Some New Results on Majority-Logic Codes for Correction of Random Errors.

BY

David McQuilton, B.Sc., M.Sc.

A Doctoral Thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology.

September 1978.

Supervisor: Dr. M.E. Woodward,
Department of Electronics and
Electrical Engineering.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the help and advice of Dr. M.E. Woodward for supervising this work and to the Electrical and Electronic Engineering Department for making this project available.

The author would also like to express his appreciation and gratitude to his wife, Mitra, and her family for their help and patience.

Gratitude is also extended to the Science Research Council without whose financial assistance this thesis would not have been possible.

Finally thanks are expressed to Mrs. B. Wright whose typing of this work can only be stated as excellent.

CONTENTS

PART I

RANDOM ERROR-CORRECTING MAJORITY-LOGIC DECODABLE BINARY BLOCK CODES.

CHAPTER 1

1. INTRODUCTION.

1.1 Merits of Majority-logic Coding.

The main advantages of random error-correcting majority-logic codes and majority-logic decoding in general are well known and two-fold. Firstly, they offer a partial solution to a classical coding theory problem, that of decoder complexity. Secondly, a majority-logic decoder inherently corrects many more random error patterns than the minimum distance of the code implies is possible.

The solution to the decoder complexity is only a partial one because there are circumstances under which a majority-logic decoder is too complex and expensive to implement.

The optimum system is a one-step decoder (developed by Massey[4]) for a cyclic code, which only requires one majority-logic gate to decode all received digits in a block. If the code is non-cyclic a one-step decoder requires one majority-logic gate for each information digit in a block. Therefore if a non-cyclic code had a large number of information digits the decoder may no longer be economic to implement.

Some codes are majority-logic decodable in L-steps[4] and this requires the decoder to have L levels of majority-logic gates, each level except the L'th having more than one majority-gate, to decode a single received digit. Thus a decoder of this type could only be viable economically if the code is cyclic, unless the number of information digits is small. Also with L-step decoders, if the random error-correcting capability of the code rises, the number of inputs to each majority-gate rises and the number of gates on each level rises too. Unfortunately as L rises the decoder complexity rises exponentially so there are also economic limitations on L-step decoding.

However L-step decoding is useful for two reasons, (a) one can correctly decode, theoretically, in the presence of a larger number of random errors, than is possible with one-step decoding, providing of course one can find the code to do it, and (b) recent developments[2,16 17,20] have lead to the discovery of large classes of good cyclic codes that can be decoded using L-step decoding.

An alternative to L-step decoding has developed around results presented by Rudolph[9], which is a one-step decoding method. Although it is a one-step decoding method, the single majority-gate requires a very large number of inputs. When used in place of L-step decoding for an L-step decodable code, this one-step method corrects less random errors and is as approximately as expensive as L-step decoding also. The one-step method can be converted into a two-step method and thereby correct as many random errors as L-step but with a large increase in decoder complexity. Since Rudolph's[9] one-step method and L-step[4] decoding are more complex than Massey's[4] one-step method, the search for codes that are Massey one-step decodable and can compete with cyclic codes which are not majority-logic decodable, is still a useful area of research.

The second advantage of majority-logic codes is with regard to their performance when more random errors occur than the code can guarantee to decode. This is sometimes referred to as a code's unbounded performance. The number of random errors a code can guarantee to decode is related to a parameter of the code called minimum distance. It has been an established practice in coding theory to compare codes on the basis of their minimum distance for a given code length and effective rate of transmission. However it is widely accepted that the minimum distance gives no information regarding the unbounded performance of a code and as such is a crude measure of the potential performance in the presence of random errors. This is particularly true of majority-logic codes since they have an inherent ability to correctly decode on many occasions, when more errors occur than the code can guarantee to decode.

1.2 Code Assessment.

One would intuitively feel that a useful measure of a codes performance could be achieved by simulating the noise encountered on a realistic transmission path, called a channel. That is, establish a noise record of a standard realistic channel over which we could subjectively assess any codes performance. Unfortunately no such channel exists and the Engineer must resort to one of two alternatives: Simulate the codes on the inary symmetric channel, or try to establish a typical noise record of the realistic channel available, trying different codes to find some 'best' code for that channel.

The former is usually resorted to by coding theorists while the latter is usually adopted by the industrial based Engineer.

There are difficulties encountered with both approaches; that is,

i) there are many different codes one can try whose only **apparent** measure of performance is the minimum distance,

ii) there is usually more than one technique whereby one can decode a given type of code, each with relative merit,

iii) many realistic channels are time-varying being different from day to day depending on ambient temperature, weather or sunspots, etc. Thus to obtain a realistic noise record may be difficult.

Therefore many coding theorists still compare codes on the basis of length, rate and minimum distance, leaving the assessment on a particular channel to the Engineer who is interested.

The engineer or newcomer who encounters coding theory for the first time, or particular codes, may be bewildered or indeed put off completely by the dazzling array of mathematical techniques and various mathematical disciplines required of him in order to achieve some understanding of the subject. This shows no sign of easing and the author feels that there is a need for more literature which treats the subject from an engineering viewpoint.

1.3 Project Resumé.

The codes developed in Chapters 4 and 5 are treated from an engineering view in the sense that they are constructed in an algorithmic fashion. The reader only requires a knowledge of the basics of coding theory, in particular majority-logic decoding, and this is presented in Chapter 3.

Any further mathematical theory is presented when required. Chapter 4 presents a method of developing codes which are shown to be cyclically decodable in one-step using simple shift-registers and a single majority-logic gate, with appropriate gating. They are shown to be completely orthogonalizable and some assessment is made of their properties for large length n.

In Chapter 5 further codes are simply developed from the codes of Chapter 4 by shortening and extending the generator matrix.

Comprehensive examples of how to develop codes are given in both chapters. Many codes are developed which meet the maximum upper bound on minimum distance from Helgert and Stinaff[40].

The author feels that once one has grasped the basic method of construction one can join in the fun of constructing the codes for oneself.

In Chapter 6 it is shown how to construct tables of minimum distances for groups of binary k-tuples. Codes can then be developed for which, by reference to the table, one can find the minimum distance by simple addition and see the weight spectrum of the code words in the code. No method of decoding these is presented.

Finally in Chapter 7 an assessment is made of the codes developed and they are compared with existing cyclic block codes on the basis of rate, length, and error-correcting ability.

CHAPTER 2

2. SURVEY OF MAJORITY-LOGIC DECODABLE BLOCK CODES.

2.1 Initial Results.

The first majority decoding algorithm was devised by Reed[30], in 1954, who developed the algorithm to decode a class of non-cyclic codes developed by Muller[31]. In 1958 Yale[32] and Zierler[33] showed that the cyclic Maximum length sequence codes could be majority-logic decoded and in 1961, Mitchell[37] showed the same for the (15,7), (21,11) and (73,45) B.C.H.[35,36] codes and the Hamming[34] codes. In 1963, Massey[4] unified the theory of majority-logic decoding and introduced the terms, one-step and L-step decoding and type 1 and type 2 decoding. He also showed that all B.C.H.[35,36] codes of length $n \leq 15$ are majority-logic decodable. In the same year Gallager[38] introduced his Low density parity check codes, decoded with a majority algorithm. Two new classes of majority-logic codes were presented by Weldon[39] and Townsend and Weldon[28] in 1966 and 1967 respectively. The former are a small class of Difference-set cyclic codes and the latter a class of quasi-cyclic Self-orthogonal codes. Other codes shown to be majority-logic decodable are graph theoretic codes[38]. From all these initial results majority-logic decodable codes were still not competitive with the powerful cyclic codes such as B.C.H. codes.

Interestingly, following research into Finite geometry codes, the Difference-set codes[39] and Maximum length sequence codes[32,33] were found to be special cases of Projective geometry codes[1]. The Reed[30]- Muller[31] codes were also found to be special cases of

Euclidean geometry codes[1]. It was the advent of Finite geometry codes that was to produce the big breakthrough, for majority-logic codes, into the areas which would make them highly competitive with the best cyclic codes.

2.2 Finite-geometry Codes.

In 1964, in his Masters Thesis[44], Rudolph considered the application of finite-geometries to the construction of majority-logic cyclic codes. Among his results were constructions for two large classes of majority-logic decodable codes based upon Euclidean and Projective geometries. However Rudolph did not show how to obtain the generator polynomial or minimum distance of the codes generally and this was done by Weldon[45,46,2]. Also, in Rudolph's approach to the codes[44,9] he proposed a decoding scheme based upon the use of non-orthogonal check sums which he showed would decode a large number of errors in one-step. But this method did not decode all errors which the codes were capable of decoding. Weldon[45,46,2], not only showed that the codes could correct more errors using the L-step Reed[30]- Massey[4] Algorithm but he also showed[47] that more errors could be corrected using Rudolph's non-orthogonal algorithm if decoding was performed in 2-steps. In terms of decoder complexity there is little to choose between 2-step non-orthogonal and L-step Reed[30]- Massey[4] decoding. In the same year, 1968, Kasami, Lin and Peterson[48] along with Weldon[46], investigated generalisations of the original Reed[30]- Muller[31] (R.M), codes after Kasami[49] et al. had shown they were equivalent to cyclic codes. They showed that while the cyclic equivalent original R.M. codes were a special case of Euclidean Geometry codes, both Euclidean Geometry (E.G.), codes and Projective Geometry codes (P.G.), were special cases of Generalised R.M. (G.R.M.), codes. Also in the same year Kasami[50] et al. defined a class of polynomial codes which contained the G.R.M. codes and B.C.H. codes as special cases. Lin[51] also produced results on this relationship to E.G. and P.G. codes. Chow[61] examined B.C.H. codes and found a class that could be decoded with non-orthogonal parity checks in one-step though the number of errors decoded was less than the code was capable of. He also showed that some double error-correcting B.C.H. codes could not be L-step orthogonalized. Duc[64] also obtained results on B.C.H. codes after obtaining necessary conditions for linear codes to be L-step decodable. In particular he showed that some triple error-correcting B.C.H. codes, most binary quadratic residue codes[5] and all B.C.H. codes of length n = 127, except two, cannot be L-step orthogonalized.

In 1969 Delsarte[52] proposed a new class of Generalised Finite Geometry codes (G.F.G.), which included Generalised E.G. and Generalised P.G. codes, (G.E.G.) and (G.P.G.) respectively. His work was extended by Lin and Weldon[53] and Hartmann and Rudolph[54] and it was seen that the G.F.G. codes were more efficient than the regular E.G. and P.G. codes.

Although these geometry codes are majority-logic decodable, nevertheless for large error-correcting capability, large L or high order geometries the complexity of the decoder was still prohibitive economically in many instances. Weldon[47] originally proposed techniques for reducing the number of steps in decoding but this only applied to R.M. and E.G. codes. In an effort to improve on this Chen[55,56] proposed an algorithm which reduced the number of decoding steps to decode E.G. and P.G. codes.

At the same time (1971) Kasami and Lin[26], following upon the connection between polynomial codes and geometry codes, showed that certain dual codes of primitive polynomial codes could be decoded using majority-logic decoding. Their decoding method involved the use of different decoding algorithms at different levels, beginning with non-orthogonal check sums and finishing with orthogonal check sums.

Various work due to Chen[57], Lin[58] and Chen and Warren[59] produced results on new codes obtained by shortening E.G. and P.G. codes. In particular Lin[58] showed that an E.G. code is actually a shortened P.G. code while Chen[59] et al. developed a procedure for finding shortened P.G. and E.G. codes that are 1-step decodable. Chen[57] shortened by deleting parity and information digits. Following on from the work of Chen[55,56] on reducing decoder complexity, Rudolph and Hartmann[60] proposed a new decoding method called "Sequential code reduction". The method permitted a significant reduction in decoder complexity, for all cyclic codes, with a modest increase in decoding time. They applied their results to the decoding of E.G. and P.G. codes with the definite result that all F.G. codes with length $n \leq 2047$ can be decoded by a restricted sequential code reduction algorithm with one majority gate at each of the L stages. Also majority sequential code reduction (S.C.R.) retains the majority-logic decoding property of being able to correct many error patterns of weight greater than the algorithm is designed for.

In the same year, 1973, Lin[17] published a new class of G.E.G. codes which contains the E.G. and some other G.E.G. codes. The decoding algorithm utilizes different decoding algorithms at different levels, notably, non-orthogonal then orthogonal check sums. These "Multifold" codes, to be efficient, require an inefficient base code on which to extend. A particular sub-class of these codes were claimed as the most efficient majority-logic decodable cyclic codes at that time. Hartmann[16], et al. presented new results also on the structure of G.E.G. and G.P.G. codes. Hybrid decoding is the term used by Hartmann[16], et al. to denote the use of non-orthogonal and orthogonal check sums on differing levels. Rudolph's[44,9] one-step non-orthogonal decoding algorithm is extended to an L-step algorithm and sequential code reduction advised to reduce complexity, however hybrid decoding is in general recommended. Further G.E.G. codes were presented by Lin[20], in 1975, as improved "Multifold" codes and were more efficient. Again hybrid decoding is used though some of the codes can be decoded in one-step.

