J. D. Ullman*

Decoding of Cyclic Codes Using Position Invariant Functions

Abstract: Ratios that are sufficient to detect classes of error patterns in cyclic codes are discussed. Systematic procedures for the correction of Bose-Chaudhuri codes are given; it is shown that these are quite fast but practical only for small numbers of random errors. It is seen that there is the capability of simultaneous independent and burst error correction.

1. Introduction

An outstanding problem in coding theory is that of designing decoding devices which are both simple and fast. It is made interesting and troublesome by the fact that, generally, speed is achieved only through the acceptance of a large memory. Although various techniques have been found that soften this dilemma by reducing the amount of information that must be stored, they usually do so at the cost of some increase in decoding time. The method to be described here is intended to achieve a similar purpose for cyclic codes. It operates by dividing the decoding problem into a first part, in which the error pattern is found, then into a second, in which the position of the error is established.

We will show that the method allows a significant reduction in the number of bits that must be stored without greatly increasing the decoding time. It must be observed, however, that the method does not give the designer an indefinite freedom to accept slightly decreased speed as a fair exchange for a reduction in storage requirements. Rather, for long code lengths or large numbers of errors the storage requirements indeed become prohibitively large.

From the received data we will calculate the error syndrome, and in part 2 we will see that it is possible to calculate, from the syndrome, certain functions which depend only on the error pattern. Part 3 gives a sufficient condition for consideration of these functions to realize the full error correcting capability of the code involved. In part 4 we will treat the special case of double-error correcting Bose-Chaudhuri codes, and in part 5 will extend these results to triple errors, or a higher multiplicity of errors. Finally, we will consider burst error correction and the simultaneous correction of bursts and random errors.

The necessary algebra can be found in Peterson[1] or Van Der Waerden.[2] A discussion of Bose-Chaudhuri codes can be found in References (1) and (3), and of general decoding methods for these codes in Peterson[1,4] and Zierler and Gorenstein.[5] The special case of double-error correcting Bose-Chaudhuri codes has been considered by Melas,[6] and Banerji.[7] Our method is actually an improvement and a generalization beyond Banerji's method. Other proposals for the decoding of Bose-Chaudhuri codes include those of Chien[8] and Kasami.[9] The detection of burst errors by logical circuitry was proposed by Meggitt.[10]

2. Definitions and preliminaries

Let α be a primitive element of the finite field with q elements, GF(q). We suppose that q is a power of two. Let n be $q - 1$, the order of the multiplicative group and the length of the code. Consider a subset of elements in that field with elements β_i, $1 \leq i \leq N$. Define m_i by $\beta_i = \alpha^{m_i}$. Let $g(x)$ be the polynomial of least degree with coefficients in GF(2) containing all β_i as roots.

The polynomial $g(x)$ defines a cyclic code which is the ideal generated by $g(x)$ in the algebra of polynomials modulo $x^n - 1$. Henceforth, all polynomial operations will be modulo $x^n - 1$. Any code word can be written $i(x)g(x)$, where $i(x)$ represents information by some method of encoding. Suppose $e(x)$ is a polynomial representing an error in transmission. Without loss of generality, we can find s such that $e(x) = x^s f(x)$, and $f(x)$ is relatively prime to x. Then the received vector $r(x)$ is given by:

$$r(x) = i(x)g(x) + e(x) = i(x)g(x) + x^s f(x). \qquad (1)$$

* The author is presently a candidate for the Ph.D. in the Department of Electrical Engineering, Princeton University. He spent the summer of 1964 at the IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y.

We will define certain ratios, R_{ij}, that are error-position invariant, and use these to detect error patterns, $f(x)$, only. Once the pattern has been determined, the location(s) can be determined easily. Define:

$$R_{ij} = \frac{r^{m_i}(\beta_i)}{r^{m_i}(\beta_j)}. \tag{2}$$

Substituting (1), and noting that $g(\beta_k) = 0$:

$$R_{ij} = \frac{\beta_i^{sm_i} f^{m_i}(\beta_i)}{\beta_j^{sm_i} f^{m_i}(\beta_j)} = \frac{\alpha^{sm_i m_i} f^{m_i}(\beta_i)}{\alpha^{sm_j m_i} f^{m_i}(\beta_j)} = \frac{f^{m_i}(\beta_i)}{f^{m_i}(\beta_j)}. \tag{3}$$

We see that R_{ij} is a function only of the pattern $f(x)$, and not of the position s.

If $f^{m_i}(\beta_j) = 0$, we write that $R_{ij} = \infty$. In decoding schemes, this condition must be detected and ∞ treated as though it were an element of the finite field.

In our system, we will carry an extra bit with each ratio, which is 1, iff [if and only if], the element represented is infinity. In the arithmetic, the properties usually attributed to ∞, such as $a + \infty = \infty$, apply here as well.

Theorem 1: Given the error pattern $f(x)$, the location may be found iff there is a set of β_i such that $f(\beta_i) \neq 0$ and the least common multiple of the orders of these β_i is n.

