmetic. Berlekamp's multiplication algorithm is described in section III and an example is worked in section IV. The architecture for the (255, 223) Reed-Solomon encoder is developed in section V with the actual layout shown in appendix C.

II. A REVIEW OF REED-SOLOMON CODES

It is assumed in this section and in section III that the reader is familiar with the basics of finite field theory. The necessary material may be found in [7].

Let GF(2^m) be the finite field with 2^m elements. A Reed-Solomon (RS) codeword is a sequence of elements (called RS symbols) in GF(2^m). This sequence can be considered to be the coefficients of a polynomial. Hence, an RS codeword may be represented as

$$(x) = \sum_{i=0}^{n-1} c_i x^i \qquad (1)$$

where $c_i \in$ GF(2^m), and x is an indeterminant.

The parameters of an RS code are summarized as follows:

m = number of bits per symbol

n = 2^m-1 = the length of a codeword in symbols

t = maximum number of error symbols that can be corrected

d = 2t+1 = design distance

2t = number of check symbols

k = n-2t = number of information symbols

In the NASA-ESA Standard RS code, m=8, n=255, t=16, d=33, 2t=32, and k=223. This code is known as a (255, 223) RS code.

The generator polynomial of an RS code is defined by

$$g(x) = \sum_{j=b}^{b+2t-1} (x - \gamma^j) = \sum_{i=0}^{2t} g_i x^i \qquad (2)$$

where b is a nonnegative integer, usually chosen to be 1, and γ is a primitive element in GF(2^m). In order to reduce the complexity of the encoder it is desirable to make the coefficients of g(x) symmetric so that $g(x) = x^{-d-1}\, g(1/x)$. To accomplish this b must be chosen to satisfy $2b+d-2 = 2^m-1$. Thus for the standard RS code, b=112.

Let $I(x) = c_{2t}x^{2t} + c_{2t+1}x^{2t+1} + \ldots + c_{n-1}x^{n-1}$ and $P(x) = c_0 + c_1x + \ldots + c_{2t-1}x^{2t-1}$ be the information polynomial and the check polynomial, respectively. Then the encoded RS code polynomial is represented by

$$C(x) = I(x) + P(x). \qquad (3)$$

To be an RS codeword, C(x) must be also a multiple of g(x). That is,

$$C(x) = q(x)g(x). \qquad (4)$$

An RS encoder must find P(x) in eq. (3) such that eq. (4) is true. It does this by dividing I(x) by g(x). The division algorithm yields

$$I(x) = q(x)g(x) + r(x), \tag{5}$$

where $r(x)$ is a remainder polynomial of degree less than 32. If $r(x) = -P(x)$, then by eq. (5)

$$q(x)g(x) = I(x) - r(x) = I(x) + P(x) = C(x). \tag{6}$$

Since the field $GF(2^m)$ has characteristic two, $-P(x) = P(x)$ so that $r(x) = P(x)$ and the operation of encoding is seen to consist of determining the remainder polynomial $r(x) = P(x)$.

Figure 1 shows the structure of a t-error correcting RS encoder over $GF(2^m)$. In Fig. 1, R_i $(0 \leq i \leq 2t-1)$ and Q are m-bit shift registers. Initially all these registers are set to zero, and both switches (controlled by the signal SL) are set to position A.

The information symbols $c_{n-1}, \ldots, c_{2t}$ are fed into the division circuit of the encoder and are also transmitted out of the encoder one-by-one. The quotient coefficients are generated and loaded into the Q register sequentially. The remainder coefficients are computed successively. Immediately after c_{2t} is fed to the circuit, both switches are set to position B. At the very same moment c_{2t-1} is computed and transmitted. Simultaneously, c_i is being computed and loaded into register R_i for each i. Next $c_{2t-2}, \ldots, c_0$ are transmitted out of the encoder one-by-one. The values of $c_{2t-2}, \ldots, c_0$ remain unchanged because the contents of Q are set to zero when the upper switch is at position B.

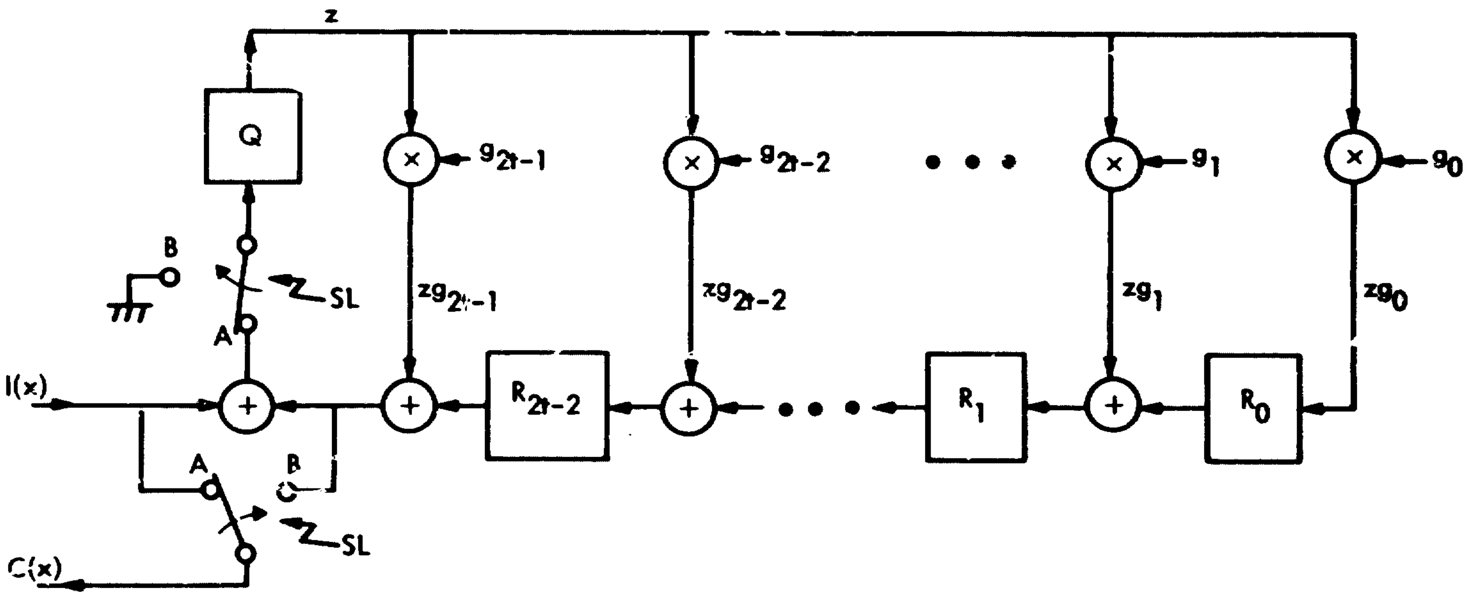


Figure 1. Block diagram of an encoder for a t-error correcting RS code

As remarked in the introduction, the complexity of an RS encoder design results from the need to compute the products zg_i $(0 \leq i \leq 2t-2)$. These computations can be performed in several ways. The Berlekamp algorithm for a bit-serial multiplier has the features needed to create a good pipeline architecture for VLSI implementation. In this report, Berlekamp's method is applied to the design of a (255, 223) RS-encoder, which can be implemented on a single VLSI chip.

III. BERLEKAMP'S MULTIPLICATION ALGORITHM

In order to understand Berlekamp's multiplication algorithm, some mathematical preliminaries are needed. Toward this end, the mathematical concepts of the "trace" and a "complementary (or dual) basis" are introduced. For more details and proofs see [8] and [9].

Definition 1: The "trace" of an element β belonging to $GF(p^m)$, the Galois field of p^m elements, is defined as follows:

$$Tr(\beta) = \sum_{k=0}^{m-1} \beta^{p^k} .$$

In particular, for p = 2,

$$\mathrm{Tr}(\beta) = \sum_{k=0}^{m-1} \beta^{2^k} .$$

The trace has the following properties, which will not be proven here:

(1) $[\mathrm{Tr}(\beta)]^p = \beta + \beta^p + \ldots + \beta^{p^{m-1}} = \mathrm{Tr}(\beta)$, where $\beta \in GF(p^m)$. This implies that $\mathrm{Tr}(\beta) \in GF(p)$, i.e., the trace is in the ground field $GF(p)$.

(2) $\mathrm{Tr}(\beta + r) = \mathrm{Tr}(\beta) + \mathrm{Tr}(r)$, where β, $r \in GF(p^m)$

(3) $\mathrm{Tr}(c\beta) = c\mathrm{Tr}(\beta)$, where $c \in GF(p)$.

(4) $\mathrm{Tr}(1) \equiv m(\mathrm{mod}\ p)$.

<u>Definition 2</u>: A "basis" $\{\mu_j\}$ in $GF(p^m)$ is a set of m linearly independent elements in $GF(p^m)$.

<u>Definition 3</u>: Two bases $\{\mu_j\}$ and $\{\lambda_k\}$ are said to be "complementary" or the "dual" of one another if

$$\mathrm{Tr}(\mu_j \lambda_k) = \begin{cases} 1, \text{ if } j = k \\ 0, \text{ if } j \neq k \end{cases} .$$

For convenience, the basis $\{\mu_j\}$ is sometimes called the original basis, and the basis $\{\lambda_k\}$ is called its dual basis, even though the concept of duality is symmetric.

Lemma: If α is a root of an irreducible polynomial (i.e., one that cannot be factored) of degree m in $GF(p^m)$, then $\{1, \alpha, \alpha^2, \ldots, ^k\}$, is a basis of $GF(p^m)$. The set $\{\alpha^k\}$ $(0 \leq k \leq m-1)$ is called a natural basis of $GF(p^m)$.

Theorem 1: Every basis has a unique dual basis.