Warren and Chen[11] introduced new very efficient codes derived from the shortening of E.G., "Multifold" E.G. and G.E.G. codes. The shortening technique developed around results obtained from the shortening of generalised polynomial codes. A new class of cyclic codes of even length were found. Though, in general, the shortened codes were non-cyclic they could be decoded cyclically using the encoder and decoder of the parent code.

2.3 Further Results.

The idea of decoding using a threshold other than that given by the Reed[30]- Massey[4] algorithm or the Rudolph[9] algorithm was first introduced by Massey[4] who showed that a more realistic method was to weight each check sum according to the probability that it would be in error given an error pattern. This resulted in the decoding procedure having the least average probability of a decoding failure, or error. Massey[4] called this A Posteriori probability (A.P.P.) decoding.

In 1969 a spate of papers on generalised threshold decoding appeared from Gore[72,63], Townsend and Weldon[28] and Rudolph[8]. Rudolph[8] showed that there existed a threshold decoder using generalised parity checks and a single threshold element though he did not show how one could obtain the parity checks or how many would be needed, for a code, generally. He did state that decoding rules have been found for the two perfect Golay codes using this procedure. Townsend[28], et al. presented a general decoding procedure in which a threshold element had a variable threshold which was adjusted according to whether correct or incorrect decodings occurred on a single digit. Many error patterns of weight greater than the codes capacity could be decoded but at the price of a large increase in decoding time.

Gore[62] showed that any linear code could be decoded in less than or equal to k levels of a generalised threshold decoder. In the same year Gore[63] examined the threshold decoding of Reed-Solomon codes and showed that they are not L-step decodable though they are threshold decodable.

The non-orthogonal decoding algorithm of Rudolph[9] was improved by Ng[7] who showed that the algorithm could correct more errors in certain circumstances if the identity (or zero) check sum were allowed more than one vote.

In 1971, Duc[19], proposed another new decoding algorithm which advocated the use of mixed orthogonal and non-orthogonal check sums, at the input to a single majority-logic gate. Duc justified his algorithm by showing that some codes may be majority-logic decoded, though they were known not to be Reed[30]- Massey[4] algorithm or Rudolph[9] algorithm decodable.

In 1972, Rudolph and Robbins[18] modified Rudolph's[8] statement regarding threshold decoding by showing that in principle any binary linear code could be one-step weighted-majority decoded by replacing the threshold element of Rudolph[8] by a weighted-majority element.

As an example Rudolph[18] decoded the code used as an example by Duc[19]. The code was decoded in one-step using Duc's[19] algorithm, Ng's[7] improvement and a weighted-majority scheme.

Also in 1971 Bobrow[69] showed that certain cut-set graph theoretic codes could be 2-step decoded. Then Kasami[70] et al. presented new majority-logic codes derived from combining existing majority-logic codes. The resulting codes are generally L-step decodable.

Various results on the majority-logic decoding of product codes have been obtained, initially, by Lin[65] et al. and Gore[66]. Lin[65] et al. showed that a product code, formed from a one-step majority-logic code with minimum distance d_1 and an L-step majority-logic code with minimum distance d_2, was L-step decodable with minimum distance $d_1 \cdot d_2$. Gore[66] showed that two codes, L_1-step and L_2-step decodable, formed a product code $(L_1 + L_2 - 1)$-step decodable. Later Duc[67] et al. showed that two codes, one decodable using non-orthogonal parity checks and one L-step decodable, formed a product code L-step decodable with the first step non-orthogonal and the remaining L-1 steps orthogonal. If both codes are non-orthogonally decodable the product code is similarly decodable. The following year, 1973, Duc[68] improved the algorithm of Lin's[65] et al. enabling correction of more errors in the product codes.

In the same year Chien and Liu[71] presented a new class of 2-step decodable arithmetic codes and a new class of L-step decodable extended arithmetic codes. For an introduction to arithmetic codes see Peterson and Weldon[2], chapter 15.

In 1974, Hashim[15] et al. presented a new class of low rate majority-logic codes based upon Walsh functions. Shiva[13] et al. showed that the subset code of a binary majority-logic code was also majority-logic decodable. Examining binary cyclic codes Riek[12] et al. showed that certain cyclic codes are majority-logic decodable if their parity check polynomial falls into a certain class. Hybrid decoding is used in general and sequential code reduction advised to improve the algorithm.

Rudolph's original paper[9] on projective geometry codes decoded using non-orthogonal check sums also related the codes structure to the combinatorial aspects of Balanced incomplete block designs, (B.I.B.D.). This aspect of majority-logic codes has been investigated by other researchers such as Goethals[71,73], Assmus and Mattson[71,72,74], and Rahman and Blake[75,76]. Assmus[72,74] et al. and Goethals[73] independently devised a one-step majority-logic decoding algorithm for the extended (24,12) Golay code and the (48,24) Quadratic residue code. Rahman[75] et al. showed that Ng's[7] improvement on Rudolph's[9] original algorithm could be further improved by generalising the combinatorics. As an example he considered the number of errors the 1st, 2nd and 3rd order Reed-Muller codes could correct compared to Ng's algorithm. Rahman[76] et al. later examined the construction of one-step decodable codes based upon supplementary difference sets. An infinite family of single and double error-correcting codes was found. Evidence is given of an infinite family of triple error-correcting codes.

Finally, interest has been shown in what is called "Soft decision" decoding which is a decoding procedure in which probabalistic information is used in conjunction with error-correcting codes in order to improve system performance. Sundberg[77] soft decodes by using reliability information to tell him which digit in a block is most likely to be in error. Then the next most likely erronous digit and so on, all digits being one-step majority-logic decoded. For a helpful discussion see also Harrison[78].

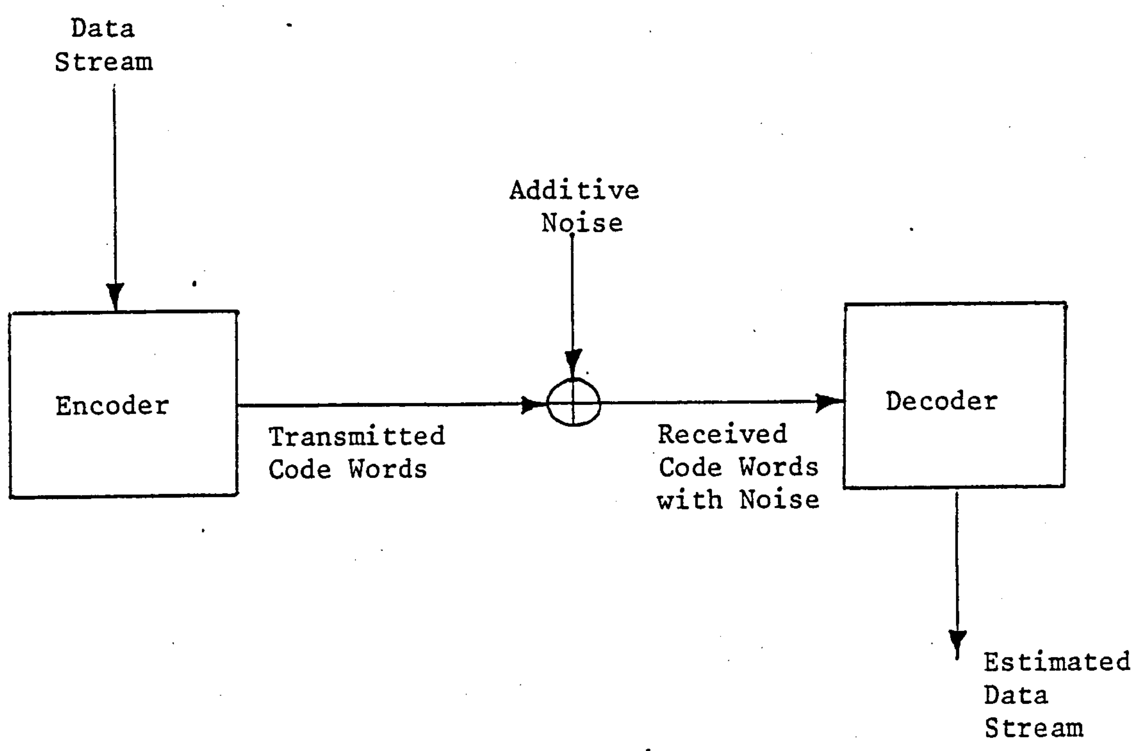


FIG. 3.1.1.

SIMPLE DATA-COMMUNICATION SYSTEM.

CHAPTER 3

3. CODING THEORY.

3.1 Linear Block Codes.[1,29]

To implement a code we, in general, require (a) an information source, (b) an encoder which constructs and transmits a set of code words related to the information source, (c) a channel, over which the code words are transmitted, which introduces errors in the code words, and (d) a receiver or decoder which, knowing all possible code words, attempts to recognise a received code word and remove its errors.
A simple block diagram incorporating the above is shown in figure 3.1.1.
We will assume that the information is in the form of successive binary digits. The encoder functions by sub-dividing the information stream into blocks of k binary digits. To each block it assigns a unique code word of length $n > k$ binary digits, called a binary n-tuple. The rules for assignment are determined by the code being used so that a binary code is specified by a set of 2^k distinct binary n-tuples from the set of 2^n n-tuples.

At the receiver each received code word is treated independently, without reference to any previously received data. The decoder attempts to discover, which block of k binary digits determined the code word, in the presence of errors. The independent treatment of the data, block by block, defines a block code. In addition, to be a linear code, the following definition on the 2^k n-tuples is necessary.

Definition 3.1.1.[1]

A set of 2^k binary n-tuples is called a linear code if and only if it is a subspace of the vector space, called V_n, of all binary n-tuples.

This subspace is also referred to as the code space.

3.2 The Generator Matrix.

Although a linear block code can be specified by the list of 2^k binary n-tuples, it is unrealistic to do this if k is large. However, any k-dimensional subspace S, of V_n, can be specified by a set of k basis n-tuples, $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_k$ such than any n-tuple $\overline{u} \in S$ can be represented by a linear combination of the basis set.[1]

Let,

$$\overline{u} = m_1 \overline{v}_1 \oplus m_2 \overline{v}_2 \oplus \cdots\cdots\cdots \oplus m_k \overline{v}_k \qquad 3.2.1.$$

where

$\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_k$ are the basis n-tuples

$m_i \in$ G.F.(2), for $1 \leq i \leq k$

$\oplus$ = summation over G.F.(2).

If the k basis n-tuples form the rows of a k × n matrix G, then

$$G = \begin{bmatrix} \overline{v}_1 \\ \vdots \\ \vdots \\ \overline{v}_k \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots\cdots & v_{1n} \\ v_{21} & v_{22} & \cdots\cdots & v_{2n} \\ \vdots & & & \\ v_{k1} & v_{k2} & \cdots\cdots & v_{kn} \end{bmatrix} \qquad 3.2.2.$$

Then equation 3.2.1. can be represented as,

$$\overline{u} = \overline{m} \cdot G \qquad 3.2.3.$$

where

$$\bar{m} = \begin{bmatrix} m_1, & m_2, \ldots, & m_k \end{bmatrix}$$

can be the $1 \times k$ row matrix representing the block of k information digits. The rows of G generate a linear code and G is called the generator matrix of the code.

If G and G' are the generator matrices of two codes, then if by rearranging the columns of G we can obtain G', then the codes are said to be "equivalent".[29]

If G' can be obtained from G by a combination of row and column permutations, then G and G' are said to be "combinatorially equivalent".[29]

Every generator matrix G is combinatorially equivalent to one G' in echelon canonical form.[29]

That is we can arrange G in the form,

$$G = \begin{bmatrix} I_k & P \end{bmatrix} \qquad 3.2.4.$$

where

$I_k = k \times k$ identity matrix

$P = k \times n\text{-}k$ matrix

by means of row and column permutations. However, it is not always prudent to do this as the original structure of G may be required in decoding. Codes whose generator matrix has the natural form of equation 3.2.4. are called "Systematic Codes".[1,29]

The first k digits of every systematic code word are reproductions of the k digits in the information block. The n-k digits generated by P are called parity-check digits.

The following theorem is from Peterson and Weldon.[29]

Theorem 3.2.1.

Every linear code is equivalent to a systematic code.

A code specified by a $k \times n$ generator matrix G is also referred to as an (n, k) code and is said to have a code rate, $R = k/n$, **the ratio of Information content before and after coding.**

3.3 The Parity-check Matrix.

If $\bar{a} = (a_1, a_2, \ldots, a_n)$ and $\bar{b} = (b_1, b_2, \ldots, b_n)$ are two binary n-tuples we can define their inner product as,

$$\bar{a} \cdot \bar{b} = a_1 b_1 \oplus a_2 b_2 \oplus \ldots \oplus a_n b_n \qquad 3.3.1.$$

and if $\bar{a} \cdot \bar{b} = 0$ we say that $\bar{a}$ and $\bar{b}$ are orthogonal. If S_1 is the subspace of n-tuples of a code generated by G, then the set of all n-tuples orthogonal to S_1 is also a subspace, S_2, called the null space of the code.[29] The parity-check matrix of a code, H, is an $n-k \times n$ matrix whose rows are basis n-tuples of the null space of the space generated by the generator matrix, G, of the code.