Proof: $r(\beta_i) = \beta_i^s f(\beta_i)$, hence $\beta_i^s = r(\beta_i)/f(\beta_i)$ gives s modulo the order of β_i. Over all elements in the subset, the residues of s modulo the order of each β_i together determine s modulo the L.C.M. of these orders. Since s is restricted between 0 and $n - 1$, it is apparent that s is unique.

For necessity, suppose that the L.C.M. of the orders of all β_i for which $f(\beta_i) \neq 0$ was L, and $L < n$. Then there will certainly be some solution, s_o, to the equations $\beta_i^s = r(\beta_i)/f(\beta_i)$ treated as modular equations in s. It is clear that $s_o + L$ is not congruent to s_o mod n. But $s_o + L$ must also satisfy all equations, so s_o is not unique.

Theorem 2: If $\beta_i = \beta_j^2$, then $R_{ik} = R_{jk}^2$, and $R_{ki} = R_{kj}^2$.

Proof:

$$R_{ik} = \frac{f^{m_k}(\beta_i)}{f^{m_i}(\beta_k)} = \frac{f^{m_k}(\beta_j^2)}{f^{2m_i}(\beta_k)} = \frac{f^{2m_k}(\beta_j)}{f^{2m_i}(\beta_k)} = R_{jk}^2.$$

Likewise for $R_{ki} = R_{kj}^2$.

Two field elements are said to be conjugate if they are roots of the same irreducible polynomial with coefficients in some ground field.[2] Specifically, we will call elements of GF(q) conjugate if they are roots of the same irreducible polynomial with coefficients in the binary field. If β is a root of an irreducible polynomial, so are β^2, β^4, β^8 etc., and these are all the roots of that polynomial.[1] Hence, we have the following corollary:

Corollary 1: In any decoding procedure dependent on R_{ij}'s alone, it is unnecessary that the set of β_i's include more than one element from each irreducible polynomial.

Proof: As in Theorem 2, we may easily show $R_{ij} = 1$ if β_i and β_j are conjugate and that, if R_{ik} is known, R_{jk} can be directly calculated.

Theorem 3: The ratio R_{ij} divides the set of polynomials $f(x)$ into equivalence classes which identify all $f(x)$ for which R_{ij} takes on a fixed value. These classes have the same multiplicative structure as the field elements whose values R_{ij} assumes.

Proof: If we identify $f_1(x)$ and $f_2(x)$ iff $R_{ij}(f_1) = R_{ij}(f_2)$ then surely:

a) $f_1(x) \equiv f_1(x)$
b) $f_1(x) \equiv f_2(x)$ implies $f_2(x) \equiv f_1(x)$
c) $f_1(x) \equiv f_2(x)$ and $f_2(x) \equiv f_3(x)$ implies $f_1(x) \equiv f_3(x)$.

Also, if $f_3(x) = f_1(x)f_2(x)$,

$$R_{ij}(f_3) = \frac{f_3^{m_i}(\beta_i)}{f_3^{m_i}(\beta_j)} = \frac{f_1^{m_i}(\beta_i) f_2^{m_i}(\beta_i)}{f_1^{m_i}(\beta_j) f_2^{m_i}(\beta_j)} = R_{ij}(f_1) R_{ij}(f_2). \tag{4}$$

As a result, it is never mandatory to store or calculate ratios corresponding to any but irreducible polynomials, as the rest could be computed by field multiplication.

3. The central theorem

We will first prove a simple lemma which is necessary in the proof of Theorem 4, then state the central theorem, Theorem 5, which tells us under what conditions the values R_{ij} are sufficient to realize the full error correcting capabilities of the code.

Lemma 1: If α is primitive in GF(q) and $\beta = \alpha^m$, then β is primitive iff there is a unique m-th root in GF(q) for every element in the field.

Proof: Consider the homomorphism that sends α to α^m. Suppose α^{k_1} and α^{k_2} go to the same element under the mapping. Then $\alpha^{k_1 m} = \alpha^{k_2 m}$, $\alpha^{m(k_1 - k_2)} = 1$ and $\beta^{(k_1 - k_2)} = 1$. If β is primitive, n divides $(k_1 - k_2)$, but then $\alpha^{k_1} = \alpha^{k_2}$.

Conversely, if β is not primitive, there is some $(k_1 - k_2)$ less than n such that $\beta^{(k_1 - k_2)} = 1$. But then α^{k_1} and α^{k_2} are distinct and their m-th powers go to the same element $\alpha^{k_1 m}$. This element does not have a unique m-th root.

The central theorem will follow almost directly from the following theorem:

Theorem 4: Let $g(x)$ have among the set of β_i, one root of each irreducible factor, and let $f_1(x)$ and $f_2(x)$ be two errors with $R_{i1}(f_1) = R_{i1}(f_2)$ for all i. Suppose further that $f_1(\beta_1) \neq 0$, and β_1 is primitive in GF(q). Then f_1 and f_2 cannot be distinguished by any decoding method.