The following corollaries are central to the workings of the Berlekamp algorithm. Their proofs follow immediately from the above lemmas and theorems.

Corollary 1: Let $\{\mu_j\}$ be a basis of $GF(p^m)$ and let $\{\lambda_k\}$ be its dual basis. Then a field element z can be expressed in the dual basis $\{\lambda_k\}$ by the expansion

$$z = \sum_{k=0}^{m-1} z_k\lambda_k$$

where $z_k = Tr(z\mu_k)$.

Corollary 2: Let $\{\mu_j\}$ be a basis of $GF(p^m)$ and let $\{\lambda_k\}$ be its dual basis. The product $w = zy$ of two field elements in $GF(p^m)$ can be expressed in the dual basis by the expansion

$$w = \sum_{k=0}^{m-1} Tr(zy\mu_k)\lambda_k \quad .$$

These two corollaries provide a theoretical basis for the new RS-encoder algorithm. In the following section, a detailed example is developed to illustrate how Berlekamp's new bit-serial multiplication algorithm can be used to realize an RS-encoder structure as presented in Fig. 1.

IV. AN EXAMPLE OF THE BERLEKAMP MULTIPLICATION ALGORITHM

This section includes a summary of the exhibition of Berlekamp's algorithm given in reference [9]. An even simpler example is worked here.

Consider a (15, 11) RS code over $GF(2^4)$. For this code; $m=4$, $n=15$, $t=2$, $d=2t+1=5$, and the number of information symbols is $n-2t=11$. Let α be a root of the primitive irreducible polynomial $f(x) = x^4 + x + 1$ over GF(2). Then α satisfies the equation $\alpha^{15} = 1$. An element z in $GF(2^4)$ is representable by 0 or α^j for some j, $0 \leq j \leq 14$. z can also be represented by a polynomial in α over GF(2). This is the representation of $GF(2^4)$ in the conventional basis $\{1, \alpha, \alpha^2, \alpha^3\}$. That is, $z = u_0 + u_1\alpha + u_2\alpha^2 + u_3\alpha^3$, where $u_k \epsilon$ GF(2) (i.e. $u_k = 0$ or 1) for $0 \leq k \leq 3$.

In Table 1, the first column contains the logarithm in base α of an element in GF(24). The logarithm of the zero element is undefinable and is denoted by an asterisk. Column 2 shows the 4-tuples of the coefficients of the elements expressed as polynomials in α.

The trace of an element z in GF(2^4) is found by Def. 1 and the linear property of the trace to be

$$Tr(z) = u_0Tr(1) + u_1Tr(\alpha) + u_2Tr(\alpha^2) + u_3Tr(\alpha^3).$$

Now $Tr(1) \equiv 4 \pmod 2 = 0$, $Tr(\alpha) = Tr(\alpha^2) = \alpha + \alpha^2 + \alpha^4 + \alpha^8 = 0$ and $Tr(\alpha^3) = \alpha^3 + \alpha^6 + \alpha^9 + \alpha^{12} = 1$. This means that $Tr(z) = u_3$. The trace of the element α^j in GF(2^4) is displayed in column 3 of Table 1.

Table 1. Representations of the elements of GF(2^4) as generated by $\alpha^4 = \alpha + 1$.

	α^j		α^j
j	in conventional basis	TR(α^j)	in the dual basis
	$\alpha^3\alpha^2\alpha^1\alpha^0$		$z_0z_1z_2z_3$
*	0000	0	0000
0	0001	0	$0001 = \lambda_3$
1	0010	0	$0010 = \lambda_2$
2	0100	0	$0100 = \lambda_1$
3	1000	1	1001
4	0011	0	0011

Table 1. Representations of the elements of $GF(2^4)$ as generated by $\alpha^4 = \alpha + 1$. (Continued)

j	α^j in conventional basis	$TR(\alpha^j)$	α^j in the dual basis
	$\alpha^3\alpha^2\alpha^1\alpha^0$		$z_0z_1z_2z_3$
5	0110	0	0110
6	1100	1	1101
7	1011	1	1010
8	0101	0	0101
9	1010	1	1011
10	0111	0	0111
11	1110	1	1111
12	1111	1	1110
13	1101	1	1100
14	1001	1	$\underline{1000 = \lambda_0}$

By Def. 2 any set of four linearly independent elements can be used as a basis for the field $GF(2^4)$. To find the dual basis of the basis $\{1, \alpha, \alpha^2, \alpha^3\}$ in $GF(2^4)$, let a field element z be expressed in the dual basis $\{\lambda_0, \lambda_1, \lambda_2, \lambda_3\}$. From Corollary 1 the coefficients of z are $z_k = Tr(z\alpha^k)$ $(0 \leq k \leq 3)$. Thus $z_0 = Tr(z)$, $z_1 = Tr(z\alpha)$, $z_2 = Tr(z\alpha^2)$ and $z_3 = Tr(z\alpha^3)$. Let $z = \alpha^i$ for some $0 \leq i \leq 14$. A coefficient z_k, relative to the dual basis $(0 \leq k \leq 3)$, of an element z can be obtained by cyclically shifting all but row one of the trace column in Table 1 upward by k positions. These appropriately shifted columns of coefficients are shown in Table 1 in the last column. In Table 1 the elements of the dual basis, λ_0, λ_1, λ_2, λ_3, are underlined.

In order to make the generator polynomial g(x) symmetric, b must satisfy the equation $2b + d - 2 = 2^m - 1$, so $b = 6$ for this code. The γ in eq. (2) can be any primitive element in $GF(2^4)$. It will be shown in section IV that γ can be chosen so as to optimize the encoding logic. In this example, let $\gamma = \alpha$, so that the code generator polynomial is given by

$$g(x) = \prod_{j=6}^{9} (x - \alpha^j) = \sum_{i=0}^{4} g_i x^i . \tag{7}$$

One may verify that $g_0 = g_4 = 1$, $g_1 = g_3 = \alpha^3$ and $g_2 = \alpha$.

Let g_i be expressed in the original basis $\{1, \alpha, \alpha^2, \alpha^3\}$. Let z, a field element, be expressed in the dual basis, i.e., $z = z_0\lambda_0 + z_1\lambda_1 + z_2\lambda_2 + z_3\lambda_3$. The products zg_i $(0 \leq i \leq 3)$ need to be computed in order to implement the encoder shown in Fig. 1.

Since $g_3 = g_1$, it is only necessary to compute zg_0, zg_1, zg_2. Let the products zg_i, $0 \leq i \leq 2$, be represented in the dual basis. By Corollary 2, zg_i can be expressed in the dual basis as

$$zg_i = \sum_{k=0}^{3} T_i^{(k)}(z)\,\lambda_k \tag{8}$$

where $T_i^{(k)}(z) = \mathrm{Tr}(zg_i\alpha^k)$ is the k-th coefficient (or k-th bit) of zg_i (for $0 \leq i \leq 2$ and $0 \leq k \leq 3$).

The intent is to express $T_i^{(k)}$ recursively in terms of $T_i^{(k-1)}$ for $1 \leq k \leq 3$. For k = 0,

$$\begin{bmatrix} T_0^{(0)}\ (z) \\ T_1^{(0)}\ (z) \\ T_2^{(0)}\ (z) \end{bmatrix} = \begin{bmatrix} \mathrm{Tr}(zg_0) \\ \mathrm{Tr}(zg_1) \\ \mathrm{Tr}(zg_2) \end{bmatrix} = \begin{bmatrix} \mathrm{Tr}(z\ \ ^{0}) \\ \mathrm{Tr}(z\ \ ^{3}) \\ \mathrm{Tr}(z\ \ ) \end{bmatrix} = \begin{bmatrix} z_0 \\ z_3 \\ z_1 \end{bmatrix} \tag{9}$$

where $\mathrm{Tr}(z\alpha^j) = \mathrm{Tr}((z_0\lambda_0 + z_1\lambda_1 + z_2\lambda_2 + z_3\lambda_3)\alpha^j) = z_j$ for $0 \leq j \leq 3$. Eq. (9) can be expressed in matrix form as follows:

$$\begin{bmatrix} T_0^{(0)}\ (z) \\ T_1^{(0)}\ (z) \\ T_2^{(0)}\ (z) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} \equiv M \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{10}$$

The matrix M is called the binary mapping matrix. Observe that $T_i^{(k)}(z) = Tr((\alpha z)g_i\alpha^{k-1}) = T_i^{(k-1)}(\alpha z)$. Hence $T_i^{(k)}$ is obtained from $T_i^{(k-1)}$ by replacing z by $y = \alpha z$. Let $\alpha z = y = y_0\lambda_0 + y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3$, where $y_m = Tr(y\alpha^m) = Tr(z\alpha^{m+1})$ for each m. Then $T_i^{(k)}$ is obtained from $T_i^{(k-1)}$ by replacing z_0 by $y_0 = Tr(z\alpha) = z_1$, z_1 by $y_1 = Tr(z\alpha^2) = z_2$, z_2 by $y_2 = Tr(z\alpha^3) = z^3$, and z_3 by $y_3 = Tr(z\alpha^4) = Tr(z(\alpha+1)) = z_0 + z_1$.

Berlekamp's bit-serial multiplication algorithm may now be stated for GF(24). The quantities $zg_i = T_i^{(0)}\lambda_0 + T_i^{(1)}\lambda_1 + T_i^{(2)}\lambda_2 + T_i^{(3)}\lambda_3$, $(0 \leq i \leq 3)$ and $z = z_0\lambda_0 + z_1\lambda_1 + z_2\lambda_2 + z_3\lambda_3$ can be computed as follows:

(1) Compute $T_0^{(0)}(z)$, $T_1^{(0)}(z)$ and $T_2^{(0)}(z)$ by Eq. (10). Also $T_3^{(0)}(z) = T_1^{(0)}(z)$.