Let,

$$H = \begin{bmatrix} \bar{h}_1 \\ \vdots \\ \bar{h}_{n-k} \end{bmatrix} = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & & \\ h_{n-k,1} & \cdots & h_{n-k,n} \end{bmatrix}$$

then

$$GH^T = 0 \qquad 3.3.2.$$

where H^T is the transpose matrix of H. In particular, $\bar{u}\, H^T = 0$, for any $\bar{u} \in S_1$.

The subspace generated by H is also a code space and is called the dual code of that generated by G.

Let an (n,k) code have generator matrix G and parity-check matrix H and let $\bar{u}$ be a code word transmitted over a noisy channel. If $\bar{e}$ is a binary n-tuple representing the noise added to $\bar{u}$ during transmission, it has digits of binary 1 in those positions where errors occurred in $\bar{u}$. Let $\bar{r}$ be the n-tuple received by the decoder, then if errors have occurred,

$$\bar{r} = \bar{u} \oplus \bar{e} \qquad 3.3.3.$$

therefore,

$$\begin{aligned} \bar{r}\, H^T &= (\bar{u} \oplus \bar{e}) H^T \\ &= \bar{u}\, H^T \oplus \bar{e}\, H^T \\ &= \bar{e}\, H^T = \bar{s} \end{aligned} \qquad 3.3.4.$$

where $\bar{s}$ is an n-k digit binary word called the syndrome. Obviously if $\bar{e} = \bar{0}$ then $\bar{s} = \bar{0}$ and the decoder knows no errors were present. Otherwise the decoder uses the information in the syndrome to find the errors. Each digit s_i, of $\bar{s}$, is obtained by forming the inner product of $\bar{r}$ and $\bar{h}_i$ from H.

If the generator matrix is in echelon canonical form, as in equation 3.2.4., then H can be found quite simply using the following theorem from Peterson and Weldon.[29]

<u>Theorem 3.3.1.</u>

If S is the code space of the generator matrix $G = \left[I_k \; P \right]$, where I_k is a $k \times k$ identity matrix and P is a $k \times (n-k)$ matrix, then S is the null space of $H = \left[-P^T \; I_{n-k} \right]$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix.

3.4 The Error-correcting Capability.

If $\bar{u}$ and $\bar{v}$ are two code words of a linear code, then since the code space is a subspace, $\bar{u} \oplus \bar{v}$ is also a code word. Therefore if $\bar{e}$ has the form of a code word and $\bar{r} = \bar{u} \oplus \bar{e}$, then $\bar{r}$ is also a code word and $\bar{r}\, H^T = \bar{0} = \bar{s}$. so that the decoder cannot decode $\bar{e}$, furthermore if $\bar{e}_1$, $\bar{e}_2$ are two different error words and $\bar{v}$, $\bar{u}$ are two code words, then

if $\quad \bar{e}_1 \oplus \bar{u} = \bar{e}_2 \oplus \bar{v} = \bar{r}$, we have

$$\bar{r}\, H^T = \bar{e}_1\, H^T = \bar{e}_2\, H^T = \bar{s} \qquad 3.4.1.$$

and the decoder cannot identify $\bar{e}_1$ and $\bar{e}_2$ uniquely. These situations are examples of what happens when the number of errors exceeds the capability of the code.

Consider the following definitions.

Definition 3.4.1.[1]

The Hamming weight of a binary n-tuple $\bar{u}$, $w_H(\bar{u})$, is the number of binary ones in $\bar{u}$, i.e. if $\bar{u}$ = (10010110001), $w_H(\bar{u}) = 5$.

We can now introduce the concept of distance between binary n-tuples, which will lead to results on the error-correcting capability.

Definition 3.4.2.[1]

The Hamming distance, $d_H(\bar{u}, \bar{v})$, between two binary n-tuples, $\bar{u}$ and $\bar{v}$, is defined as the number of components (digits) in which they differ, i.e. if,

$$\bar{u} = 10010110001$$

$$\bar{v} = 11001010101$$

then $\quad d_H(\bar{u}, \bar{v}) = 5$.

It is apparent that,

$$d_H(\bar{u}, \bar{v}) = w_H(\bar{u} \oplus \bar{v}). \qquad 3.4.2.$$

Since $\overline{u} \oplus \overline{v}$ is another code word, say $\overline{y}$,

$$d_H(\overline{u}, \overline{v}) = w_H(\overline{y}).$$

The minimum value of $d_H(\overline{u}, \overline{v})$, obtained by forming all possible sums of pairs of n-tuples, $\overline{u}$ and $\overline{v}$, from a code space, is called the minimum distance of the code, d_m. Let $d_m = d_H(\overline{a}, \overline{b}) = w_H(\overline{a} \oplus \overline{b})$, but $\overline{a} \oplus \overline{b}$ is another code word, so that the minimum distance is equal to the minimum weight of the non-zero code words in a code.

If the errors which occur affect each digit of a transmitted code word, independently, they are called random errors and codes designed for this type of error are called random-error-correcting codes.

If the errors which occur tend to be strung together in bursts the codes designed to combat these are called burst-error-correcting codes.

We will confine the rest of the discussion to random-error-correcting codes.

Let us assume that a generator matrix G specifies an (n, k) code with minimum distance d_m. Let $\overline{u}$ and $\overline{r}$ be the transmitted and received n-tuples respectively and $\overline{v}$ any other code word. For maximum likelihood decoding the decoder will identify $\overline{r}$ with that code word which has the minimum Hamming distance between itself and $\overline{r}$. Therefore if

$$d_H(\overline{v}, \overline{r}) < d_H(\overline{u}, \overline{r}) \qquad 3.4.3.$$

the decoder will choose $\overline{v}$ and thereby incorrectly decode $\overline{r}$. However, since from LIN[1],

$$d_H(\overline{v}, \overline{r}) + d_H(\overline{u}, \overline{r}) \geqslant d_H(\overline{v}, \overline{u}) \qquad 3.4.4.$$

if $$d_H(\overline{u}, \overline{r}) \leqslant \left[\frac{d_m - 1}{2}\right]$$

where $\left[x \right]$ means the largest integer $\leqslant (x)$.

Then from equation 3.4.4.

$$d_H(\overline{v}, \overline{r}) \geqslant d_H(\overline{v}, \overline{u}) - \left[\frac{d_m - 1}{2} \right]$$

and since the minimum $d_H(\overline{v}, \overline{u}) = d_m$,

$$d_H(\overline{v}, \overline{r}) \geqslant \left[\frac{d_m + 1}{2} \right] \qquad 3.4.6.$$

so that we always have,

$$d_H(\overline{u}, \overline{r}) < d_H(\overline{v}, \overline{r}) \qquad 3.4.7.$$

Since from equation 3.3.3.

$$d_H(\overline{u}, \overline{r}) = w_H(\overline{e}) \qquad 3.4.8.$$

then providing,

$$w_H(\overline{e}) \leqslant \left[\frac{d_m - 1}{2} \right] \qquad 3.4.9.$$

the decoder will correctly decode $\overline{u}$ in the presence of all error n-tuples whose weight conforms to the inequality 3.4.9. Conventionally we say, a (n,k) code with d_m can correct any error pattern of $t \leqslant \left[\frac{d_m - 1}{2} \right]$ errors, in a block of n digits. The error-correcting capability of the code is then,

$$t = \left[\frac{d_m - 1}{2} \right] \qquad 3.4.10.$$

errors, as a maximum.

3.5 Decoding Cyclic Codes.

Let $\overline{u}$ be a code word from the (n,k) code generated by G. If the n-tuple code word, $\overline{u}$, is given by,

$$\bar{u} = (u_1, u_2, \ldots, u_n)$$

then the code is said to be cyclic if for every $\bar{u}$,

$$\bar{u}' = (u_n, u_1, u_2, \ldots, u_{n-1})$$

is also a code word.

We can consider the code words as polynomials over G.F.(2), such that if u(x) is the polynomial representing $\bar{u}$,

$$u(x) = u_1 \cdot x^o + u_2\, x + u_3\, x^2 + \ldots\ldots + u_n\, x^{n-1} \qquad 3.5.1.$$

with $u_i \in \text{G.F.}(2) \qquad 1 \leqslant i \leqslant n$

It is known that the code word polynomials of a cyclic code can be represented as multiples of a unique polynomial, g(x), called the generator polynomial. At the encoder the k-digit information block is considered as a polynomial, C(x), where,

$$C(x) = c_{n+1-k}\, x^{n-k} + \ldots\ldots + c_n\, x^{n-1} \qquad 3.5.2.$$

The encoder divides C(x) by g(x) so that we can write,

$$C(x) = q(x)\ g(x) + r(x) \qquad \deg(r(x)) < \deg(g(x)) \qquad 3.5.3$$

so that

$$C(x) + r(x) = q(x)\ g(x) \qquad 3.5.4.$$

which is the transmitted code word.

The degree of the generator polynomial is n-k, so that

$$\deg(r(x)) < n-k \qquad 3.5.5.$$

and r(x) is called the parity-check polynomial.

At the decoder the process is repeated. The received message and check-digits are separated and the message digits divided by g(x). The resulting remainder r"(x) is added to the received check-digits, that is r'(x), to form the n-k digit syndrome, s(x).

Since,

$$s(x) = r'(x) + r''(x) \qquad 3.5.6.$$

if there are no errors, $r'(x) = r(x)$ and $r''(x) = r(x)$, so that $s(x) = 0$, due to G.F.(2) addition of coefficients.

Let $e_r(x)$ be the (n-k)-digit polynomial representing errors in the received check-digits and $e_k(x)$ be the (n-k)-digit polynomial that is the remainder upon dividing the errors in the k-digit received message block by $g(x)$, then

$$\begin{aligned} r'(x) &= r(x) + e_r(x) \\ r''(x) &= r(x) + e_k(x) \end{aligned} \qquad 3.5.7.$$

From equations 3.5.6. and 3.5.7.

$$s(x) = e_r(x) + e_k(x) \qquad 3.5.8.$$

in the presence of errors.

If there are no errors in the received message digits $e_k(x) = 0$, and

$$s(x) = e_r(x) \qquad 3.5.9.$$

and of course this is the actual error pattern in the received code word.

A most useful result for cyclic codes, from Peterson and Weldon[2] p.p.230-232, is given by the following theorem.

Theorem 3.5.1.

Let $s(x)$ denote the syndrome of an n-tuple $R(x)$. The syndrome of a cyclic shift of $R(x)$, say $x\,R(x) \bmod(x^n + 1)$, is obtained by shifting the syndrome generator of $g(x)$, once, with initial contents $s(x)$. Therefore, for cyclic codes, if we can decode one received digit, we can decode all received digits by syndrome shifting. Of course how we use the syndrome bits to decode a digit depends upon the code structure, but there are procedures which are applicable to any cyclic code.

One such procedure is called Error-trapping Decoding[3,22] (E.T.D.). If a code has error-correcting capability t, when a code word is received we form the syndrome and check its weight. If $\leqslant$ t errors have occurred and the errors are trapped in the check-digits, then from equation 3.5.9. the Hamming weight of s(x), written $w_H(s(x))$, gives

$$w_H(s(x)) \leqslant t \qquad 3.5.10.$$

and this error pattern is taken to be the received error pattern.

If, $w_H(s(x)) > t$, the syndrome is shifted cyclically one digit and $w_H(s(x))$ checked again. If $w_H(s(x)) \leqslant t$ we know the errors have been trapped in the check-digits of the code word that is a single cyclic shift of the received code word. The decoded error pattern is then a shifted version of the received error pattern.

However, if $w_H(s(x)) > t$ again, the process is repeated. If we always obtain $w_H(s(x)) > t$ we assume that the error pattern has weight > t or that it is untrappable. Otherwise we assume a shifted version of the error pattern has been decoded and compensate accordingly.

Many variations on E.T.D. have been devised, Kasami[23], MacWilliams[24], Omwra[25], but these seem to be limited to codes of relatively short length.

E.T.D. is successful for all single error-correcting codes and all burst error-correcting codes. For random error-correcting codes with capability t, it can be shown[5] that a necessary and sufficient condition for all error patterns of weight $\leqslant$ t to contain at least k successive zero's is

$$t < \frac{1}{R} , \qquad R = \frac{k}{n} . \qquad 3.5.11.$$

The search for useful decoding procedures for cyclic codes is an active branch of research in the field of error-correcting codes. Before examining another approach to decoding we will clear up at this point some definitions of various types of codes.

a) Pseudo-cyclic codes.[5]

The generator polynomial, $g(x)$, of a cyclic (n,k) code always divides $x^n + 1$. A pseudo-cyclic code has code words generated by $g(x)$ which divides $x^{n'} + 1$, for $n' > n$. Since $g(x)$ generates a cyclic code of length n', the pseudo-cyclic code is that code whose code words are taken from the cyclic code of length n' whose digits $a_n, a_{n-1}, \ldots, a_{n'-1}$ are zero and are dropped.

b) Shortened-cyclic codes.[5]

Given an (n,k) cyclic code with generator matrix G, if we delete the first i columns and rows of G, the resulting code is an $(n-i, k-i)$ shortened cyclic code.