Proof: We will show that there exist $i_1(x)$, $i_2(x)$, and s such that

$$i_1(x)g(x) + x^s f_1(x) = i_2(x)g(x) + f_2(x) \tag{5}$$

$$g(x)[i_1(x) + i_2(x)] = x^s f_1(x) + f_2(x). \tag{6}$$

To show Eq. (6) we simply show that there is some s such that $g(x)$ divides $x^s f_1(x) + f_2(x)$.

We are given that

$$\frac{f_1^{m_1}(\beta_i)}{f_1^{m_i}(\beta_1)} = \frac{f_2^{m_1}(\beta_i)}{f_2^{m_i}(\beta_1)}, \tag{7}$$

for all i. Since β_1 is primitive, we can find s such that

$$\beta_1^s = f_2(\beta_1)/f_1(\beta_1) \quad \text{or} \quad \beta_1^s f_1(\beta_1) + f_2(\beta_1) = 0. \tag{8}$$

We now wish to show:

$$\beta_i^s f_1(\beta_i) + f_2(\beta_i) = 0 \tag{9}$$

for all i. Equation (9) is certainly true if $f_1(\beta_i) = 0$ since, from Eq. (7), $f_1(\beta_i) = 0$ implies $f_2(\beta_i) = 0$. If $f_1(\beta_i) \neq 0$ then, from Eqs. (7) and (8)

$$(\beta_1^s)^{m_i} = \frac{f_2^{m_i}(\beta_1)}{f_1^{m_i}(\beta_1)} = \frac{f_2^{m_1}(\beta_i)}{f_1^{m_1}(\beta_i)}. \tag{10}$$

But,

$$(\beta_1)^{s m_i} = \alpha^{s m_1 m_i} = (\beta_i^s)^{m_1}. \tag{11}$$

Hence,

$$(\beta_i^s)^{m_1} = \left[\frac{f_2(\beta_i)}{f_1(\beta_i)}\right]^{m_1}. \tag{12}$$

Since β_1 is primitive, by Lemma 1 there is a unique m_1-th root, hence

$$\beta_i^s = \frac{f_2(\beta_i)}{f_1(\beta_i)}, \tag{13}$$

and Eq. (9) is satisfied. Since $x^s f_2(x) + f_1(x)$ is annihilated by a root of each irreducible factor of $g(x)$, it must be divisible by $g(x)$ and the theorem is proven.

Theorem 5: If all β_i's are primitive, then any two errors which are distinguishable by any method are distinguishable by examining their ratios R_{ij}.

Proof: Let the errors $f_1(x)$ and $f_2(x)$ have all ratios identical. If $f_1(\beta_i) = 0$ for all β_i, the same is true of $f_2(\beta_i)$. Hence $g(x)$ divides $f_1(x)$ and $f_2(x)$ and neither is detectable. If there is some β_i such that $f_1(\beta_i) \neq 0$, let β_i take the place of β_1 in Theorem 4. Then $f_1(x)$ and $f_2(x)$ are not distinguishable by any method since there are received vectors which could have occurred by either error pattern.

Since β_i is primitive, Theorem 1 must apply, hence the position can always be determined if not all $f_1(\beta_i)$ are zero. We mention also that the error patterns are determined by a set of ratios $R_{1j}, R_{2j}, \cdots, R_{nj}$ whenever $f(\beta_j) \neq 0$. It is not necessary to consider the whole triangular array of ratios at any time. We also note that R_{ij} and R_{jk} determine R_{ik} unless both are zero or both infinity.

4. Double error correction

The advantage of solving independently for position and error pattern is that at each step there are fewer variables than if the decoding took place all at once. This advantage is especially important in the double error case, since, eliminating position, we have only one unknown involved. Banerji[7] has proposed a procedure for decoding Bose-Chaudhuri codes for which $\beta_1 = \alpha$, $\beta_2 = \alpha^3$. He looks at the function we call $(1 + R_{21})$ and notices that any double error in a code of length n can be represented by $f(x) = x^k + 1$ where $k \leq \frac{1}{2}(n-1)$. He then proposes that the value of k be identified by the corresponding value of $(1 + R_{21})$.

We will show, first of all, that a double error is uniquely determined by R_{21}, hence by $(1 + R_{21})$. We will also show that it is not necessary to store all the correspondences between R_{21} and k. Let $f(x) = x^k + 1$, $n > k > 0$. Then:

$$R_{21} = \frac{f(\alpha^3)}{f^3(\alpha)} = \frac{\alpha^{3k} + 1}{(\alpha^k + 1)^3}. \tag{14}$$

Let $\gamma = \alpha^k$. Since $\gamma \neq 1$,

$$R_{21} = \frac{\gamma^3 + 1}{(\gamma + 1)^3} = \frac{\gamma^2 + \gamma + 1}{\gamma^2 + 1}, \tag{15}$$

or

$$\gamma^2 + \frac{1}{1 + R_{21}}\gamma + 1 = 0, \tag{16}$$

provided $R_{21} \neq 1$; in that case α would be zero, from Eq. (15), hence no double error could have an R_{21} of 1. However, $R_{21} = 1$ does indicate a single error, and it is good to check that a single error can be distinguished from a double error by R_{21} alone.