(2) For k = 1, 2, 3, compute $T_i^{(k)}(z)$ $(0 \leq i \leq 3)$ by

$$T_i^{(k)}(z) = T_i^{(k-1)}(y)$$

where $y = \alpha z = y_0\lambda_0 + y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3$ with $y_0 = z_1$, $y_1 = z_2$, $y_2 = z_3$ and $y_3 = z_0 + z_1 = T_f$, and $T_f = z_0 + z_1$ which is the feedback term of the algorithm.

The above example illustrates Berlekamp's bit-serial multiplication algorithm. This algorithm requires shifting and XOR operations only. Berlekamp's dual basis RS-encoder is well-suited to a pipeline structure which can be implemented in VLSI design. The same procedure extends similarly to the design of a (255, 223) RS-encoder over $GF(2^8)$.

V. A VLSI ARCHITECTURE FOR A (255, 223) SINGLE CHIP RS-ENCODER

In this section, an architecture is developed for implementing a (255, 223) RS-encoder using Berlekamp's multiplication algorithm. The circuit makes use of Berlekamp's bit-serial multiplication algorithm as developed in the previous sections and the Mead-Conway VLSI design approach [10]. This architecture can be realized quite readily on a single NMOS VLSI chip.

Let $GF(2^8)$ be generated by α, where α is a root of the primitive irreducible polynomial $f(x) = x^8 + x^7 + x^2 + x + 1$ over $GF(2)$. The conventional basis of this field is therefore $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7\}$. The representations of elements in this field in both the conventional basis and its dual basis are tabulated in Appendix A. Corollary 1 states that the coefficients of a field element α^j can be obtained by $z_k = Tr(\alpha^{j+k})$ $(0 \leq k \leq 7)$, where $\alpha^j = z_0\lambda_0 + \ldots + z_7\lambda_7$. From Table 2 in Appendix A, the dual basis $\{\lambda_0, \lambda_1, \ldots, \lambda_7\}$ of the conventional basis is the set $\{\alpha^{99}, \alpha^{197}, \alpha^{203}, \alpha^{202}, \alpha^{201}, \alpha^{200}, \alpha^{199}, \alpha^{100}\}$.

γ, in eq. (2) can be chosen to minimize the number of ones in the binary mapping matrix. Two binary matrices, one for the primitive element $\gamma = \alpha^{11}$ and the other for $\gamma = \alpha$, were computed. It was found that the binary mapping matrix for $\gamma = \alpha^{11}$ had a smaller number of 1's. Hence this binary mapping matrix was used in the design. For this case the generator polynomial g(x) of the RS-encoder over $GF(2^8)$ was given by

$$g(x) = \prod_{j=112}^{143} (x - \alpha^{11j}) = \sum_{i=0}^{32} g_i x^i . \qquad (11)$$

Table 2 lists the coefficients g_i of $g(x)$.

The binary mapping matrix for the coefficients of the generator polynomial in eq. (11) is computed and shown in Appendix B. The feedback term T_f in Berlekamp's algorithm is:

$$T_f = Tr(\alpha^8 z) = Tr \ ((\alpha^7 + \alpha^2 + \alpha + 1))z \ = z_0 + z_1 + z_2 + z_7. \qquad (12)$$

Table 2. The code generator polynomial for the (255, 223) RS code

$$g(x) = \prod_{j=112}^{143} (x-\alpha^{11j}) = \sum_{i=0}^{32} g_i x^i$$

i	g(i)	i	g(i)
0	1		α^5
1	α^{249}	17	α^{170}
2	α^{59}	18	α^{66}
3	α^{66}	19	α^{50}
4	α^4	20	α^{213}

Table 2. The code generator polynomial for the (255, 223) RS code (Continued)

$$g(x) = \prod_{j=112}^{143} (x-\alpha^{11j}) = \sum_{i=0}^{32} g_i x^i$$

i	g(i)	i	g(i)
5	α^{43}	21	α^{3}
6	α^{126}	22	α^{30}
7	α^{251}	23	α^{97}
8	α^{97}	24	α^{251}
9	α^{30}	25	α^{126}
10	α^{3}	26	α^{43}
11	α^{213}	27	α^{4}
12	α^{50}	28	α^{66}
13	α^{66}	29	α^{59}
14	α^{170}	30	α^{249}
15	α^{5}	31	1
16	α^{24}	32	

A diagram showing the input and output signals chip is shown in Fig. 2.; VDD and GND are power pins. CLK is a clock signal, which in general is a periodic square wave supplied by an external signal generator. The information symbols are fed into the chip from the data-in pin, DIN, serially. This means that it takes eight clock times to read in each symbol. Similarly, the encoded RS codeword is transmitted out of the chip serially from the data-out pin, DOUT. The control signal LM (Load Mode) is set to 1 (logic 1) when the information symbols are loaded into the chip.

The DIN and LM signals are expected to be synchronized to the CLK signal, while the internal operations of the circuit and output data signal are synchronized to two non-overlapping clock signals $\phi 1$ and $\phi 2$ that are derived from CLK inside the chip. To save space, dynamic shift registers are used in this design for memory. A logic diagram of a 1-bit dynamic register with reset is shown in Fig. 3. The timing diagram of CLK, $\phi 1$, $\phi 2$, LM, DIN and DOUT signals are shown in Fig. 4. The delay of DOUT with respect to DIN is due to input and output buffering flip-flops.

Figure 5 shows the block diagram of the (255, 223) RS-encoder. The circuit is divided into five units as follows:

(1) Product Unit: The Product Unit is used to compute T_f, T_{31}, ..., T_0. This circuit is realized by a Programable Logic Array (PLA) [9]. Since $T_0 = T_{31}$, $T_1 = T_{30}$, ..., $T_{15} = T_{17}$, only T_f, T_{31}, ..., T_{17} and T_{16} are actually calculated in the PLA T_0, ..., T_{15} are connected directly to T_{31}, ..., T_{17},