Pseudo-cyclic and shortened cyclic codes are not truly cyclic since there is always some code word whose cyclic shifted version is not in the code.

c) Quasi-cyclic codes.[5]

If $u(x)$ is a code word in a (n,k) quasi-cyclic code, then if $x^{n_o} u(x) \bmod (x^n + 1)$ is another code word, the code is said to be quasi-cyclic of order n_o. Cyclic codes such that m divides n and k are quasi-cyclic of order m, but there exist codes where $x^{n_i} u(x) \bmod (x^n + 1)$ for $n_i < n_o$, is not a code word.

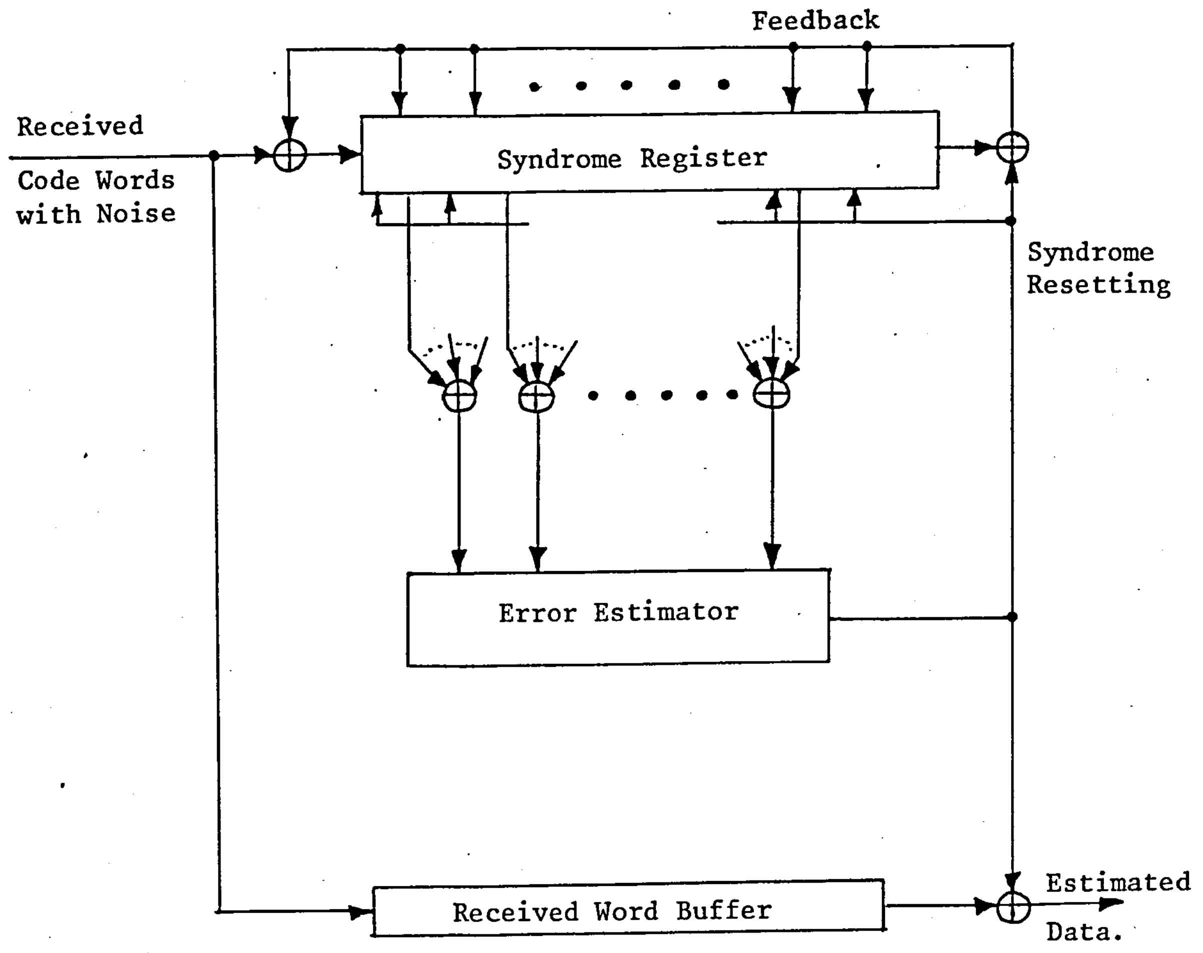


FIG. 3.6.1.

A GENERAL TYPE I DECODER.

The columns of the generator matrix can be considered as being composed of circulant sets of k-tuples of order n_o.

A circulant set of k-tuples, or generally a circulant, is defined as, the set of k-tuples that are all cyclically shifted versions of one k-tuple in the set. If a circulant has n_o distinct k-tuples in its set, it is said to have order n_o. The generator matrix of a quasi-cyclic code is composed of circulants, all of the same order.

3.6 Majority Logic Decoding.[6,2,4]

The idea of majority-logic decoding, (M.L.D.), is based upon the concept of "orthogonal* check sums". A check sum or parity-check sum is a linear equation, or sum, of error digits. From equation 3.3.4. we see that each digit of $\bar{s}$ is a sum of error digits and is thus a parity-check sum. Also the addition of digits from $\bar{s}$ form other check sums. If a check sum s_1 contains e_m, s_1 is said to check e_m. If we assemble all those check sums which check e_m, then they are "orthogonal" on e_m if they conform to the following definition.

Definition 3.6.1.[6]

A set of parity-check sums s_1, s_2,..., s_J is said to be "orthogonal" on the error digit e_m if e_m is checked by each check sum s_i in the set and no other error digit is checked by more than one sum.

There are two basic methods in which the check sums can be formed by the decoder, referred to as Type 1. and Type 2. decoding[4]. Figures 3.6.1. and 3.6.2. show typical Type 1. and Type 2. decoders respectively.

---

* Orthogonal here is not the orthogonal defined in equation 3.3.1.

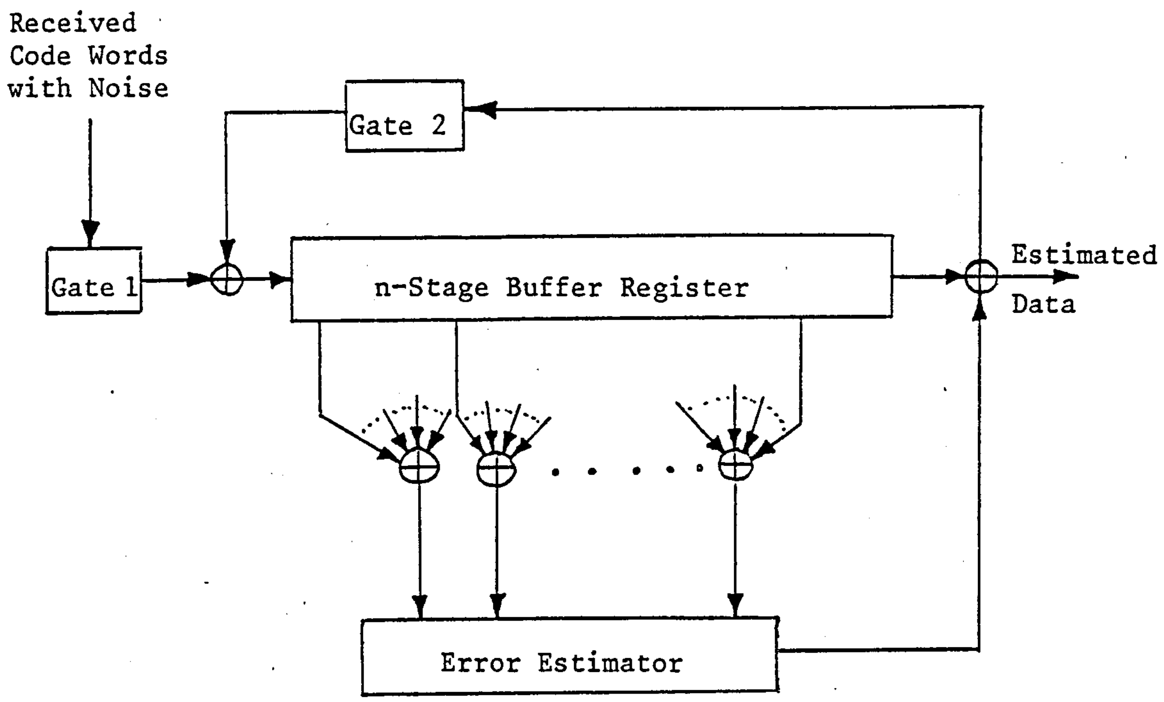


FIG. 3.6.2.

A GENERAL TYPE II DECODER.

With Type 1. decoding we re-encode the message-digits, form the syndrome and form orthogonal check sums from linear sums of syndrome digits.

With Type 2. decoding we utilize a set of J n-tuples, from the null space of the code that are orthogonal* on a received digit. We then form the J inner products with the received word directly, so that if $\bar{b}_i$ is an n-tuple in the null space, we obtain the set of check sums,

$$s_i = \bar{e} \cdot \bar{b}_i \quad \text{for} \quad 1 \leqslant i \leqslant J \,. \qquad 3.6.1.$$

Since a syndrome digit is a linear sum of error digits, that is a check sum, then every linear codes' syndrome digits are a set of check sums. The problem is to find codes whose syndrome digits (or null space n-tuples), can be arranged into sets of orthogonal check sums.

3.7 One-step M.L.D.

Consider the set of J orthogonal check sums below for an (n,k) linear block code.

$$\begin{aligned} s_1 &= e_1 \oplus e_n \\ s_2 &= e_2 \oplus e_n \\ &\vdots \\ s_J &= e_J \oplus e_n \end{aligned} \qquad 3.7.1.$$

Let us assume that J is even and that $t = J/2$ errors have occurred. If e_n is in error then only $(J/2 - 1)$ errors can occur in the other digits, so that there are always, at least, $J - (J/2 - 1) = J/2 + 1$,

* Orthogonal as in equation 3.3.1.

check sums which give a correct estimate of e_n, that is, a majority. If e_n is not in error, then the worst condition is that the J/2 errors affect J/2 check sums and an even split of J/2 0's and 1's occurs. If $t < J/2$ errors occur then the majority always favours e_n.

To obtain a correct estimate of e_n under the circumstances given above, we require an electronic block whose function is to look at the s_i, and output an estimate e'_n of e_n, according to the following rules.

$$\begin{array}{lll} \text{a)} & \text{If } \sum_i s_i > J/2 \ , & e'_n = 1 \\ \text{b)} & \text{If } \sum_i s_i < J/2 \ , & e'_n = 0 \\ \text{c)} & \text{If } \sum_i s_i = J/2 \ , & e'_n = 0 \ . \end{array} \qquad 3.7.2.$$

An electronic unit which performs just such an operation is available and is called a Majority-logic Gate (M.L.G.). For the code above we require a J-input, single output M.L.G. to decode e_n. If the code is non-cyclic, we would require k such M.L.G.'s to decode the errors in the k message-digits. However with a cyclic code, after decoding e_n, if we cyclically shift the syndrome register (Type 1.) or buffer register (Type 2.) one digit, we will obtain an identical set of check sums orthogonal on e_{n-1}. Then all n digits can be decoded sequentially, by successive cyclic shifts. The Error Estimator in figures 3.6.1. and 3.6.2. can then be replaced by k - M.L.G.'s or 1 - M.L.G. depending on the code. In both cases, it will be noted, we have achieved an estimate of the errors by using one M.L.G.

Codes where this is possible are referred to as "One-step M.L.D. codes".

If we could obtain J orthogonal check sums without adding syndrome digits (Type 1.) or received digits (Type 2.), that is the code has orthogonal check sums inherent in its structure, it is said to be a "Self-orthogonal M.L.D. code".

If the code has minimum distance, d_m, then if we can form $J = d_m - 1$ check sums on the errors, we can correct,

$$\frac{J}{2} = \left\lfloor \frac{d_m - 1}{2} \right\rfloor = t \qquad 3.7.3.$$

errors and the code is said to be "completely orthogonalizable". In the case where only $J << (d_m - 1)$ check sums are obtainable then M.L.D. would be considered inefficient for that code.

The development of equations 3.7.2. and 3.7.3. arose from the form of equations 3.7.1., that is, the check sums are linear equations in the errors only. One may alternatively be able to form check sums directly on the message digits, but one then requires

$$J = 2t + 1$$

check sums to decode in the presence of t errors. This can be seen from the check sums below.

$$\begin{aligned} s_1 &= m_1 \oplus e_1 \oplus e_2 \\ s_2 &= m_1 \oplus e_3 \oplus e_s \\ \hline s_3 &= m_1 \oplus e_4 \oplus e_6 \end{aligned} \qquad 3.7.4.$$

Assume one error, in position e_1, and only s_1 and s_2 are used.

i) if $m_1 = 0$, $s_1 = 1$, $s_2 = 0$

and a split vots gives $m_1 = 0$ by majority.

ii) if $m_1 = 1$, $s_1 = 0$, $s_2 = 1$

and a split vote gives $m_1 = 0$ by majority.