The roots of Eq. (16) can be called γ and γ^{-1}, since the product of the roots is 1. If $\gamma = \alpha^k$, then $\gamma^{-1} = \alpha^{n-k}$. However, $x^k + 1$ and $x^{(n-k)} + 1$ represent the same error pattern in a cyclic code of length n and so it is sufficient to find either root of Eq. (16) to decode.

Theorem 6: Equation (16) has solutions for γ in GF(q) *when R_{21} takes on exactly one less than half the values of elements in that field. If there is a solution in* GF(q) *for $R_{21} = R$, then there is a solution in* GF(q) *for all R's conjugates.*

Proof: $\gamma^2 + 1 = 0$ implies $\gamma = 1$ since $x^2 + 1 = 0$ has a double root $x = 1$. Hence, any $\gamma \neq 1$ determines an R_{21} by Eq. (15). $\gamma = 0$ determines $R_{21} = 1$, which cannot be put in the form of Eq. (16). Hence, $q - 2$ elements represent solutions to Eq. (16) for some R_{21}. Since the roots come in pairs, determining the same R_{21}, there are solutions for only $(q/2) - 1$ values of R_{21}.

Suppose γ in GF(q) is a solution for $R_{21} = R$.

$$\gamma^2 + \frac{1}{1 + R}\gamma + 1 = 0. \tag{17}$$

Squaring, we obtain

$$(\gamma^2)^2 + \frac{1}{1 + R^2}(\gamma^2) + 1 = 0. \tag{18}$$

Hence, γ^2 is a solution of Eq. (16) in GF(q) for $R_{21} = R^2$. Likewise, for the rest of R's conjugates. We can say, in fact, that for a set of conjugate R's their corresponding solutions are themselves conjugates.

As a consequence, we can give the following modification of the look-up procedure:

(1) Store one R from each set of conjugates and a corresponding γ.
(2) Compare the computed R_{21} with each stored R; if a match is found, read out that γ.
(3) If no match is found in (2), square each R and γ; repeat the comparison.
(4) If after $\log_2 q - 1$ squarings, no match has been found, then there is no solution to Eq. (16).

We now give an over-all decoding procedure for the double-error case:

(1) Compute the syndromes $r(\alpha)$, $r(\alpha^3)$; if both are zero, assume no error exists.

(2) If either is not zero, compute

$$R_{21} = \frac{r(\alpha^3)}{r^3(\alpha)} = \frac{f(\alpha^3)}{f^3(\alpha)};$$

if $R_{21} = 1$, assume a single error has occurred at location s where $\alpha^s = r(\alpha)$.

(3) If $R_{21} \neq 1$, as computed in the previous step, perform the look-up and comparison outlined above.

(4) If a solution γ is found, then find s where $\alpha^s = r(\alpha)/\gamma + 1$.

(5) If a solution γ is not found, say an error has been detected.

An example will be given later and a system for implementing the procedure will be shown.

5. Higher multiplicity of errors and bursts

In the case $\beta_1 = \alpha, \beta_2 = \alpha^3$, we were able to show specifically that the equations involved have a unique solution and that the look-up mapping is one-to-one.

In the case of triple or higher order errors, or even of double errors with syndromes corresponding to α and α^k with $k > 3$, it is hard to prove the uniqueness of solutions for the error patterns. However, if we know that the code may be decoded by any method—for example, the Peterson procedure[4]—and Theorem 5 is satisfied, we have a round-about proof that there will be a unique set of ratios corresponding to any error in the correctable class. We can generalize Theorem 6.

Theorem 7: Let $f(x) = 1 + \sum_k x^{p_k}$ *and* $\gamma_k = \alpha^{p_k}$. *Suppose that the* γ_k *satisfy, for all i and j, the relation:*

$$R_{ij}(1 + \sum_k \gamma_k^{m_i})^{m_j} = (1 + \sum_k \gamma_k^{m_j})^{m_i}. \tag{19}$$

Then if R_{ij} *is squared, the substitution of* γ_k^2 *for each* γ_k *is a solution to Eq. (19).*

Proof: Square Eq. (19). Note that $f(\beta_i) = 1 + \sum_k \gamma_k^{m_i}$.

Hence, we need store only one representative from a set of R_{ij}'s and corresponding γ_k's.

We note that in general, if $f(\beta_1) \neq 0$ it is sufficient to decode using only the R_{i1}'s. For those $f(x)$'s with $f(\beta_1) = 0$, $f(\beta_2) \neq 0$, we use the R_{i2}'s etc. In general, we may not need to compute all R_{ij}'s. For example, it is possible that no error we are trying to decode has $f(\beta_1)$ and $f(\beta_2)$ simultaneously zero. In that case it would be sufficient to compute the R_{i1}'s and R_{i2}'s.