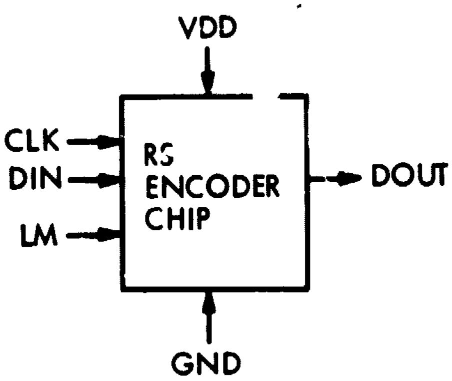


Figure 2. Inputs and output for the RS encoder chip

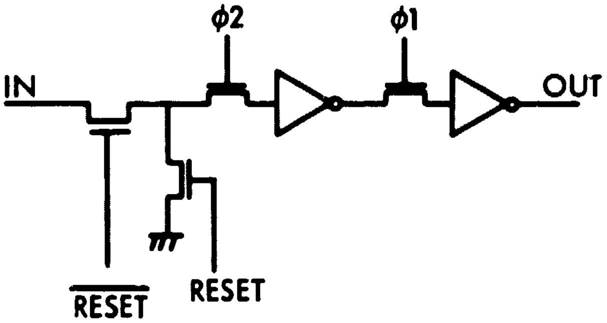


Figure 3. Logic diagram of a 1-bit dynamic shift register with reset

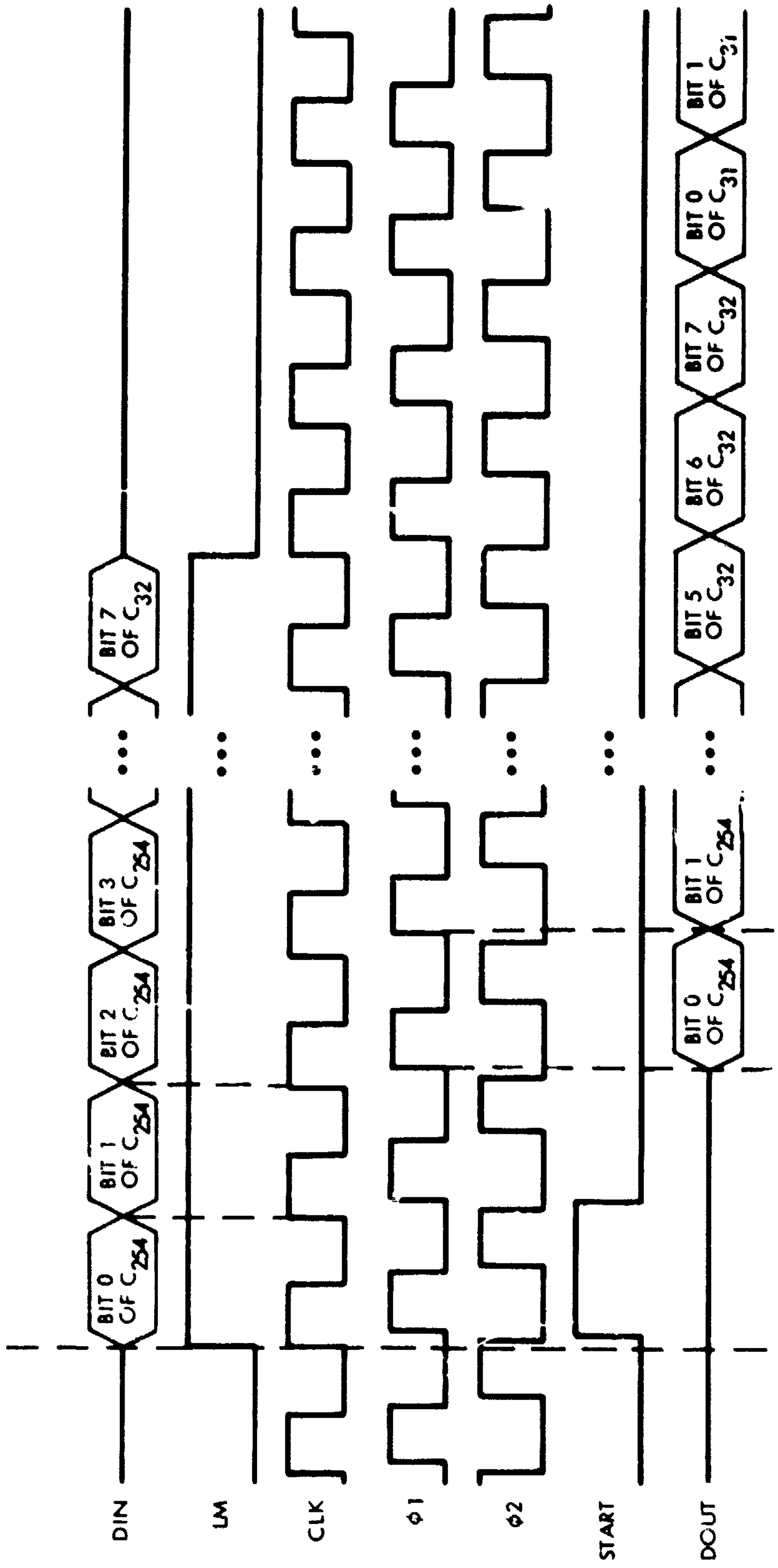


Figure 4. Timing diagrams of DIN, LM, CLK, 1, 2, START, and DOUT

ORIGINAL PAGE IS
OF POOR QUALITY

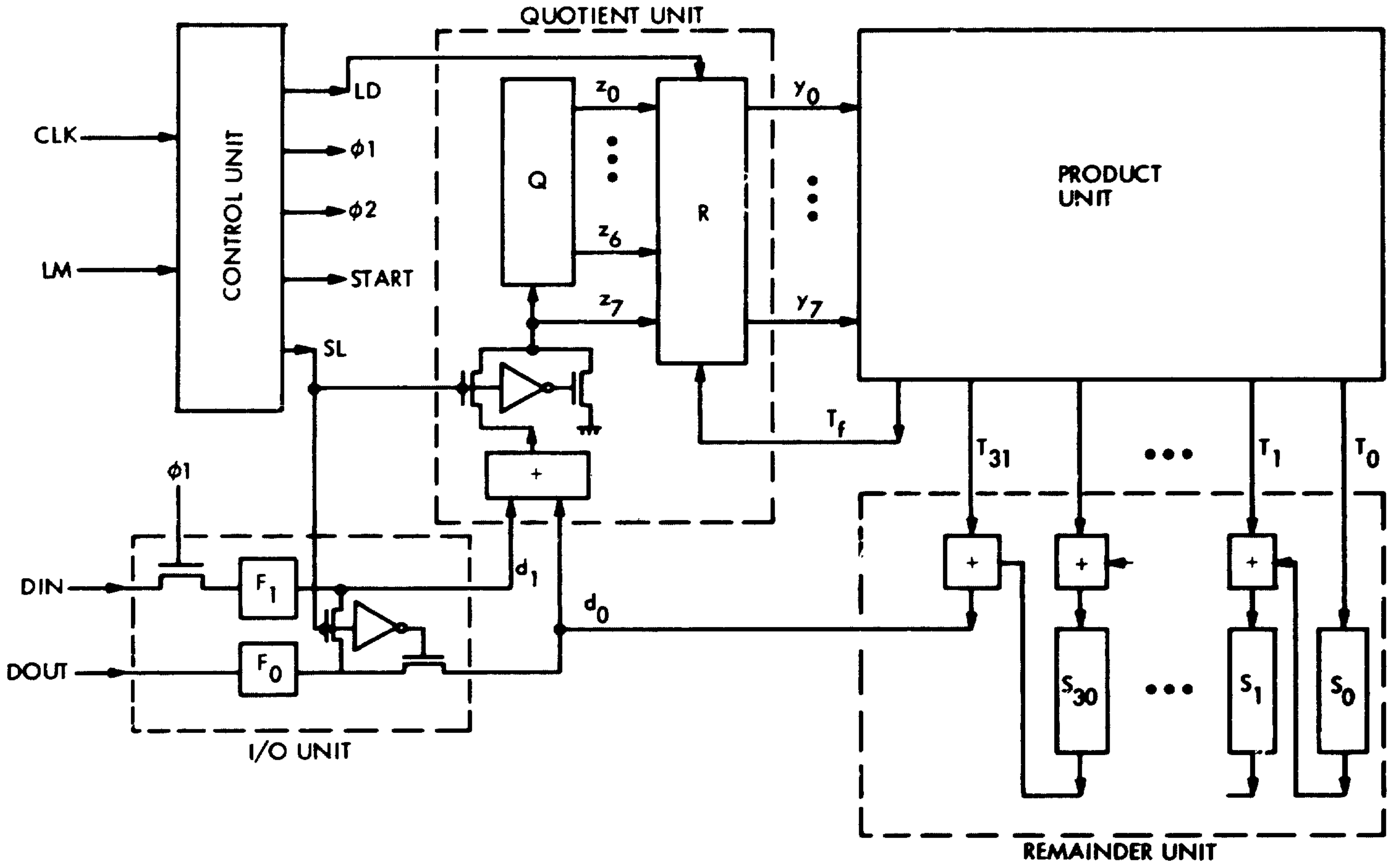


Figure 5. Block diagram of the (255, 223) RS encoder

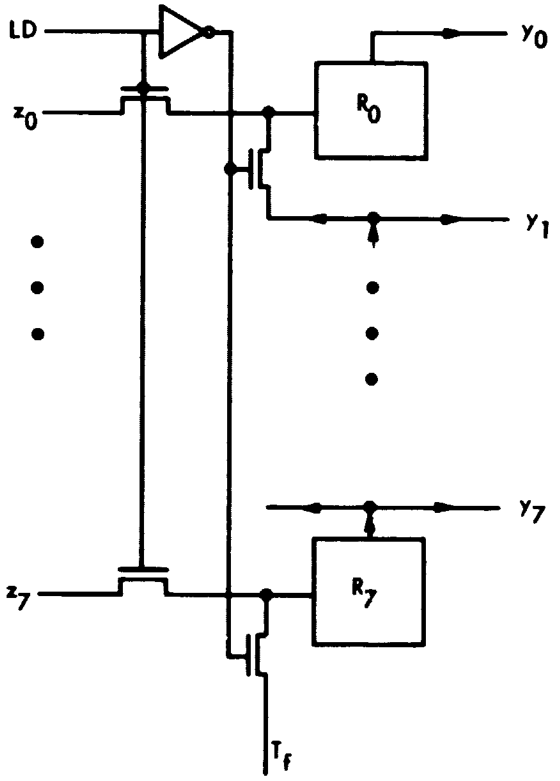
respectively. The implementation of the product unit PLA means that it would be easy to reconfigure the encoder to use a different representation of GF(2^8).

(2) Remainder Unit: The Remainder Unit is used to store the coefficients of the remainder during the division process. Each S_i $(0 \leq i \leq 30)$ is an 8-bit dynamic shift register with reset. The addition in the circuit is a modulo 2 addition (Exclusive-OR operation). While c_{32} is being fed to the circuit, c_{31} is being computed and then loaded into S_i $(0 \leq i \leq 30)$. The $c_{30}, \ldots, c_0$ are transmitted out of the encoder serially.

(3) Quotient Unit: Q and R represent a 7-bit shift register with reset and an 8-bit shift register with reset and parallel load, respectively. R and Q store the currently operating coefficient and the next coefficient of the quotient polynomial, respectivley. A logic diagram of register R is shown in Fig. 6. Each z_i is loaded into R_i every eight clock cycles. Immediately after all 223 information symbols have been fed into the circuit, the control signal SL changes to a logical 0. Henceforth the contents of Q and R are set to zero so that the check symbols in the Remainder Unit return their current values.

(4) I/O Unit: This unit handles the input/output operations. Both F_0 and F_1 are flip-flops. A pass transistor controlled by $\phi 1$ is inserted before F_1 for the purpose of synchronization. The control signal SL selects whether a bit of an information symbol or a check symbol is to be transmitted.

ORIGINAL PAGE IS
OF POOR QUALITY



R_i: A 1-BIT REGISTER WITH RESET

Figure 6. The R register

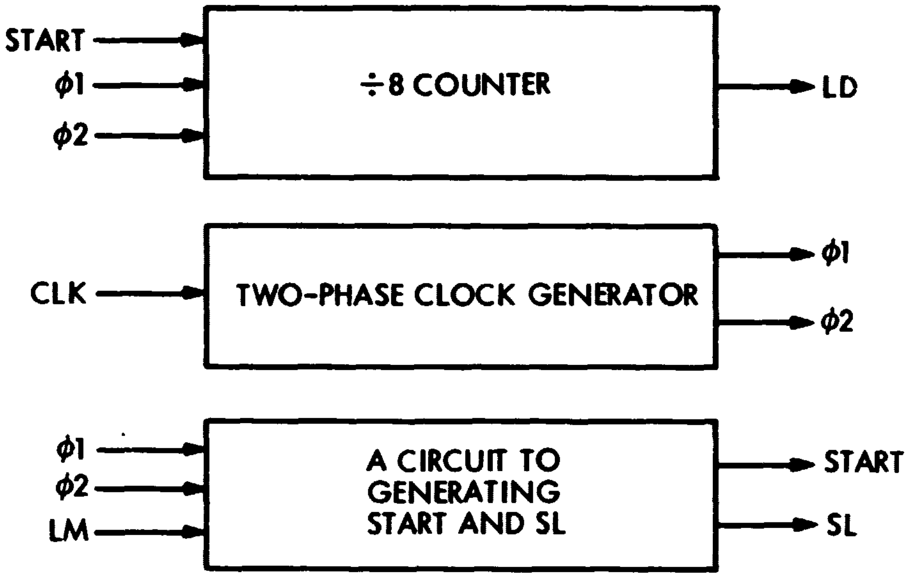

START
ϕ1
ϕ2
÷8 COUNTER
LD
CLK
TWO-PHASE CLOCK GENERATOR
ϕ1
ϕ2
ϕ1
ϕ2
LM
A CIRCUIT TO GENERATING START AND SL
START
SL

Figure 7. The control unit

(5) Control Unit: The Control Unit generates the necessary control signals. This unit is further divided into 3 portions, as shown in Fig. 7. The two-phase clock generator circuit in [10] is used to convert the externally supplied CLK clock signal into the two phase clock needed for the operation of this chip. Fig. 8 shows a logic diagram of the circuit for generating the control signals START and SL. The control signal START resets all registers and the divide-by-8 counter before the encoding process begins. The control signal SL is simply a delayed version of LM. The control signal LD is simply generated by a divide-by-8 counter and is used to load the z_i's into the R_i's in parallel.