The ambiguity arises because m_1 can assume both states, 0 and 1, regardless of errors. If we include s_3, then m_1 is correctly decoded.

Generally with check sums of this form, we require $J = 2t + 1$ check sums, so that if t errors occur and affect t independent check sums, there are still t+1 check sums in favour of the common message-digit.

Since we have,

$$t = \left\lfloor \frac{J - 1}{2} \right\rfloor \qquad 3.7.5.$$

and from equation 3.4.10.

$$t = \left\lfloor \frac{d_m - 1}{2} \right\rfloor$$

complete orthogonality occurs if $J = d_m$, with this form of check sum.

Example 3.7.1.

The quasi-cyclic code with $n = 12$, $k = 6$, $d_m = 4$ with circulant generator[5] 7 has a generator matrix, G, below,

$$G = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&1&1&1 \\ 0&1&0&0&0&0&1&0&0&0&1&1 \\ 0&0&1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&1&0&0&1&1&1&0&0&0 \\ 0&0&0&0&1&0&0&1&1&1&0&0 \\ 0&0&0&0&0&1&0&0&1&1&1&0 \end{bmatrix}$$

After forming the syndrome, we have the following equations for s_i, $1 \leqslant i \leqslant 6$.

$$s_o = e_o \oplus e_9 \oplus e_{10} \oplus e_{11}$$

$$s_1 = e_1 \oplus e_6 \oplus e_{10} \oplus e_{11}$$

$$s_2 = e_2 \oplus e_6 \oplus e_7 \oplus e_{11}$$

$$s_3 = e_3 \oplus e_6 \oplus e_7 \oplus e_8$$

$$s_4 = e_4 \oplus e_7 \oplus e_8 \oplus e_9$$

$$s_5 = e_5 \oplus e_8 \oplus e_9 \oplus e_{10}$$

Let A_1, A_2, A_3 be the $J = d_m - 1 = 3$, check sums.

$$A_1 = s_1 = e_1 \oplus e_6 \oplus e_{10} \oplus e_{11}$$

$$A_2 = s_2 \oplus s_3 = e_2 \oplus e_3 \oplus e_8 \oplus e_{11}$$

$$A_3 = s_o \oplus s_4 \oplus s_5 = e_o \oplus e_4 \oplus e_5 \oplus e_7 \oplus e_9 \oplus e_{11}$$

which are orthogonal on e_{11}.

Furthermore if the syndrome is shifted cyclically

$$s_o = e_5 \oplus e_8 \oplus e_9 \oplus e_{10}$$

$$s_1 = e_o \oplus e_9 \oplus e_{10}$$

$$s_2 = e_1 \oplus e_6 \oplus e_{10}$$

$$s_3 = e_2 \oplus e_6 \oplus e_7$$

$$s_4 = e_3 \oplus e_6 \oplus e_7 \oplus e_8$$

$$s_5 = e_4 \oplus e_7 \oplus e_8 \oplus e_9$$

$$A_1 = s_1 = e_o \oplus e_9 \oplus e_{10}$$

$$A_2 = s_2 \oplus s_3 = e_1 \oplus e_2 \oplus e_7 \oplus e_{10}$$

$$A_3 = s_o \oplus s_4 \oplus s_5 = e_3 \oplus e_4 \oplus e_5 \oplus e_6 \oplus e_8 \oplus e_{10}$$

which are orthogonal on e_{10}.

If the first column of G, generates the 11'th digit of the code word, which we assume is transmitted first, then e_{11}, e_{10}, e_9, e_8, e_7

are the errors in the message-digits. Although the code is not cyclic these message-digit errors can be decoded cyclically.

The code is therefore completely orthogonalizable and one-step majority-logic decodable.

For examples of one-step M.L.D. cyclic difference set codes and maximum-length sequence codes see Shu Lin[6], chapter 7.

3.8 L-step M.L.D.

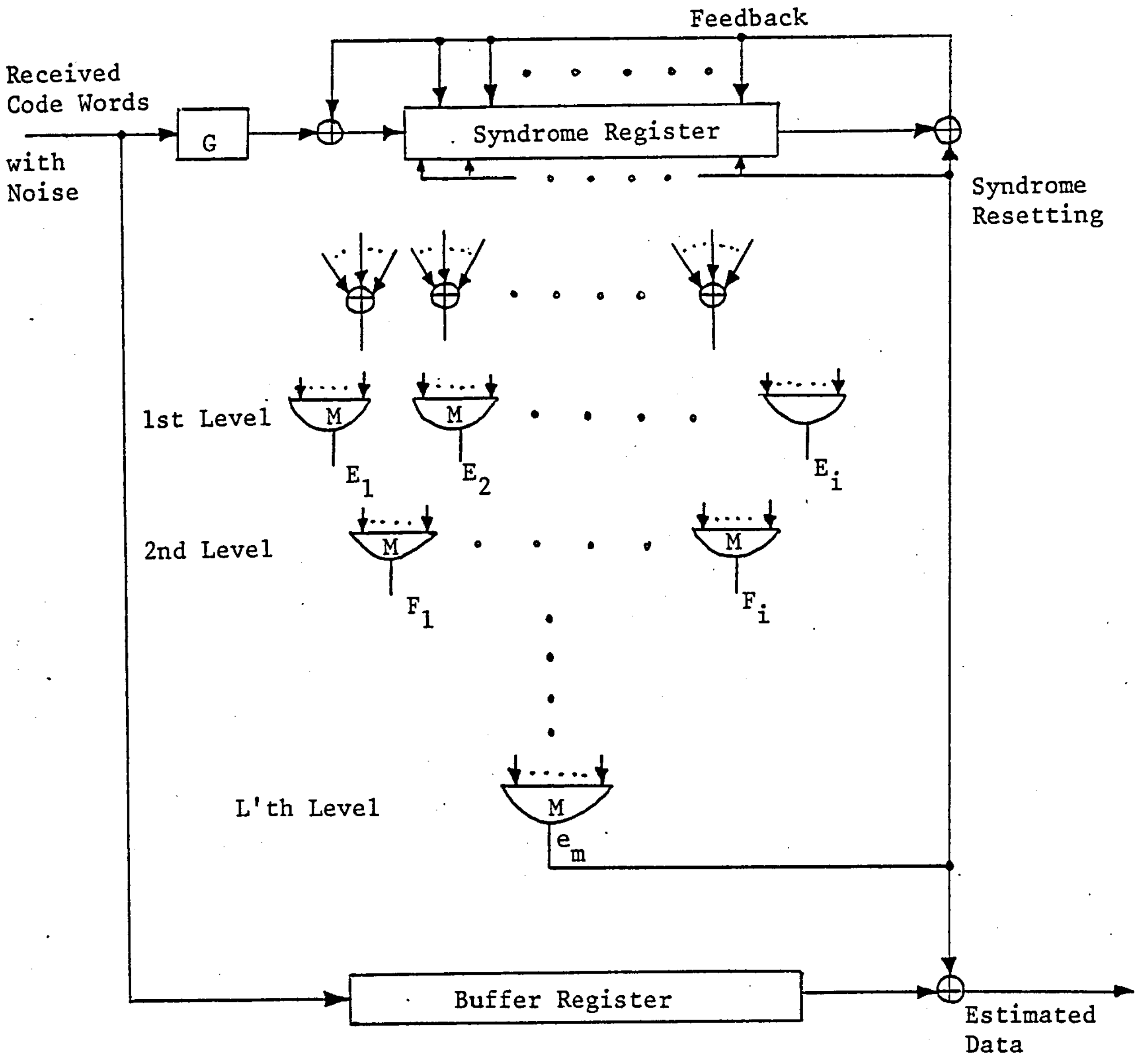
The definition of orthogonal check sums given in definition 3.5.1. can be seen to apply to one-step decodable codes but we can generalize the definition as below.

Definition 3.8.1.[6]

A set of parity-check sums $s_1, s_2, \ldots, s_J$ is said to be orthogonal on a set of error digits E if and only if every error digit in E is checked by every s_i and no other error digit is checked by more than one sum.

Thus the set E can be correctly decoded in the presence of $t \leqslant J/2$ errors. If we can obtain $E_1, E_2, \ldots, E_J$, such that they are orthogonal on another set F, then by using two levels of majority-logic, F can be decoded with $t \leqslant J/2$ errors. If by using L-1, levels of majority-logic we can decode a set $N_1, N_2, \ldots, N_J$, which are orthogonal on a single error digit, e_m, then with an L'th level of majority-logic, we can decode, e_m, in the presence of $t \leqslant J/2$ errors. With a cyclic code this is all that is necessary to decode all the received digits and the code is said to be "L-step majority-logic decodable".

Note that we require J orthogonal check sums on all E_i, $F_i, \ldots, N_i$, e_m.



m = Majority-Logic Date.

G = Gate.

+ = Modulo 2 adder.

FIG. 3.8.1.

A GENERAL TYPE I L-STEP DECODER.

In addition if $J = d_m - 1$, the code is said to be "completely orthogonalizable in L-steps".

Type 1. or Type 2. decoding can be used for L-step codes. Figure 3.8.1. shows a general Type I, L-step decoder and can be seen to be the same as that in Figure 3.6.1. except the "error estimator" has been replaced by L levels of majority-logic gates.

A general Type II, L-step decoder can be likewise obtained by replacing the "error estimator" in Figure 3.6.2. by L levels of majority-logic gates.

Example 3.8.1.

Consider the quasi-cyclic code of example 3.7.1. for which the following syndrome equations are obtained using Type 1. decoding.

$$s_o = e_o \oplus e_9 \oplus e_{10} \oplus e_{11}$$

$$s_1 = e_1 \oplus e_6 \oplus e_{10} \oplus e_{11}$$

$$s_2 = e_2 \oplus e_6 \oplus e_7 \oplus e_{11}$$

$$s_3 = e_3 \oplus e_6 \oplus e_7 \oplus e_8$$

$$s_4 = e_4 \oplus e_7 \oplus e_8 \oplus e_9$$

$$s_5 = e_5 \oplus e_8 \oplus e_9 \oplus e_{10}$$

Orthogonal on $e_{10} \oplus e_{11}$, we have,

$$E_1 = s_1 = e_1 \oplus e_6 \oplus e_{10} \oplus e_{11}$$

$$E_2 = s_o = e_o \oplus e_9 \oplus e_{10} \oplus e_{11}$$

$$E_3 = s_2 \oplus s_3 \oplus s_4 \oplus s_5$$

$$= e_2 \oplus e_3 \oplus e_4 \oplus e_5 \oplus e_7 \oplus e_8 \oplus e_{10} \oplus e_{11}$$

And $F_1 = e_{10} \oplus e_{11}$

Orthogonal on $e_6 \oplus e_{11}$, we have

$$E_1 = s_1 = e_1 \oplus e_{10} \oplus e_6 \oplus e_{11}$$

$$E_2 = s_2 = e_2 \oplus e_7 \oplus e_6 \oplus e_{11}$$

$$E_3 = s_o \oplus s_3 \oplus s_4 \oplus s_5$$

$$= e_o \oplus e_3 \oplus e_4 \oplus e_5 \oplus e_8 \oplus e_9 \oplus e_6 \oplus e_{11}$$

And $F_2 = e_6 \oplus e_{11}$.

Orthogonal on $e_8 \oplus e_{11}$, we have

$$E_1 = s_2 \oplus s_3 = e_2 \oplus e_3 \oplus e_8 \oplus e_{11}$$

$$E_2 = s_o \oplus s_5 = e_o \oplus e_5 \oplus e_8 \oplus e_{11}$$

$$E_3 = s_1 \oplus s_4 = e_1 \oplus e_4 \oplus e_6 \oplus e_7 \oplus e_9 \oplus e_{10} \oplus e_8 \oplus e_{11}$$

And $F_3 = e_8 \oplus e_{11}$.

Since F_1, F_2, F_3 are orthogonal on e_{11}, this can be decoded with $t = 1$ error, by having a second level of majority-logic.

After a cyclic shift of the syndrome we obtain, after cancelling e_{11},

$$F_1 = e_9 \oplus e_{10}$$

$$F_2 = e_{10}$$

$$F_3 = e_7 \oplus e_{10}$$

After decoding and cancelling e_{10}, shifting the syndrome register one digit, gives,

$$F_1 = e_8 \oplus e_9$$

$$F_2 = e_9$$

$$F_3 = e_6 \oplus e_9 \ .$$

Continuing the process we obtain,

$$F_1 = e_7 \oplus e_8$$

$$F_2 = e_8$$

$$F_3 = e_8$$

and decode e_8.

$$F_1 = e_6 \oplus e_7$$

$$F_2 = e_7$$

$$F_3 = e_7$$

and decode e_7.

$$F_1 = e_6$$

$$F_2 = e_6$$

$$F_3 = e_6.$$

So that the errors in the $k = 6$, message-digits can be decoded cyclically in 2-steps. If the process is repeated, providing all errors, e_{11}, e_{10}, e_9, e_8, e_7, e_6, have been correctly decoded and cancelled, we should obtain six zero outputs from the last level of majority-logic. If at least one non-zero output is obtained, we must assume an uncorrectable error pattern of weight $> t$ has occurred.