The storage required becomes large very quickly. As a lower bound, we assume that in GF(q), squaring a set of R_{ij}'s produces $\log_2 q$ distinct sets. Since $x^q = x$, no more than $\log_2 q$ distinct sets may be produced. The number of distinct e-tuple error patterns, remembering that each pattern can be represented by e polynomials of degree $n - 1$ or less, is

$$\frac{(n-1)!}{(n-e)!\,e!}.$$

We note that each storage represents $(e - 1)$ field elements, of $\log_2 q$ bits; hence, the number of solutions to be stored (see Fig. 1) is,

$$S \geq \frac{(n-1)!}{(n-e)!\,e!\,\log_2 q}, \tag{20}$$

and the bit storage is

$$S_b \geq \frac{(n-1)!\,(e-1)}{(n-e)!\,e!}. \tag{21}$$

The redundancy which the Bose-Chaudhuri codes require to correct a given multiplicity of errors is enough so that many sets of ratios do not correspond to an error in the class of known correctable errors. We can assign to a set the error in its equivalence class having the greatest probability of occurence. Unfortunately, there seems to be no way of telling how much additional redundancy is necessary to correct additional error patterns such as bursts. In practice, the burst-length we can correct depends on which β we choose, for if we wish to correct a specified set of errors, we need a unique identification of error by ratio set.

However, in the burst case, if we have a set of β_i sufficient to correct bursts of specified length, we can use Theorem 3 to advantage and reduce the storage. We select a set of small degree polynomials, and a set of

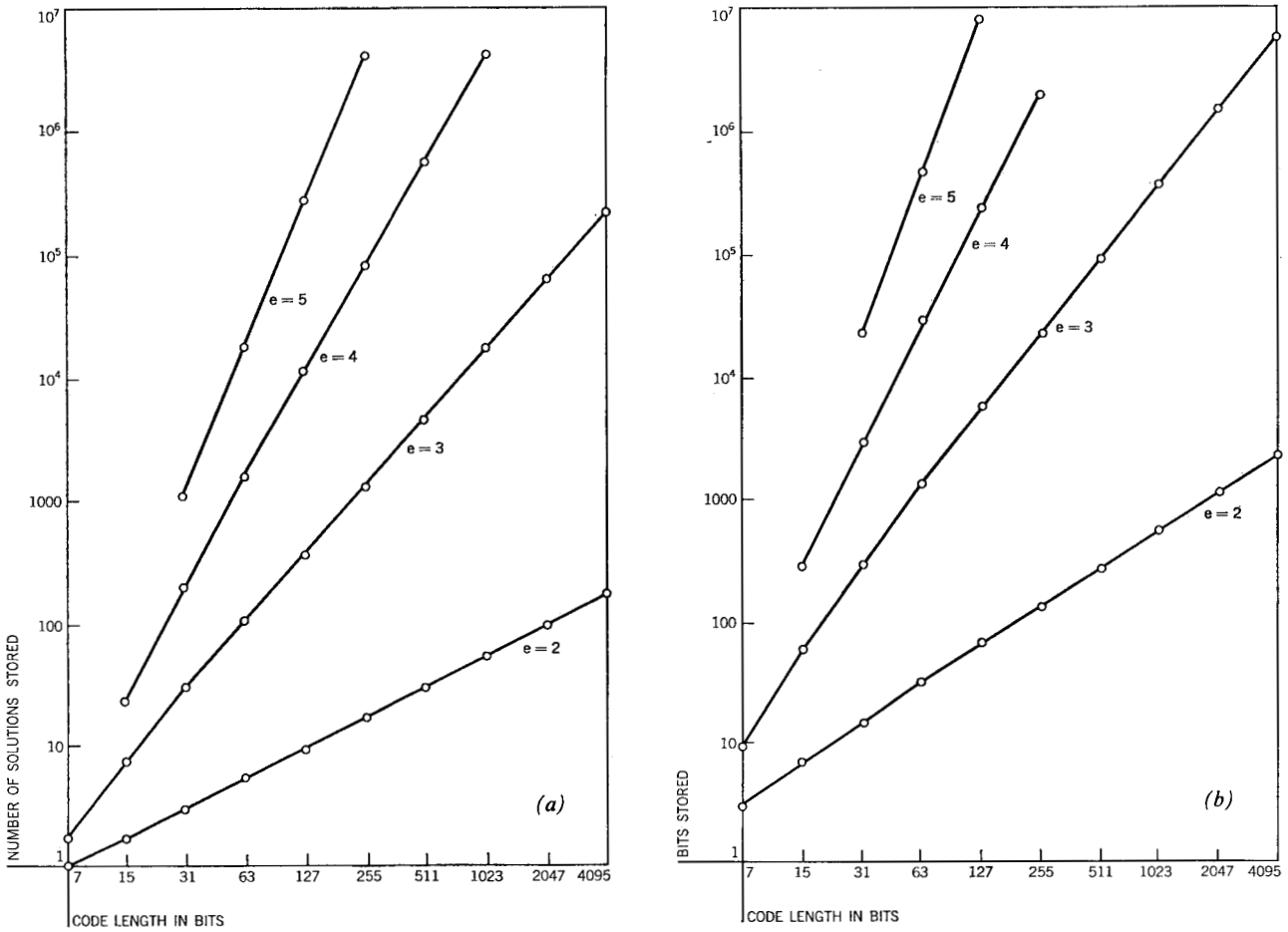


Figure 1 Storage requirements and their relation to code length in bits: a) number of solutions stored; b) bits stored.

bursts whose ratios are stored. The bursts are successively multiplied by each of the small degree polynomials, while the ratios are multiplied by the ratios of the small polynomial.