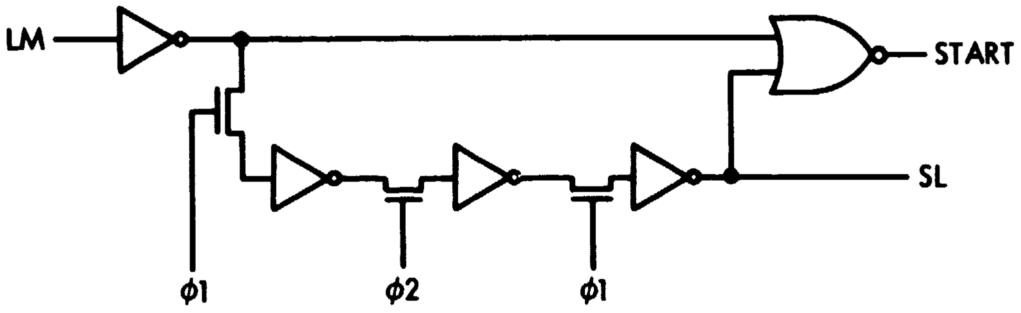


Figure 8. A logic diagram of the circuit for generating the control signals START and SL.

Since a codeword contains 255 symbols, the computation of a complete encoded codeword requires 255 "symbol cycles". A symbol cycle is the time interval required for executing a complete cycle of Berlekamp's algorithm. Since a symbol consists of 8 bits, a symbol cycle contains 8 "bit cycles". A bit cycle is the time interval for executing one step in Berlekamp's algorithm. In this design a bit cycle corresponds to one period of the clock.

The total number of clock cycles required to encode a single RS word is therefore 255 x 8 = 2040. Although it is not known at this time what the maximum clock speed for this chip will be, a conservative estimate would be 1 MHz. This would mean that data may be input to the chip at an average rate (223/255) x 10^6 = 874 kbps. The delay through the chip would be about 2 ms.

The layout design of this (255, 223) RS-encoder has been completed (See Appendix C). Before the design of the lay›ut each circuit was simulated on a general-purpose computer by using SPICE (a transistor-level circuit simulation program) [11]. The circuit requires about 3000 transistors, while a similar design without VLSI requires more than 30 CMOS IC chips [6]. This RS-encoder design will be fabricated and tested in the near future. Table 3 shows how the encoder would be changed to implement other RS codes.

Table 3. Modifications to the encoder circuit needed to change a code parameter

Parameter to be changed	The value used for the circuit in Fig. 2	New value	Modifications
1. Generator polynomial	Eq. (8)	g(x)	The PLA of the Product Unit is reprogrammed.
2. The finite field used	GF(2^8)	GF(2^m)	All 8-bit registers become m-bit registers, and Q becomes an (m-1) bit register. A divide-by-m counter is used.

Table 3. Modifications to the encoder circuit needed to change a code parameter (Continued)

Parameter to be changed	The value used for the circuit in Fig. 2	New value	Modifications
3. Error-Correcting Capability	16	t	2t-2 shift registers are required in the Remainder Unit. (The generator polynomial is also changed.)
4. Number of Information Symbols	223	k	No change is required, since k is implicitly contained in the control signal LM. This may be used to generate shortened RS codes.

VI. CONCLUSIONS

This project has proven the feasibility of 8-bit Reed-Solomon encoder implementation on a single chip. The obvious application of such a device is as part of an on-board data encoding system for satellites and deep spacecraft. The savings in weight, size, and power over present encoders is evident. Also, the reduced number of circuit interconnects and the absence of discrete components should make the VLSI unit more reliable.

Two problems arise, however, when one considers using this encoder as flight hardware. The first is that this encoder is designed for implementation in NMOS and so will probably not meet radiation-hardness requirements. The decision to design for NMOS was made on the basis of low cost and low risk for this experimental first chip. There would be no problem in redesigning the encoder for implementation in other VLSI technologies if the need arises.

The second problem is that the encoder, as presently designed, does not do symbol interleaving. Interleaving is particularly useful when data are to be transmitted over bursty channels (such as the Viterbi channel [3]). There are three solutions to this problem.

First, an external interleaver may be added. The use of VLSI random access memories would keep the total chip count down to the point where the interleaved RS encoder would still be smaller than the present encoders.

Second, several RS encoder chips could be used in parallel to achieve interleaving. Since standard RS interleaving depths are less than or equal to five [2], at most five encoder chips would be needed, plus a very simple time sharing logic.

Third, the encoder chip may be redesigned so as to perform the interleaving operation internally. Presently, the chip design consists of about 3,000 transistors - a modest amount for current VLSI technology. If interleaving to a depth of five were added, then the count would go up to around 13,000 transistors. This is because the remainder unit, which contains the storage registers, is about 80% of the chip by area. Even this number of transisters is considered only medium density for VLSI. It would also be a simple manner to make the interleaving depth programmable between one and five by adding three I/O connections to the chip to allow for the appropriate signals.

References

[1] R. R. Stephens and M. F. Pellet, "Joint NASA/ESA Telemetry Channel Coding Guideline: Issue 1", NASA/ESA Working Group (NEWG) Publication, January 1982.

[2] R. L. Miller, L. J. Deutsch, and S. A. Butman, On the Error Statistics of Viterbi Decoding and the Performance of Concatenated Codes, Jet Propulsion Laboratory, Pasadena, Calif., September 1, 1981.

[3] K. Y. Liu and J. J. Lee, An Experimental Study of the Concatenated Reed-Solomon/Viterbi Channel Coding System and Its Impact on Space Communications. Publication 81-58, Jet Propulsion Laboratory, Pasadena, Calif., August 15, 1981.

[4] R. F. Rice, Some Practical Universal Noiseless Coding Techniques, Publication 79-22, Jet Propulsion Laboratory, Pasadena, Calif., March 15, 1979.

References (Continud)

[5] W. W. Peterson and E. J. Weldon, Error-Correcting Codes, MIT Press, 1972.

[6] E. R. Berlekamp, "Technical Proposal for a Low-Power Reed-Solomon Encoder/Interleaver Using About 30 CMOS IC's," Cyclotomics, Inc., in response to RFP No. BP-6-9007.

[7] I. N. Herstein, Topics in Algebra, Blaisdell, 1964.

[8] P. J. MacWilliams and N. J. A. Sloane, The Theory of Error-Correcting Codes, North-Holland Publishing Company, 1978.

[9] M. Perlman and J. J. Lee, "A Comparison of Conventional Reed-Solomon Encoders and Berlekamp's Architecture," NASA Tech Brief, No. 3610-81-119, Jet Propulsion Laboratory, Calif., July 10, 1981.

[10] C. Mead and L. Conway, Introducing To VLSI Systems, Addison-Wesley Publishing Company, Calif., 1980.

[11] L. W. Negal and D. O. Pederson, "SPICE - Simulation Program with Integrated Circuit Emphasis," Memorandum No. ERL-M382, Electronics Research Laboratory, University of California, Berkeley, April 12, 1973.

Appendix A. Table of Elements of GF(2^8)

This appendix lists all 256 elements in GF(2^8). These field elements are expressed in both the conventional basis $\{1, \alpha, \alpha^2, \ldots, \alpha^7\}$ and its dual basis, $\{\lambda_0, \lambda_1, \ldots, \lambda_7\}$. The elements of the dual basis are indicated by underlines.