Peterson and Weldon[2] have shown that for a given code one can hope to correct roughly twice as many errors with L-step decoding as one can with one-step decoding. In particular if the minimum distance of the null space of an (n,k) code is $\bar{d}_m$ and t_1 and t_L are the number of errors one can correct using 1-step and L-step decoding respectively, then Peterson and Weldon[2] showed,

$$t_1 \leqslant \left[\frac{n - 1}{2(\bar{d}_m - 1)}\right] \qquad 3.8.1.$$

$$t_L \leqslant \left[\frac{n}{\bar{d}_m} - \frac{1}{2}\right] \qquad \bar{d}_m \text{ even}$$

$$\leqslant \left[\frac{n + 1}{\bar{d}_m + 1} - \frac{1}{2}\right] \qquad \bar{d}_m \text{ odd.} \qquad 3.8.2.$$

This needs a little explaining. If we can form 2t check sums orthogonal on a set of B error digits, then if the check sums are n-tuples, in the null space of the code, with minimum distance $\bar{d}_m$, each check sum must contain at least $\bar{d}_m$ - B digits, not contained in another check sum. Since there are only n - B digits to choose from, the maximum number of check sums, J, is given by

$$J = \frac{n - B}{\bar{d}_m - B} \qquad 3.8.3.$$

So the maximum number of correctable errors is given by,

$$t = \left[\frac{n - B}{2(\bar{d}_m - B)}\right] \qquad 3.8.4.$$

Equations 3.8.3. and 3.8.4. assume every check sum has the same weight and that this weight is $\bar{d}_m$. Let B = 1 and we have 1-step decoding giving,

$$t_1 = \left[\frac{n - 1}{2(\bar{d}_m - 1)}\right] \qquad 3.8.5.$$

So that equation 3.8.1. has equality when all check sums have weight $\bar{d}_m$. If any check sum has weight > $\bar{d}_m$ then equation 3.8.1. holds. Equation 3.8.5. is given by the minimum value of B, but B has a maximum value of $\bar{d}_m/2$ if $\bar{d}_m$ is even and $(\bar{d}_m - 1)/2$ if $\bar{d}_m$ is odd.

This must be since, if $B > \overline{d}_m/2$ or $> (\overline{d}_m - 1)/2$ in the respective cases, then the sum of two check sums would have weight $< \overline{d}_m$, which is not possible since they are code words in the null space. Putting $B = \overline{d}_m/2$ in equation 3.8.4. gives,

$$t_L = \left[\frac{n}{\overline{d}_m} - \frac{1}{2}\right] \qquad 3.8.6.$$

Again the equality holds since we are assuming all check sums have weight $\overline{d}_m$. If $B = (\overline{d}_m - 1)/2$

$$t_L = \left[\frac{n + 1}{(\overline{d}_m + 1)} - \frac{1}{2}\right] \qquad 3.8.7.$$

As given by equality in equation 3.8.2., again due to all check sums having the same weight of $\overline{d}_m$. Again, in equation 3.8.6. and 3.8.7., if any check sum has weight $> \overline{d}_m$ then equations 3.8.2. hold.

These bounds ignore the structure of the code so that it is not always possible to obtain the ideal conditions of check sums of the same weight $\overline{d}_m$. Even when a code is completely orthogonalizable it is not necessarily optimum in the sense of equations 3.8.5, 3.8.6. and 3.8.7.

The case of the quasi-cyclic code in examples 3.8.1. and 3.7.1. is obviously sub-optimum since $t_1 = t_L$, also the check sums do not all have the same weight. The idea of obtaining J check sums orthogonal on a set of error digits B, such that each check sum is assigned equal priority in the decoding scheme is referred to as the Reed-Massey[4] algorithm.

3.9 The Least Average Probability of Error.

If we assume the code words of a majority-logic code are transmitted over the binary symmetric channel with additive independent noise, then is it possible to find a decoding rule which gives the least probability of error?

Consider the two check sums below, from some code,

$$s_1 = e_n \oplus e_{n-t+1}$$
$$s_2 = e_n \oplus e_1 \oplus e_2 \oplus e_3 \oplus \ldots\ldots\ldots \oplus e_{n-t} \qquad 3.9.1.$$

S_1 will be in error only if e_{n-t+1} is in error, while s_2 will be in error if any odd number of the set $\{e_1, e_2, e_3, \ldots, e_{n-t}\}$ are in error. With additive independent noise the probability that any digit e_i will be binary one is the same for any $1 \leq i \leq n$. Thus the probability that s_2 will be in error is greater than the probability that s_1 will be in error, for a given error pattern. Therefore it seems obvious that we can reduce the probability of a decoding error by taking account of these check sum probabilities. Massey[4] approached this problem and developed the following decoding rule, which gives the least average probability of a decoding error.

Rule for decoding.

Given a set of check sums, A_i, orthogonal on error digit e_m, then if $p_i = 1 - q_i$ is the probability of an odd number of binary ones among the error digits in check sum A_i, excluding e_m, then we choose $e_m = 1$ if and only if

$$\sum_{i=1}^{J} A_i \; 2 \log (q_i/p_i) > \sum_{i=0}^{J} \log (q_i/p_i) \qquad 3.9.2.$$

Massey[4] called this "a posteriori probability" (A.P.P.) decoding and although given in the context of one-step decoding, is also valid for L-step decoding at each majority-logic gate on each level.

Let us assume that the probability of a single digit being in error is

$$p(e_i = 1) = p_o = 1 - q_o \qquad 3.9.3.$$

and let us assume that a check sum A_i has t_i digits excluding some set of digits B. Then we have,

$$p_i = \sum_x \frac{t_i!}{x!\,(t_i - x)!}\, p_o^{\,x}\, q_o^{\,n-x} \qquad 3.9.4.$$

where x is all odd numbers. However it is obvious that if $t_i = t_j$ then $p_i = p_j$ and in particular if t_i is constant for all check sums, or equally, all check sums have the same weight,

$$\log\,(q_i/p_i) = \log\,(q/p) = \text{constant}$$

and from 3.9.2. the decoding rule becomes,

$$\sum_{i=1}^{J} A_i > \frac{1}{2}\sum_{i=0}^{J} 1 \qquad 3.9.5.$$

$$> \frac{J}{2}\,.$$

Thus the one-step and L-step decoding rules of sections 3.7. and 3.8. minimise the average probability of error if and only if all check sums have the same weight. This is the case for equations 3.7.1. but not for the quasi-cyclic code of examples 3.7.1. and 3.8.1.

We can now define an optimum majority-logic decodable code, using the Reed-Massey[4] algorithm, where the code has optimum error-correcting capability t and minimum average probability of error, in the following way.

<u>Definition 3.9.1.</u>

A binary M.L.D. code is optimum if all check sums A_i have equal weight, $\overline{d}_m$, such that they are orthogonal on a set of error digits B, where

$$B = \frac{\overline{d}_m}{2}, \quad \text{if } d_m \text{ is even,}$$
$$= \frac{\overline{d}_m - 1}{2}, \quad \text{if } d_m \text{ is odd,}$$

where $\overline{d}_m$ is the minimum distance of the null space code.

3.10 Non-orthogonal Check-sums.

This algorithm was first introduced by Rudolph[9] and is an alternative to the Reed-Massey[4] algorithm. It is basically a one-step majority-logic decoding algorithm using non-orthogonal check sums.

Consider the check sums s_1, s_2, s_3, s_4, below.

$$\begin{aligned} s_1 &= e_1 \oplus e_3 \oplus e_4 \oplus e_m \\ s_2 &= e_1 \oplus e_2 \oplus e_5 \oplus e_m \\ s_3 &= e_2 \oplus e_3 \oplus e_6 \oplus e_m \\ s_4 &= e_4 \oplus e_5 \oplus e_6 \oplus e_m \end{aligned} \qquad 3.10.1.$$

They are uniform on e_m but every other digit appears twice so that they are not orthogonal as specified in definition 3.6.1.

However if we assume a single error has occurred,

(a) if e_m is in error a majority vote of the s_i will correctly decode e_m.

(b) if some $e_i \neq e_m$ is in error it can only affect at most two check sums and since this represents a split vote e_m will be correctly decoded as zero, using a conventional 4-input majority gate.

Providing each of the error digits, other than the digit to be decoded, appear in λ of the check sums then we can decode in the presence of t errors, where,

$$t = \left[\frac{N}{2\lambda} \right] \qquad 3.10.2.$$

and N = the number of check sums. So that we require N = 2tλ check sums and therefore a 2tλ- input majority-logic gate.

The largest classes of codes for which the algorithm can be used are the Euclidean and Projective geometry codes, see Peterson and Weldon[2]. Here it is shown that for both classes of binary codes,

$$N = \lambda \frac{(2^{sm} - 1)}{(2^{sr} - 1)}$$

$$t = \left[\frac{1}{2} \frac{(2^{sm} - 1)}{(2^{sr} - 1)} \right] \qquad 3.10.3.$$

However both classes of codes can also be decoded using L-step decoding and for long codes L-step decoding corrects more errors.

Nevertheless by decoding non-orthogonally in two steps we can correct the same number of errors as L-step so that it becomes a matter of decoder complexity to decide which algorithm to use.

Type I or Type II decoding can be used with this algorithm also.

3.11 Pseudstep Orthogonalization.

In an attempt to increase the range of application of majority-logic decoding Duc[19] proposed a decoding algorithm which quite simply stated that it is possible to decode using a combination of orthogonal and non-orthogonal check sums on a set of B digits.

The set below, as an example, will majority decode e_m in the presence of t = 2 errors.

$$\begin{aligned}
&e_m \oplus e_1 \\
&e_m \oplus e_2 \\
&e_m \oplus e_3 \oplus e_4 \\
&e_m \oplus e_4 \oplus e_5 \\
&e_m \oplus e_3 \oplus e_5
\end{aligned} \qquad 3.11.1.$$

The algorithm has been successful in the majority-logic decoding of codes previously thought not to be majority-logic decodable. This has been the algorithms primary use so far.

This algorithm is not to be confused with the practice, used for the decoding of some L-step decodable codes, of using different algorithms at different decoding levels.

3.12 Generalized Threshold Decoding.

A number of variations on the majority-logic decoding algorithms so far presented exist and are outlined below.

(a) Weighted Majority Decoding.

In each of the majority-logic algorithms previously presented, having obtained a set of check sums, each is given equal priority in the decoding scheme. We could say each is given a vote of one. In Rudolphs[9] paper, the non-orthogonal check sums were obtained on received digits directly and he showed that we could obtain $(2t\lambda + 1)$ check sums by utilizing an identity check sum on the digit to be decoded. In terms of check sums on error digits, this identity check sum corresponds to the zero check sum and Ng[7] showed that the algorithm could be improved by allowing the (identity) zero check sum λ votes instead of only one.

For example consider the set of weighted check sums below.

$$\begin{array}{lll} s_o = 0 & \text{2 votes} & \\ s_1 = e_1 \oplus e_2 \oplus e_m & \text{1 vote} & \\ s_2 = e_2 \oplus e_3 \oplus e_m & \text{1 vote} & 3.12.1. \\ s_3 = e_1 \oplus e_3 \oplus e_m & \text{1 vote} & \end{array}$$

For the set $s_{1,2,3,}$ we have $N = 3$, $\lambda = 2$, giving, from equation 3.10.2. $t = \left[3/4 \right] = 0$. But with s_o having λ votes, we have,

i) e_m in error gives 3 to 2 vote in its favour

ii) e_i, $i \neq m$, in error gives 3 to 2 vote in favour of $e_m = 0$, with s_o's votes.

So that now $t = 1$.

Giving s_o, λ votes, does not always improve the error-correcting capability. Rudolph[18,8] followed up this idea and developed one-step weighted majority decoding, whereby, check sums other than the zero check sum, are allowed more than one vote. He also showed that in principle any decoding function for any code can be realised by properly weighting the votes of generalized parity-check equations.

(b) Variable Threshold Decoding.[28]

Rather than decode with a majority-logic gate which has a fixed threshold we set the threshold at $(d_m - 1)$, initially, and attempt to decode each bit of a received word. At this initial stage error correction is effected only if all $d_m - 1$ inputs agree. If attempts to decode all n received bits are unsuccessful, the threshold is lowered by one and the process is repeated. Except for the initial stage, if any change (or correction) is made during an attempt to decode with a given threshold, the syndrome is reset and the threshold increased by one. Nevertheless after this if all n bits are not decoded successfully the threshold is again lowered by one. This continues until the threshold reaches the familiar figure of $(d_m - 1)/2$ when the received word is deemed to be decoded. Although many error patterns of weight > t can be corrected, this must be traded against decoder complexity and time to decode.