The number of bits stored is reduced, because if all bursts of degree b are to be corrected, and the small polynomials are of degree d or less, every burst of degree $(b - d)$ or less generates another burst in the class for each small polynomial chosen.

There is no procedure known to the author whereby a minimal number of bursts can be selected to generate all bursts of length b. The following procedure would seem to produce a reasonable result, for the case where the set of small polynomials is that of all polynomials of degree d or less and none of higher degree.

For degree $i = (n - 1)$ down to $i = (b - d)$ we eliminate all polynomials of degree higher than i that are the product of a polynomial of degree i and one of degree less than d. Suppose that a polynomial $f(x)$ of degree $j > i$ is eliminated by $g(x)$ of degree i, and that previously, $h(x)$ of degree $k > j$ had been eliminated on the strength of $f(x)$'s presence. Then $h(x)/g(x) = h(x)/f(x) \times f(x)/g(x)$ is a polynomial of degree less than d, and so $g(x)$ generates $h(x)$. Thus, every burst of degree $(b - d)$ or greater is either selected or generated by the product of a stored burst and a polynomial of degree d or less.

When i is less than $(b - d)$ we no longer have the assurance that $h(x)/g(x)$ is of degree less than d, and so cannot continue with the procedure. We can instead eliminate polynomials only if they are not the only way to generate some polynomial of degree higher than $(i + d)$.

There is no guarantee that this procedure produces an optimal selection. However, we note that the only bursts of degree b that are not eliminated are those that have no factor of degree d or less. These polynomials could could not be eliminated by any scheme, and so the number of bursts of degree b is minimum. Since the number of bursts of length b is equal to the number of bursts of all lower degrees, we have an indication that the procedure is good.

Note that this procedure concerns itself with bursts only, and does not take into account that some "bursts"

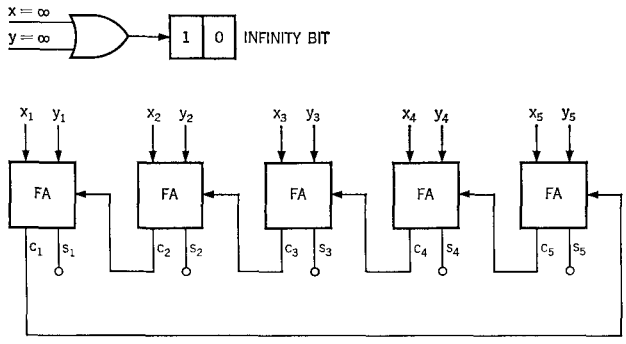


Figure 2 The squaring device diagrammed.

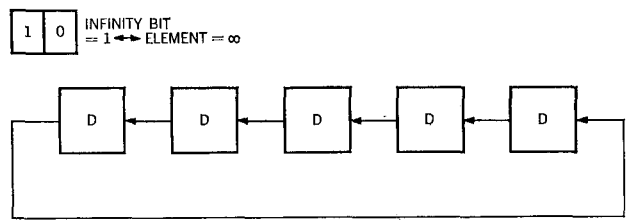


Figure 3 The multiplier diagrammed.

Table 1 The correspondence between k and R_{21}. See text at left and Eq. (16).

Table 1

k[a]	$(31 - k)$	i[b]
1	30	6
2	29	12
4	27	24
8	23	17
16	15	3
3	28	22
6	25	13
12	19	26
24	7	21
17	14	11
5	26	18
10	21	5
20	11	10
9	22	20
18	13	9

[a] Where $f(x) = x^k + 1$
[b] Where $R_{21} = \alpha^i$

can be decoded as random errors. If the code is intended to correct bursts and random errors, some burst storage may be saved.

6. An example

We will consider an example of a double-error correcting code and "design" the decoder involved. The system is required to perform several field additions, multiplications, and exponentiations; so, the method by which we represent the field will to a large extent determine the complexity of the circuitry. One common method is to generate the field GF(q) in terms of powers of a root of a polynomial over the binary field, which is $\log_2 q$-th degree, primitive and irreducible. In this case, addition is performed by component-wise modulo 2 addition, but multiplication is complex.

For our purposes, it is simpler to represent the element α^k by a binary representation of k. All zeros will represent the field element zero, and all ones will represent α^n, the field element unity. The value ∞ will be denoted by a special bit.

Squaring becomes simple, especially, as it is merely a cyclic shift to the left, as is shown in Fig. 2. Multiplication in the field becomes a cyclic addition, as is shown in Fig. 3. Unfortunately, addition must be done using complex logic, but there is not too much addition to be performed; when necessary, we can convert to the first notation mentioned, add by modulo 2 addition, and reconvert.