Table A1. Representative of the elements of GF(2^8)

	α^j		α^j		α^j		α^j
	CONVENTIONAL	TR(α^j)	DUAL		IN CONVENTIONAL	TR(α^j)	IN DUAL
j	BASIS		BASIS	j	BASIS		BASIS
*	00000000	0	00000000	18	00111011	0	01110010
0	00000001	0	01111111	19	01110110	1	11100100
1	00000010	1	11111111	20	11101100	1	11001001
2	00000100	1	11111110	21	01011111	1	10010011
3	00001000	1	11111101	22	10111110	0	00100110
4	00010000	1	11111010	23	11111011	0	01001101
5	00100000	1	11110101	24	01110001	1	10011010
6	01000000	1	11101010	25	11100010	0	00110101
7	10000000	1	11010101	26	01000011	0	01101010
8	10000111	1	10101011	27	10000110	1	11010100
9	10001001	0	01010111	28	10001011	1	10101000
10	10010101	1	10101110	29	10010001	0	01010000
11	10101101	0	01011100	30	10100101	1	10100001

Table A1. Representative of the elements of GF(2^8) (Continued)

j	α^j CONVENTIONAL BASIS	TR(α^j)	α^j DUAL BASIS	j	α^j IN CONVENTIONAL BASIS	TR(α^j)	α^j IN DUAL BASIS
12	11011101	1	10111001	31	11001101	0	01000011
13	00111101	0	01110011	32	00011101	1	10000110
14	01111010	1	11100111	33	00111010	0	00001101
15	11110100	1	11001110	34	01110100	0	00011011
16	01101111	1	10011100	35	11101000	0	00110111
17	11011110	0	00111001	36	01010111	0	01101110
37	10101110	1	11011100	60	11111110	1	11001100
38	11011011	1	10111000	61	01111011	1	10011000
39	00110001	0	01110000	62	11110110	0	00110001
40	01100010	1	11100000	63	01101011	0	01100010
41	11000100	1	11000001	64	11010110	1	11000100
42	00001111	1	10000011	65	00101011	1	10001000
43	00011110	0	00000110	66	01010110	0	00010001
44	00111100	0	00001100	67	10101100	0	00100011
45	01111000	0	00011000	68	11011111	0	01000110
46	11110000	0	00110000	69	00111001	1	10001101
47	01100111	0	01100001	70	01110010	0	00011010
48	11001110	1	11000011	71	11100100	0	00110100
49	00011011	1	10000111	72	01001111	0	01101001
50	00110110	0	00001110	73	10011110	1	11010011

Table A1. Representative of the elements of GF(2^8) (Continued)

j	α^j CONVENTIONAL BASIS	TR(α^j)	α^j DUAL BASIS	j	α^j IN CONVENTIONAL BASIS	TR(α^j)	α^j IN DUAL BASIS
51	01101100	0	00011100	74	10111011	1	10100111
52	11011000	0	00111000	75	11110001	0	01001111
53	00110111	0	01110001	76	01100101	1	10011110
54	01101110	1	11100011	77	11001010	0	00111101
55	11011100	1	11000110	78	00010011	0	01111010
56	00111111	1	10001100	79	00100110	1	11110100
57	01111110	0	00011001	80	01001100	1	11101001
58	11111100	0	00110011	81	10011000	1	11010010
59	01111111	0	01100110	82	10110111	1	10100100
83	11101001	0	01001000	106	00000111	0	01111110
84	01010101	1	10010001	107	00001110	1	11111100
85	10101010	0	00100010	108	00011100	1	11111001
86	11010011	0	01000101	109	00111000	1	11110010
87	00100001	1	10001010	110	01110000	1	11100101
88	01000010	0	00010101	111	11100000	1	11001010
89	10000100	0	00101011	112	01000111	1	10010100
90	10001111	0	01010110	113	10001110	0	00101001
91	10011001	1	10101101	114	10011011	0	01010010
92	10110101	0	01011011	115	10110001	1	10100101
93	11101101	1	10110110	116	11100101	0	01001011

Table A1. Representative of the elements of GF(2^8) (Continued)

j	α^j CONVENTIONAL BASIS	TR(α^j)	α^j DUAL BASIS	j	α^j IN CONVENTIONAL BASIS	TR(α^j)	α^j IN DUAL BASIS
94	01011101	0	01101100	117	01001101	1	10010110
95	10111010	1	11011000	118	10011010	0	00101101
96	11110011	1	10110000	119	10110011	0	01011010
97	01100001	0	01100000	120	11100001	1	10110101
98	11000010	1	11000000	121	01000101	0	01101011
99	00000011	1	$\underline{10000000\lambda_0}$	122	10001010	1	11010111
100	00000110	0	$\underline{00000001\lambda_7}$	123	10010011	1	10101111
101	00001100	0	00000011	124	10100001	0	01011111
102	00011000	0	00000111	125	11000101	1	10111110
103	00110000	0	00001111	126	00001101	0	01111100
104	01100000	0	00011111	127	00011010	1	11111000
105	11000000	0	00111111	128	00110100	1	11110001
129	01101000	1	11100010	152	01011001	1	10010010
130	11010000	1	11000101	153	10110010	0	00100101
131	00100111	1	10001011	154	11100011	0	01001010
132	01001110	0	00010110	155	01000001	1	10010101
133	10011100	0	00101100	156	10000010	0	00101010
134	10111111	0	01011001	157	10000011	0	01010101
135	11111001	1	10110010	158	10000001	1	10101010
136	01110101	0	01100100	159	10000101	0	01010100

Table A1. Representative of the elements of GF(2^8) (Continued)

j	α^j CONVENTIONAL BASIS	TR(α^j)	α^j DUAL BASIS	j	α^j IN CONVENTIONAL BASIS	TR(α^j)	α^j IN DUAL BASIS
137	11101010	1	11001000	160	10001101	1	10101001
138	01010011	1	10010000	161	10011101	0	01010011
139	10100110	0	00100001	162	10111101	1	10100110
140	11001011	0	01000010	163	11111101	0	01001100
141	00010001	1	10000101	164	01111101	1	10011001
142	00100010	0	00001010	165	11111010	0	00110010
143	01000100	0	00010100	166	01110011	0	01100101
144	10001000	0	00101000	167	11100110	1	11001011
145	10010111	0	01010001	168	01001011	1	10010111
146	10101001	1	10100010	169	10010110	0	00101110
147	11010101	0	01000100	170	10101011	0	01011101
148	00101101	1	10001001	171	11010001	1	10111010
149	01011010	0	00010010	172	00100101	0	01110100
150	10110100	0	00100100	173	01001010	1	11101000
151	11101111	0	01001001	174	10010100	1	11010001
175	10101111	1	10100011	198	00000101	1	10000001
176	11011001	0	01000111	199	00001010	0	$\underline{00000010\lambda_6}$
177	00110101	1	10001110	200	00010100	0	$\underline{00000100\lambda_5}$
178	01101010	0	00011101	201	00101000	0	$\underline{00001000\lambda_4}$
179	11010100	0	00111011	202	01010000	0	$\underline{00010000\lambda_3}$

Table A1. Representative of the elements of GF(2^8) (Continued)

	α^j		α^j		α^j		α^j
	CONVENTIONAL	TR(α^j)	DUAL		IN CONVENTIONAL	TR(α^j)	IN DUAL
j	BASIS		BASIS	j	BASIS		BASIS
180	00101111	0	01110110	203	10100000	0	$\underline{00100000\lambda_2}$
181	01011110	1	11101100	204	11000111	0	01000001
182	10111100	1	11011001	205	00001001	1	10000010
183	11111111	1	10110011	206	00010010	0	00000101
184	01111001	0	01100111	207	00100100	0	00001011
185	11110010	1	11001111	208	01001000	0	00010111
186	01100011	1	10011111	209	10010000	0	00101111
187	11000110	0	00111110	210	10100111	0	01011110
188	00001011	0	01111101	211	11001001	1	10111101
189	00010110	1	11111011	212	00010101	0	01111011
190	00101100	1	11110110	213	00101010	1	11110111
191	01011000	1	11101101	214	01010100	1	11101110
192	10110000	1	11011010	215	10101000	1	11011101
193	11100111	1	10110100	216	11010111	1	10111011
194	01001001	0	01101000	217	00101001	0	01110111
195	10010010	1	11010000	218	01010010	1	11101111
196	10100011	1	10100000	219	10100100	1	11011110
197	11000001	0	$\underline{01000000\lambda_1}$	220	11001111	1	10111100
221	00011001	0	01111000	238	01101101	0	01100011
222	00110010	1	11110000	239	11011010	1	11000111

Table A1. Representative of the elements of GF(2^8) (Continued)

j	α^j CONVENTIONAL BASIS	TR(α^j)	α^j DUAL BASIS	j	α^j IN CONVENTIONAL BASIS	TR(α^j)	α^j IN DUAL BASIS
223	01100100	1	11100001	240	00110011	1	10001111
224	11001000	1	11000010	241	01100110	0	00011110
225	00010111	1	10000100	242	11001100	0	00111100
226	00101110	0	00001001	243	00011111	0	01111001
227	01011100	0	00010011	244	00111110		11110011
228	10111000	0	00100111	245	01111100	1	11100110
229	11110111	0	01001110	246	11111000	1	11001101
230	01101001	1	10011101	247	01110111	1	10011011
231	11010010	0	00111010	248	11101110	0	00110110
232	00100011	0	01110101	249	01011011	0	01101101
233	01000110	1	11101011	250	10110110	1	11011011
234	10001100	1	11010110	251	11101011	1	10110111
235	10011111	1	10101100	252	01010001	0	01101111
236	10111001	0	01011000	253	10100010	1	11011111
237	11110101	1	10110001	254	11000011	1	10111111

Appendix B: The Binary Mapping Matrix for the (255, 223) RS Encoder

The binary mapping matrix for $\gamma = \alpha^{11}$ of the (255, 223) RS-encoder is given by the following:

$$\begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \\ T_{10} \\ T_{11} \\ T_{12} \\ T_{13} \\ T_{14} \\ T_{15} \\ T_{16} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \\ Z_7 \end{bmatrix}$$

Appendix C: VLSI Layout for the (255, 223) RS Encoder

The figures in this appendix show the actual layout for the (255, 223) RS encoder chip. The total layout is shown in Figure C1. Figures C-2 through C-5 show the polysilicon, metal, diffusion, and contact layers for the chip.