CHAPTER 4

4. A CLASS OF BINARY CODES.

4.1 Introduction.

The class of binary block codes presented in this chapter are not cyclic or quasi-cyclic in the sense of the strict definitions given in Chapter 3. Whereas in a quasi-cyclic code the generator matrix, G, is composed of k-tuple circulants of the same order, in the codes presented in this chapter, this is relaxed to allow G to be composed of (k-1)-tuple circulants of differing orders, with an overall parity-check, on every column of G, on the k'th digit. We therefore design a given code by a choice of (k-1)-tuple circulants and provided this permits the decoding of these (k-1) digits in the presence of t errors, we can decode the k'th digit by special provision.

The codes are majority-logic decodable in one-step and are cyclically decodable using a form of Type II decoding. The cyclic decoding procedure only decodes the message digits, whose decoded estimates are presented at the majority-logic gate output.

We also show that the codes are completely orthogonalizable up to their minimum distance. Initially it is shown how to obtain check sums using circulants and then the existence of circulants of various orders is examined. It is shown that this can be done by considering the successive doubling of positive integers modulo $(2^{k'}-1)$, where $k' = k-1$.

An initial code construction is proposed and then extended until the most optimum code is obtained. This is done with the help of a comprehensive example for $k'=6$.

A simple encoding and decoding scheme is proposed.

4.2 Orthogonal Check Sums from Circulants.

The generator matrix, G, of a binary code of length n, is composed of n columns of binary k-tuples, where k is the number of information-digits in the code.

Let T(x) be a polynomial representing a code word of the code, then

$$T(x) = c_o + c_1x + \ldots\ldots + c_{n-1}x^{n-1}$$

where;

$$c_i = c_{i_1} m_1 \oplus c_{i_2} m_2 \oplus \ldots\ldots \oplus c_{i_k} m_k \qquad 4.2.1.$$

m_i = i'th digit of information, $\in GF(2)$.

c_{i_j} = j'th digit of the i'th column of G, $\in GF(2)$.

$\oplus$ = addition over GF(2).

Therefore each c_i is a linear equation in the message-digits, determined by the i'th column of the generator matrix, such that $c_i \in GF(2)$. The coefficients of T(x) are therefore binary and it is the binary coefficients that the encoder transmits and the decoder receives in the presence of noise. The noise which affects a code word of length n, can be represented by a polynomial E(x), with binary coefficients, such that,

$$E(x) = e_o + e_1x + \ldots\ldots + e_{n-1}x^{n-1}.$$

If an error has occurred in the p'th digit $e_{p-1} = 1$, otherwise $e_{p-1} = 0$.

Let R(x) be the polynomial representing the received code word, then

$$R(x) = T(x) + E(x)$$
$$= r_o + r_1x + \ldots\ldots + r_{n-1}x^{n-1}$$

where

$$r_i = c_i \oplus e_i \qquad 4.2.2.$$

In equation 3.7.5. we saw that if we wish to correctly decode a message-digit, m_j, in the presence of errors, by obtaining orthogonal check-sums on m_j itself, we require at least $J = 2t+1$ check sums, where t is the number of errors or coefficients of $E(x)$ that are binary 1.

Let $c(i)$ represent the binary k-tuple in the i'th column of G and let $c(j)$ differ from $c(i)$ only in the first digit, that is

$$c(i) = (c_{i_1}, c_{i_2}, \ldots, c_{i_k}), \; c(j) = (c_{j_1}, c_{j_2}, \ldots, c_{j_k})$$

and

$$c_{i_k} = c_{j_k} \quad \text{for} \quad k > 1$$

$$c_{i_1} \neq c_{j_1} .$$

The two transmitted digits c_i and c_j are determined by these two columns i and j of G, so that at the receiver the two received digits will be r_i and r_j where,

$$r_i = c_i \oplus e_i$$

$$r_j = c_j \oplus e_j .$$

However it is obvious that

$$r_i \oplus r_j = c_i \oplus c_j \oplus e_i \oplus e_j$$

and since $c_{i_k} \oplus c_{j_k} = 0$ but

$$c_{i_1} \oplus c_{j_1} = 1 \text{ then}$$

$$c_i \oplus c_j = (c_{i_1} \oplus c_{j_1})\, m_1 = m_1, \text{ and}$$

$$r_i \oplus r_j = m_1 \oplus e_j \oplus e_i \qquad 4.2.3.$$

and we obtain an orthogonal check sum on m_1. The same two received digits cannot be used to obtain another check sum on m_1 because the errors associated with these received digits must remain unique to this check sum, with respect to m_1. If $J = 2t+1$ such pairs of columns are contained in the generator matrix, for every message-digit, then the code is majority-logic decodable in the presence of $\leq t$ errors.

Based on the above we can give a set of simple, and rather sub-optimum, codes in the following theorem.

Theorem 4.2.1.

If two complete sets of k-tuples, one of weight b and one of weight b+1, are used as columns of the generator matrix of a binary code, then we can form J orthogonal check sums on every message-digit, where,

$$J = \frac{(k-1)!}{b!(k-1-b)!}$$

Proof:

For each k-tuple of weight b there is a k-tuple of weight b+1 that differs only in the i'th digit. From equation 4.2.1. we know that the sum of the received digits generated by such a pair of k-tuples gives an orthogonal check sum on m_i.

The number of such pairs on digit i is equal to the number of k-tuples from the set of weight (b+1) whose i'th digit is binary one or the number of k-tuples in the set of weight b whose i'th digit is binary 0.

In both cases this number, J, is the number of (k-1) tuples of weight b, therefore,

$$J = \frac{(k-1)!}{b!(k-1-b)!}$$

Q.E.D.

Providing a full set of k-tuples of weight (b+1) is present in G, one only need guarantee that there are m zero's in each column of the set of weight b, to obtain m check sums. One way of guaranteeing this is to use circulant sets of weight b.

Theorem 4.2.2.

If one can form a circulant set of order e, from a k-tuple of weight b, then the number of zero's in each column of the circulant set is given by,

$$J = \frac{e(k-b)}{k} .$$

Proof:

If e = k, the columns are also cyclic versions of the generating k-tuple and J = (k-b).

If e < k, then digit i = digit i+e so that the generating k-tuple must be composed of k/e repeated e-digit sections. The i'th column of the circulant set will be the digits i, i+1,......, i+e-1, from the generating k-tuple. But this is precisely an e-digit repeating section so that the number of zero's per column is the number of zero's in a repeating section, giving

$$J = \frac{(k-b)}{k/e} .$$

Q.E.D.

So that we can shorten the codes of theorem 4.2.1. by using circulants of weight b.

The following example illustrates the theorems 4.2.1. and 4.2.2.

Example 4.2.1.

If we use the two sets of 5-tuples of Hamming weight 4 and 3, below,

b+1 = 4	b = 3
01111	00111
11110	01110
11101	11100
11011	11001
10111	10011
	01011
	10110
	01101
	11010
	10101

as columns of the generator matrix, we have b+1 = 4 and b = 3 so that from theorem 4.2.1. we can form,

$$J = \frac{(5-1)!}{3!(5-1-3)!} = 4$$

orthogonal check sums on all 5 digits by adding pairs of k-tuples. For example, if the rightmost column of all the k-tuples, represents message-digit m(1) we have,

$$(01111) \oplus (01110) = 00001 = m(1)$$

$$(11101) \oplus (11100) = 00001 = m(1)$$

$$(11011) \oplus (11010) = 00001 = m(1)$$

$$(10111) \oplus (10110) = 00001 = m(1)$$

This would represent a binary code, of length n = 15, k = 5, J = 4, with each k-tuple being a column of G. The results obtained on m(1) can also be obtained on all m(2), m(3), m(4), m(5).

The set of k-tuples with b = 3, can be split into two circulants of order e = 5, as below.

00111	01011
01110	10110
11100	01101
11001	11010
10011	10101

From theorem 4.2.2. we have J = (5-3) = 2, zero's per column.

Using either of these, along with the set b+1 = 4, to form the columns of G, gives a code with, n = 10, k = 5, J = 2.

The usefulness of circulants leads us to an investigation of their existence. For example it would be helpful to know when and under what conditions circulants of a given order e exist, for various k.

This is answered in the following section.

4.3 The Existence of Circulants.

We wish to show that the existence of circulants can be examined from the theory of successive doubling of positive integers modulo (2^k-1). We begin by establishing results necessary to the general proof.

Theorem 4.3.1.

If B(m(x)) is the decimal equivalent of the binary number representation of the coefficients of the polynomial, m(x), over G.F.(2), then

(i) $B(x^p) = 2^p$

(ii) $B(x^p.(m(x))) = 2^p.B(m(x))$.

Proof:

Let the polynomial,

$$m(x) = c_o + c_1x + \dots + c_ix^i \quad , \quad \text{then}$$

$$B(m(x)) = c_o2^o + c_12^1 + \dots + c_i2^i$$

if $c_p = 1$, but $c_{j\neq p} = 0$

$$B(m(x)) = 2^p = B(x^p)$$

$$x^p.m(x) = c_ox^p + c_1x^{p+1} + \dots + c_ix^{p+i}$$

$$B(x^p.m(x)) = c_o2^p + c_12^{p+1} + \dots + c_i2^{p+i}$$

$$= 2^p.(c_o + c_12 + \dots + c_i2^i)$$

$$= 2^p.B(m(x)).$$

Q.E.D.

We can also restate this in equation form as,

$$B(x^p.m(x)) = B(x^p).B(m(x)). \qquad 4.3.1.$$

Lemma. 4.3.1.

If $B(m(x))$ is odd, and $B(m(x)) = t$, then $B(m(x) \oplus 1) = t - 1$.

Proof:

If $B(m(x))$ is odd and

$$m(x) = c_o + c_1x + \dots . + c_ix^i \qquad c_i \in GF(2)$$

then $c_o \neq 0$, so that $c_o = 1$ and

$$1 + m(x) = c_1x + c_2x^2 + \dots . + c_ix^i$$

Q.E.D.

Lemma. 4.3.2.

If $B(m(x))$ is even, and $B(m(x)) = t$, then $B(m(x) \oplus 1) = t + 1$.

The proof follows from Lemma 4.3.1.

Lemma. 4.3.3.

If $m(x)$ has degree $(k-1)$, and $B(m(x)) = t$, then $B(m(x) \oplus x^k) = t + 2^k$.

Proof:

Since $m(x)$ has degree $(k-1)$,

$$B(m(x)) = c_o2^o + c_12^1 + \dots . + c_{k-1}2^{k-1}, \text{ so that}$$

$$B(m(x) + x^k) = c_o2^o + c_12^1 + \dots . + c_{k-1}2^{k-1} + 2^k$$

$$= t + 2^k .$$

Q.E.D.

Lemma 4.3.4.

If $m(x)$ has degree k, and $B(m(x)) = t$, then $B(m(x) \oplus x^k) = t - 2^k$.

The proof follows in a similar manner to Lemma 4.3.4.

The above lemma's enable us to prove the following theorem.

Theorem 4.3.2.

If $B(m(x))$ is the decimal equivalent of the binary number representation of the coefficients, $\in$ G.F.(2), of $m(x)$, then if

$$\text{degree } (m(x)) < k$$

and $0 < B(m(x)) < 2^k - 1$, then

$$B(x.m(x) \bmod (x^k + 1)) \equiv 2.B(m(x)) \bmod (2^k - 1).$$

Proof:

Let $x.m(x) \equiv r(x) \bmod (x^k + 1)$, then

$$x.m(x) = q(x)(x^k + 1) + r(x) \qquad \deg(r(x)) < k.$$

There are two cases to consider:-

a) if $\deg(m(x)) < k-1$ then $\deg(x.m(x)) < k$ therefore $q(x) = 0$ and $r(x) = x.m(x)$ giving

$$B(r(x)) = B(x.m(x)) < 2^k - 1$$

and from theorem 4.3.1., $B(x.m(x)) = 2.B(m(x))$

thus $B(r(x)) \equiv 2.B(m(x)) \bmod (2^k-1)$.

b) if $\deg(m(x)) = k - 1$, then $\deg(x.m(x)) = k$ this implies $\deg(q(x)) = 0$ and so $q(x) = 1$ giving

$$x.m(x) = x^k + 1 + r(x)$$

from theorem 4.3.1., let

$$B(x.m(x)) = 2.t \qquad t < 2^k - 1,$$

then from Lemma 4.3.2., since 2t is even.

$$B(x.m(x) + 1) = 2t + 1,$$

and from Lemma 4.3.4. since $\deg(x.m(x) + 1) = k$

$$B(x.m(x) + 1 + x^k) = 2t + 1 - 2^k,$$

therefore $B(r(x)) = 2t + 1 - 2^k$

$2t = 2^k - 1 + B(r(x))$

and $2t \equiv B(r(x)) \bmod(2^k-1)$

or $B(r(x)) \equiv 2.B(m(x)) \bmod(2^k-1)$

Q.E.D.