Our example will be that of a (31, 21) Bose-Chaudhuri double-error correcting code. We generate GF(2^5) using as α, a root of $x^5 + x^2 + 1 = 0$. Let $\beta_1 = \alpha$, $\beta_2 = \alpha^3$. Table 1 gives the correspondence between k and R_{21}, if $\gamma = \alpha^k$, as determined by Eq. (16). Note that the first five entries in each column are conjugate; likewise, the second five and the last five. Every error is represented either in the column headed k or in that headed $(31 - k)$. We will generate all necessary correction data by storing the fact that $R_{21} = \alpha^6$ corresponds to $k = 1$, $R_{21} = \alpha^{22}$ corresponds to $k = 3$ and $R_{21} = \alpha^{18}$ corresponds to $k = 5$.

Since performing the squaring operations would require hardware, we will avoid that task as much as possible. Instead of squaring the stored R_{21}'s we will square the calculated one only, and take square roots of the stored solution γ. The square root operation corresponds to a cyclic right shift, if squaring is a cyclic left shift.

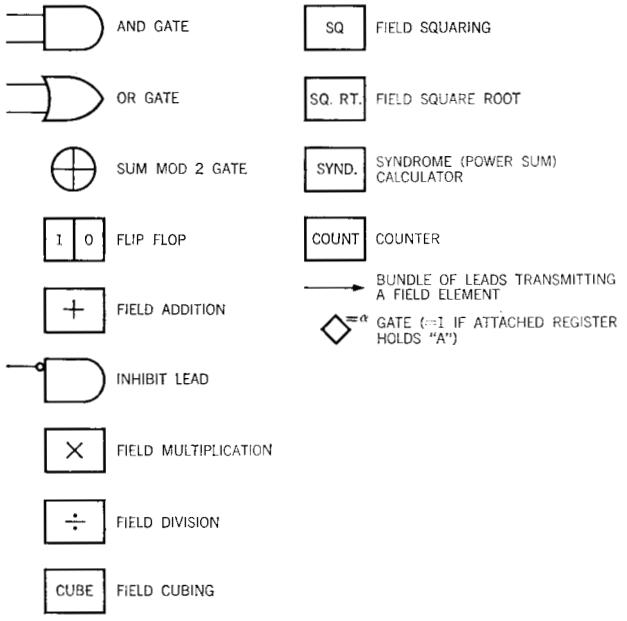
Figure 4 represents the system to decode double errors. In it the clock outputs separately accomplish the following:

- Output A resets counter, solution storage register, and the flip-flop which releases the data from the buffer.
- Output B delivers, at the appropriate time, pulses that cycle the squarers and counts to make sure that detection occurs after four such cycles.
- Output C releases, at the maximum time to detect the error, the output to the buffer by originating a level signal.
- Output D reduces the counter by one for each bit read out of the buffer.

The operation of the system is as follows:

(1) Input is read into a buffer, and its syndromes calculated.

Figure 4 The double error corrector. The notation is as tabulated below:



(2) The squaring cycle counter and the solution storage are reset.

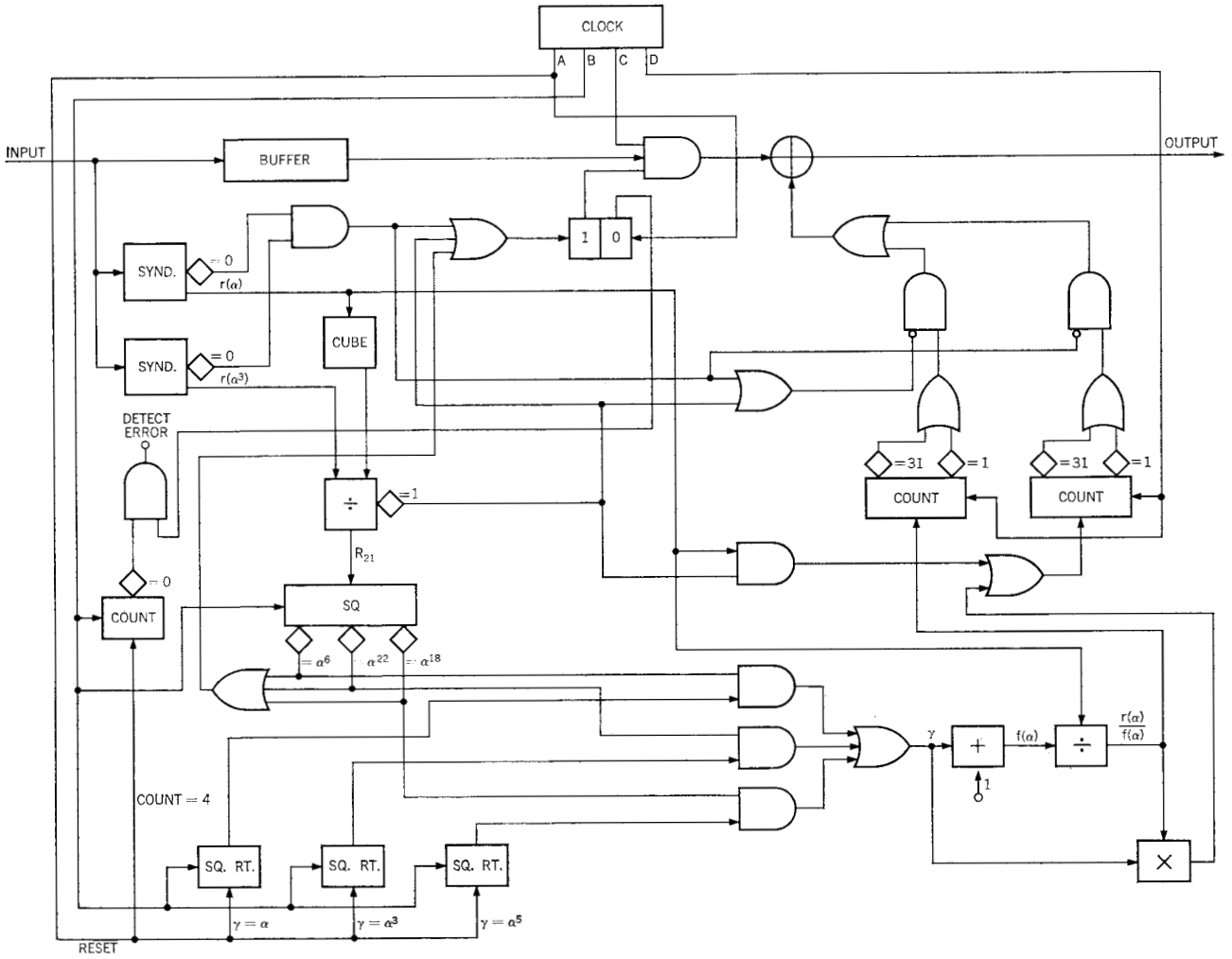
(3) If it is detected that both syndromes are zero, both counters for correction are inhibited and the system awaits the C signal from the clock.