ORIGINAL PAGE
COLOR PHOTOGRAPH

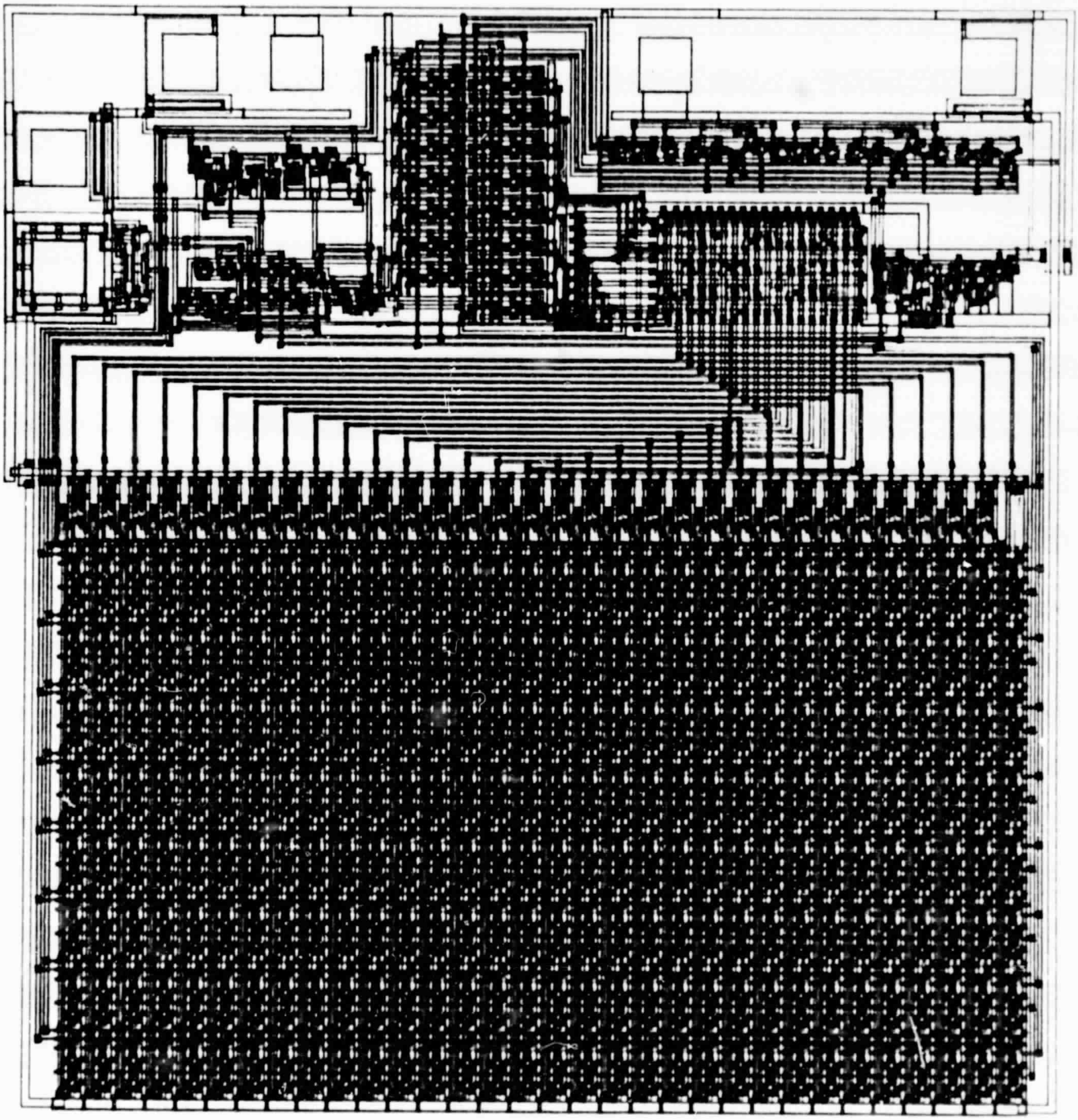


Figure Cl. Total layout of the (255, 223) RS-encoder chip

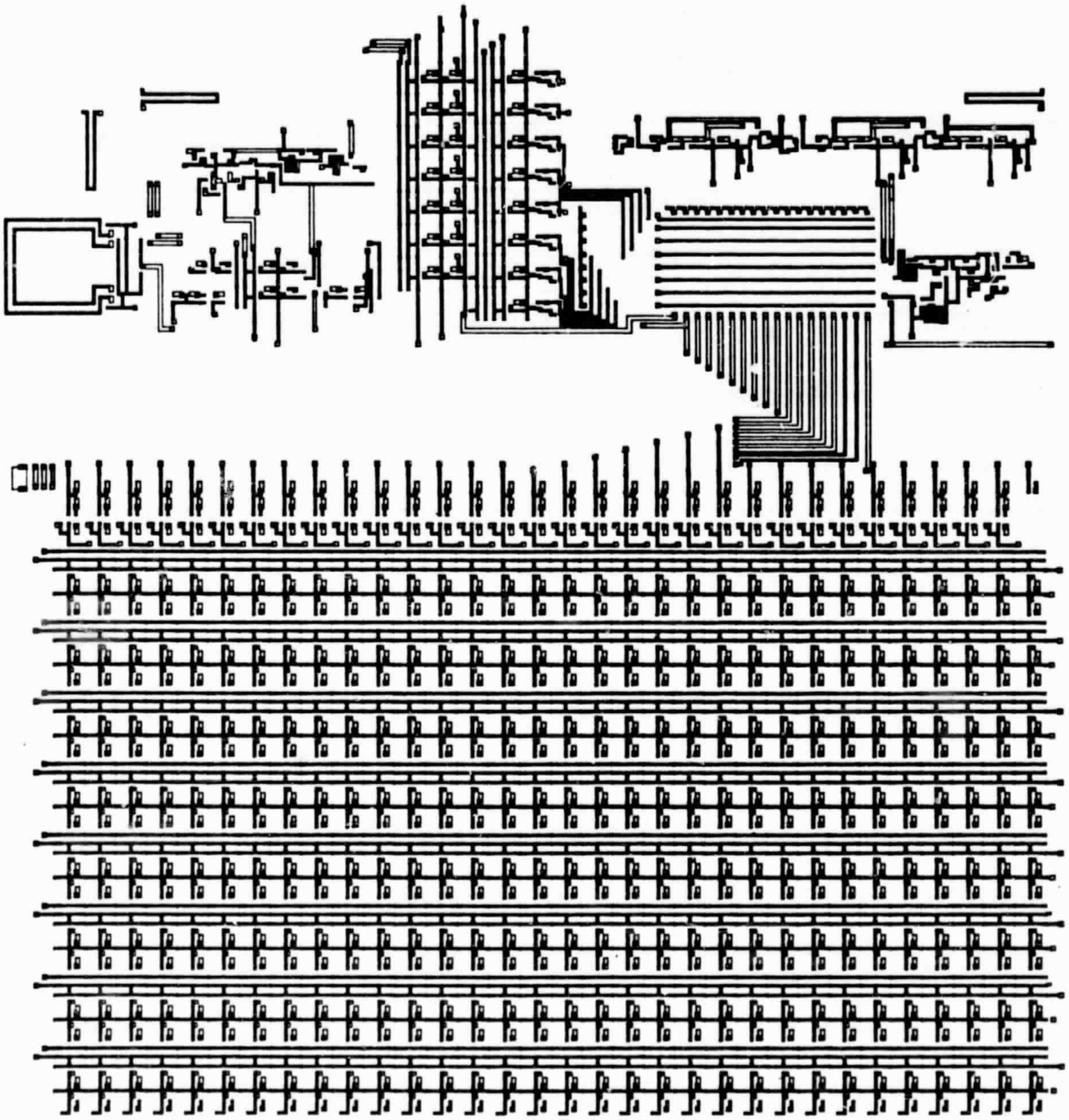


Figure C2. Polysilicon layout of the (255, 223) RS-encoder chip

ORIGINAL PAGE IS
OF POOR QUALITY

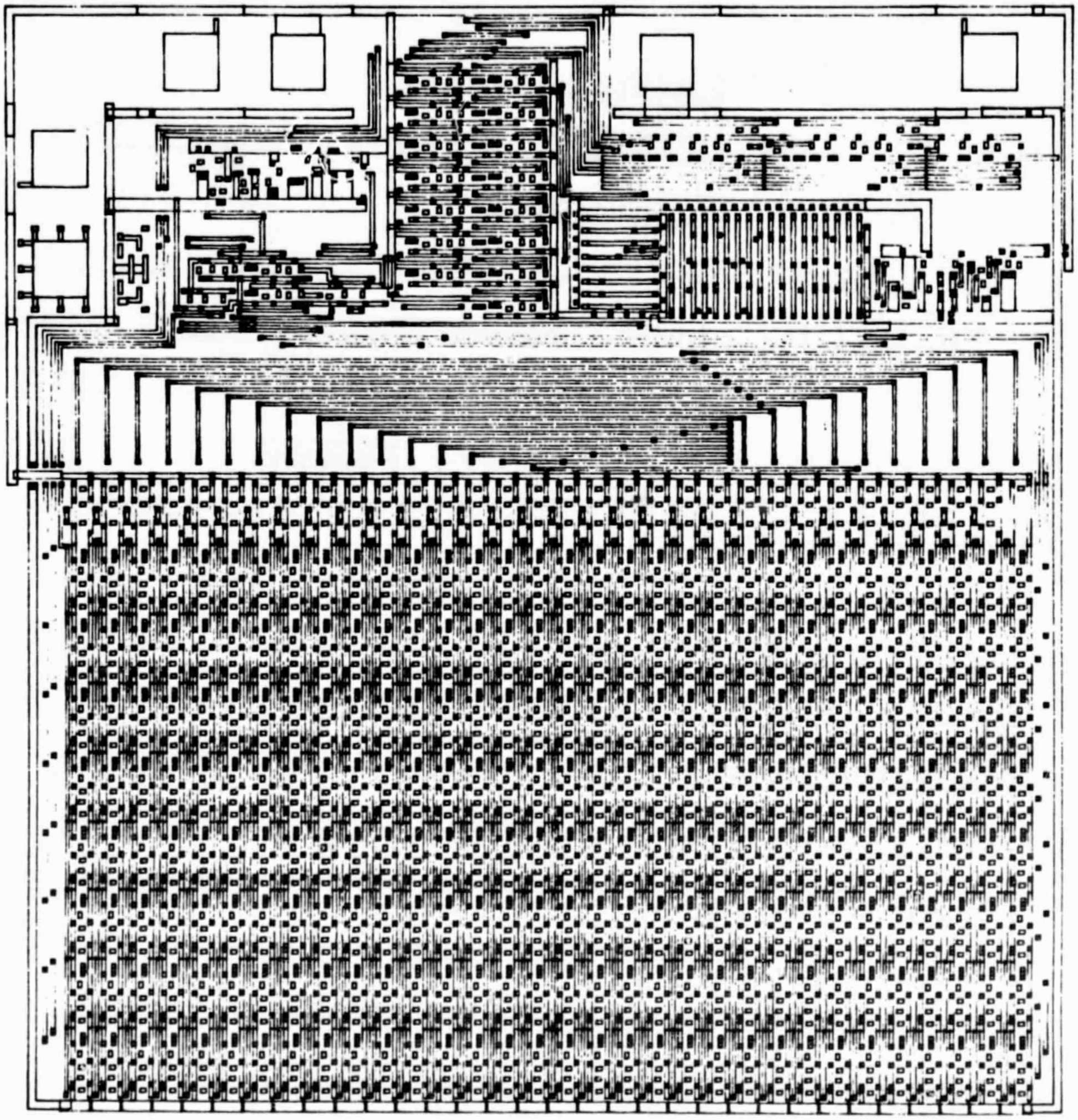


Figure C3. Metal layout of the (255, 223) RS-encoder chip

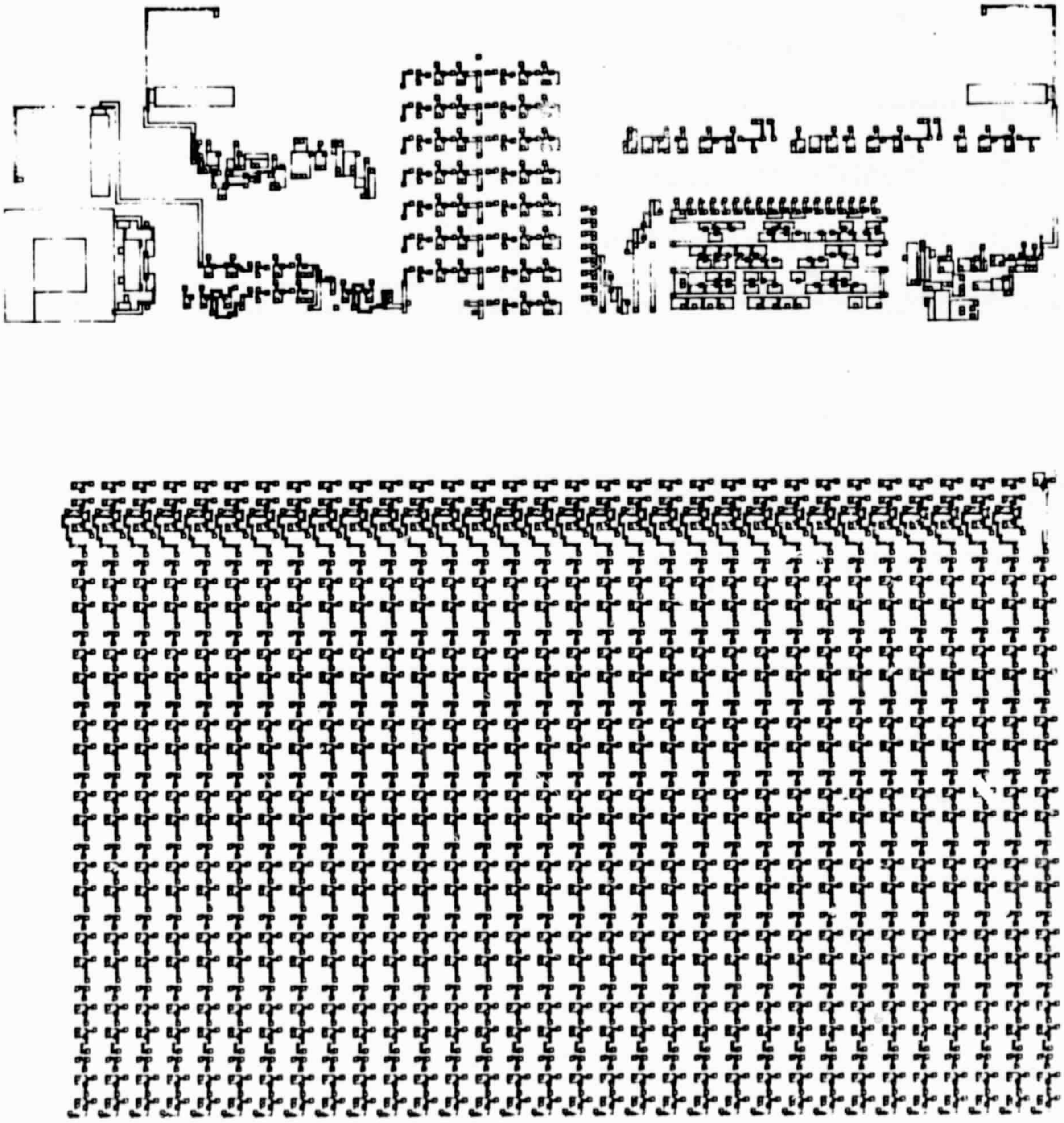


Figure C4. Diffusion layout of the (255,223) RS-encoder chip

ORIGINAL PAGE IS
OF POOR QUALITY

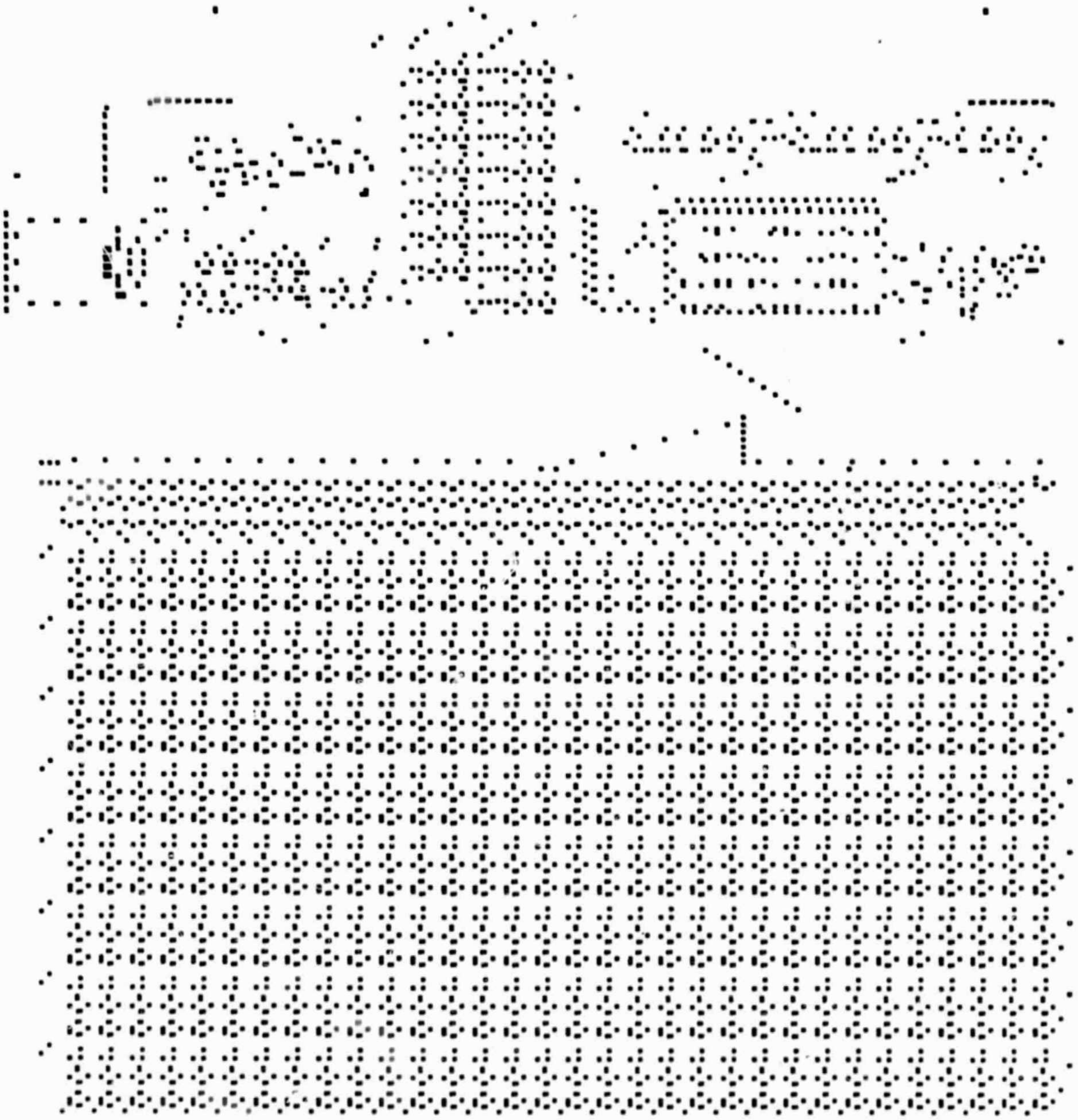


Figure C5. Contact layout of the (255, 223) RS-encoder chip

NASA—JPL—Coml., L.A., Calif.