(4) Otherwise, calculate R_{21}; if the condition $R_{21} = 1$ is detected, inhibit one counter, loading $r(\alpha)$ into the other.

(5) If $R_{21} \neq 1$, load it into the squarer. At this time, five pulses from Output B will cycle R_{21} through all its conjugates, simultaneously using the square root operations to calculate the solution γ.

(6) When a match is found, the corresponding γ is gated into a unit which calculates $f(\gamma) = \gamma + 1$, and s where $\alpha^s = r(\gamma)/f(\gamma)$. Note that s is the representation for α^s, so that α^s is loaded directly into one counter. The other counter is loaded with $\gamma\alpha^s$.

(7) If no match is found, an error detect signal is given.



(8) Otherwise, level signal C begins shifting data out of the buffer. If the count on either counter is one, the bit is corrected. Pulse D reduces the count by one after each bit that emerges from the buffer. Note that α^0 is represented by all 1's and so that case must be detected separately.

7. Conclusions

The method presented offers two advantages, speed and the ability to handle burst and independent errors, or burst errors alone. The disadvantage is that storage goes up exponentially with either the number of independent errors, or the length of burst.

For the independent errors, the speed is primarily determined by the number of times the ratios must be squared to insure a match, in other words, by the logarithm of the code length. For burst errors, speed depends on how many small polynomials are used. As a result, it should be possible to construct a decoder that operates in a little more time than it takes to calculate the syndromes. The actual time is determined by what sort of arithmetic units one is prepared to use.

A lower bound on the storage is given by Eq. (21); it is seen that for large numbers of errors, the requirements are too great to make this a feasible method.

When storage is small, this method uses quantities of equipment commensurate with or less than that employed with other methods[4,8,9] and operates at higher speeds. Also, the storage required is an order of magnitude less than that of Banerji[7] in the double-error case.

It would appear that for many cases involving short codes or small numbers of errors the method described will offer distinct advantages.

Acknowledgement

The author thanks Dr. R. T. Chien for his many helpful comments and suggestions.

References

1. W. W. Peterson, *Error Correcting Codes*, John Wiley and Sons, New York, 1961.
2. B. L. Van Der Waerden, *Modern Algebra*, Fredrick Ungar Publishing Co., 1949.
3. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and Control*, **3**, 68–79, 1960.
4. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IRE Transactions on Information Theory*, **IT-6**, 459–70, 1960.
5. N. Zierler and D. Gorenstein, "A Class of Error Correcting Codes in p^m Symbols," *Journal of the Society for Industrial and Applied Mathematics* **9**, 2, pp. 207–14.
6. C. M. Melas, "A Cyclic Code for Double Error Correction" *IBM Journal* 4, 3 (1960).
7. R. B. Banerji, "A Decoding Procedure for Double Error Correcting Bose-Ray-Chaudhuri Codes," *Proceedings of IRE* 49, (1961).
8. R. T. Chien, "A Cyclic Decoder for Bose-Chaudhuri-Hocquenghem Codes," *IEEE Transactions on Information Theory* **IT-10**, 357–62, 1964.
9. T. Kasami, "A Decoding Procedure for Multiple Error Correcting Cyclic Codes," *IEEE Transactions on Information Theory* **IT-10**, pp. 134–8, (1964).
10. J. E. Meggitt, "Error Correcting Codes for Correcting Bursts of Errors," *IBM Journal* 4, 1 (1960).

Received March 19, 1965