Since $\deg(r(x)) < k$ it follows that,

$$B(x.r(x) \bmod(x^k+1)) \equiv 2(2.B(m(x)) \bmod(2^k-1) \qquad 4.3.2.$$

and it follows by induction on theorem 4.3.2.

$$B(x^p.m(x) \bmod(x^k+1)) \equiv 2^p B(m(x)) \bmod(2^k-1). \qquad 4.3.3.$$

In theorem 4.3.2. above it was stipulated that $0 < B(m(x) < 2^k-1$ and this is because if

$B(m(x)) = 2^k-1$, then

$x.m(x) \equiv m(x) \bmod(x^k+1)$, and if

$B(m(x)) = 0$, again

$x.m(x) \equiv m(x) \bmod(x^k+1)$.

The circulants generated by the numbers 0 and 2^k-1, are trivial circulants of order 1 which exist for all values of k.

The following example illustrates the principles above.

Example 4.3.1.

Let $m(x) = 1 + x + x^2$, be a polynomial modulo (x^5+1), with coefficients $\in$ G.F(2), then we have $B(m(x)) = 7$.

$x.m(x) \equiv x + x^2 + x^3 \bmod(x^5+1)$

$x^2.m(x) \equiv x^2 + x^3 + x^4 \bmod(x^5+1)$

$x^3.m(x) \equiv 1 + x^3 + x^4 \bmod(x^5+1)$

$x^4.m(x) \equiv 1 + x + x^4 \bmod(x^5+1)$

$x^5.m(x) \equiv 1 + x + x^2 \bmod(x^5+1)$

$$B(x + x^2 + x^3) = 14 \equiv 2.7 \bmod (31)$$
$$B(x^2 + x^3 + x^4) = 28 \equiv 2^2.7 \bmod (31)$$
$$B(1 + x^3 + x^4) = 25 \equiv 2^3.7 \bmod (31)$$
$$B(1 + x + x^4) = 19 \equiv 2^4.7 \bmod (31)$$
$$B(1 + x + x^2) = 7 \equiv 2^5.7 \bmod (31)$$

the above numbers put in binary form give the following circulant set,

00111	(7)
01110	(14)
11100	(28)
11001	(25)
10011	(19) .

In the light of theorem 4.3.2. we can now determine some of the properties of circulants of k-tuples by examining the properties of integers, $0 < a < 2^k-1$, $\bmod(2^k-1)$, when they are successively doubled.

In particular if $B(m(x)) = a$, $0 < a < 2^k-1$ then the order of the circulant generated by the k-tuple, $m(x)$, can be determined by solving the congruence,

$$2^p a \equiv a \bmod(2^k-1) \qquad 4.3.4.$$

whereby, if p is the smallest integer satisfying the congruence, then the circulant has order p.

We begin by showing what is intuitively obvious, that is, $2^k a \equiv a \bmod(2^k-1)$, for all $a < 2^k-1$.

Theorem 4.3.3.

The congruence,

$$2^k a \equiv a \bmod(2^k-1)$$

is true for all $a \leqslant 2^k-1$.

Proof:

Since $a2^k - a = a(2^k-1)$,

$$2^k a = a(2^k-1) + a, \quad \text{and if } a < 2^k-1$$

$$2^k a \equiv a \bmod (2^k-1)$$

Q.E.D.

As an immediate consequence of theorem 4.3.3. we have the following corollary.

Corollary 4.3.1.

If p is the smallest positive integer such that,

$$2^p a \equiv a \bmod (2^k-1) \ , \quad a < 2^k-1$$

then $p \leqslant k$.

With the help of the following theorem we can develop conditions for which $p = k$ is the only solution.

Theorem 4.3.4.[41,42]

Let $(c,m) = d$, and write $m = m_1 d$ and $c = c_1 d$, then if

$$ca \equiv cb \bmod (m), \text{ then}$$

$$a \equiv b \bmod (m_1).$$

As a further corollary, we have,

Corollary 4.3.2.

If $(c,m) = 1$, and

$$ca \equiv cb \bmod (m), \quad \text{then}$$

$$a \equiv b \bmod (m).$$

Therefore, from equation 4.3.4. if $(a, 2^k-1) = 1$, then from corollary 4.3.2.

$$2^p \equiv 1 \bmod (2^k-1)$$

and the only solution is, $p = k$. Note that this occurs whenever 2^k-1 is prime. We can restate this result in another way with the following theorem.

Theorem 4.3.5.

If p is the smallest positive integer such that,

$$2^p a \equiv a \bmod(2^k-1), \qquad a < 2^k-1$$

then if $p < k$,

$$(a,\ 2^k-1) \neq 1.$$

We can obtain a useful result regarding the values p can assume, when a and 2^k-1 are not relatively prime, with the help of the following theorem.

Theorem 4.3.6.[41,42]

If e is the smallest positive integer such that

$$a^e \equiv 1 \bmod(m), \text{ then}$$

$$a^k \equiv 1 \bmod(m)$$

if and only if $e|k$.

Let us assume that for some a, a circulant is generated of order $p < k$. Then from theorem 4.3.5. we have $(a,\ 2^k-1) \neq 1$. Assume, $(a,\ 2^k-1) = d$, and write

$$2^k-1 = d.b \quad , \quad a = a'.d$$

then from theorem 4.3.4.

$$2^p \equiv 1 \bmod(b) \qquad 4.3.5.$$

However from theorem 4.3.3.

$$2^k a \equiv a \bmod(2^k-1)$$

and since $2^k-1 = db$ and $a = a'd$

from theorem 4.3.4. again,

$$2^k \equiv 1 \bmod(b) \qquad 4.3.6.$$

Therefore from theorem 4.3.6. the simultaneous congruences 4.3.5. and 4.3.6. can only be so if $p|k$. We have proved the following theorem.

Theorem 4.3.7.

If p is the smallest positive integer such that,

$$2^p a \equiv a \bmod(2^k-1), \quad a < 2^k-1$$

then $p < k$, if and only if, $p|k$.

Therefore only circulants of orders which divide k are possible.

We also have the following corollary.

Corollary 4.3.3.

If p is the smallest positive integer such that,

$$2^p a \equiv a \bmod(2^k-1), \quad a < 2^k-1$$

then if k is a prime, $p = k$.

Since if k is prime it has no divisors except 1, the trivial circulant order, and itself k. We also have the following Corollary.

Corollary 4.3.4.

If p is the smallest positive integer such that,

$$2^p a \equiv a \bmod(2^k-1)$$

then if $p < k$, k is a composite integer with more than one factor.

Summarizing the results so far, we have;

(a) $p = k$, if

(i) $(a, 2^k-1) = 1$ 4.3.7.

(ii) $k =$ a prime

(b) if $p < k$,

(i) $(a, 2^k-1) \neq 1$

(ii) $k =$ a composite. 4.3.8.

(iii) $p|k$

There is one further useful result, which demonstrates a particular case when b(i) and (ii) of equations 4.3.8. are satisfied but $p = k$.

This occurs when $a = 2^n-1$, $n < k$. To show this requires some results to be derived for numbers of the form 2^k-1. Numbers of the form 2^k-1 have been the subject of considerable research among mathematicians and the following points are known.[43]

i) if 2^k-1 is prime, k must be prime.

ii) if k is a prime, 2^k-1 may be composite.

iii) if k is composite, 2^k-1 is always composite.

If k is prime 2^k-1 is called a Mersenne number. There is no known generalized method for finding the factors of composite Mersenne numbers and many such numbers have still not been successfully factored.

However to show that $p = k$ when $a = 2^n-1$ we begin with the following theorem.

<u>Theorem 4.3.8.</u>

If p and k are positive integers such that $(p,k) = 1$, then

$$(2^p-1,\ 2^k-1) = 1.$$

Proof:

Assume it is not true and

$$(2^p-1,\ 2^k-1) = t, \quad \text{then we have,}$$

$$2^p \equiv 1 \bmod(t)$$

$$2^k \equiv 1 \bmod(t)$$

there are two cases to consider.

(a) let $p < k$ and assume p is the smallest integer such that the congruence holds, then from theorem 4.3.6. $p|k$ and $(p,k) = p$. Similarly if $k < p$ and k is the smallest integer, $k|p$ and $(p,k) = k$.

(b) if neither p nor k are the smallest integers satisfying the congruence, then there exists some integer s, such that

$$s < p \text{ and } s < k \text{ and}$$

$$2^s \equiv 1 \bmod (t)$$

but then from theorem 4.3.6.

$$s|p \text{ and } s|k \text{ and } (p.k) = s.$$

If $(p,k) = 1$, both cases are impossible and so $(2^p-1,\ 2^k-1) = 1$.

Q.E.D.

Two corollaries follow.

<u>Corollary 4.3.5.</u>

If k is a positive prime, then for all $p < k$, $(2^p-1,\ 2^k-1) = 1$.

<u>Corollary 4.3.6.</u>

All Mersenne numbers are relatively prime to all other Mersenne numbers.

We require two more results before the general proof is presented.

<u>Theorem 4.3.9.</u>

If $e|k$, then $2^e-1|2^k-1$.

Proof:

Let $k = en$, then

$$2^{en}-1 = (2^e-1)((2^e)^{n-1} + (2^e)^{n-2} + \dots + 2^e + 1)$$

Q.E.D.

<u>Theorem 4.3.10.</u>

If n and k are positive integers such that $n < k$ and

$(2^n-1,\ 2^k-1) = d$, then

$d = 2^s-1$, where $s \leqslant n$.

Proof:

Since d is a common divisor

$$2^n \equiv 1 \bmod(d)$$
$$2^k \equiv 1 \bmod(d)$$

there are two cases to consider.

a) if $n|k$, then $2^n-1|2^k-1$ from theorem 4.3.9. and $d = 2^n-1$.

b) if $n\not|k$, there exists some integer s, such that $s|n$ and $s|k$ and

$$2^s \equiv 1 \bmod(d). \qquad 4.3.9.$$

However, since $s|n$ and k, from theorem 4.3.9. $2^s-1|2^n-1$ and $2^s-1|2^k-1$ therefore $2^s-1|d$. But $d|2^s-1$ from equation 4.3.9., therefore $d = 2^s-1$.

Q.E.D.

Corollary 4.3.7.

If n and k are positive integers such that, $n < k$ and $(n,k) = s$, then

(a) $(2^n-1,\ 2^k-1) = 2^s-1$, and

(b) $\left(\dfrac{2^n-1}{2^s-1}\ ,\ \dfrac{2^k-1}{2^s-1}\right) = 1$

Proof:

If $(n,k) = s$, then 2^s-1 is the greatest integer of this form that divides 2^n-1 and 2^k-1. Since the greatest common divisor must be of the same form, from theorem 4.3.10, then 2^s-1 is the greatest common divisor of 2^n-1 and 2^k-1.

If there existed some $d > s$ such that 2^d-1 divided 2^n-1 and 2^k-1, then this implies $d|n$ and $d|k$ and $(n,k) = d$, which is not possible and (a) is proved.

It follows from the definition of greatest common divisor that (b) is proved once (a) is proved.

Q.E.D.

Returning to equation 4.3.4., if $a = 2^n-1$, $n < k$, we have,

$$2^p(2^n-1) \equiv (2^n-1) \bmod(2^k-1), \text{ or}$$

$$2^p(2^n-1) = q(2^k-1) + 2^n-1, \quad 2^n-1 < 2^k-1$$

where q is a positive integer. In terms of q we can write,

$$q = \frac{(2^p-1)(2^n-1)}{(2^k-1)} \qquad 4.3.10a$$

We can now show that if $n < k$, for q to be a positive integer, $p = k$ is the only value for p.

Theorem 4.3.11.

If q,p,n and k are positive integers such that $n < k$ and $p \leq k$, if

$$q = \frac{(2^p-1)(2^n-1)}{(2^k-1)}$$

then $p = k$ and $q = 2^n-1$.

Proof:

There are three cases to consider,

a) if $(n,k) = 1$, $(2^n-1, 2^k-1) = 1$ from theorem 4.3.8. therefore 2^k-1 must wholly divide 2^p-1 so that $p = k$.

b) if $(n,k) = n$, let $k = nb$ and from theorem 4.3.9. $2^n-1 | 2^k-1$ and q can be written,

$$q = \frac{2^p-1}{(2^n)^{b-1} + (2^n)^{b-2} + \ldots + 2^n+1}$$

but since $p|k$, let $k = pc = nb$ and $p = nb/c$, and therefore

$$q = \frac{2^{nb/c}-1}{(2^n)^{b-1} + (2^n)^{b-2} + \ldots + 2^n+1}$$

But if q is to be an integer, we require

$$\frac{nb}{c} > n(b-1)$$

or $$\frac{b}{c} > (b-1)$$

which can only be so if $c = 1$.

Thus, $k = pc = p$.

c) if $(n,k) = s$, $s|n$ and k, and from theorem 4.3.9. $2^s-1|2^n-1$ and 2^k-1. We can then write $n = f.s$, $k = ts$, then $(f,t) = 1$ and

$$q = \frac{(2^p-1)((2^s)^{f-1} + (2^s)^{f-2} + \dots + 2^s+1)}{((2^s)^{t-1} + (2^s)^{t-2} + \dots + 2^s+1)}$$

But from theorem 4.3.10 and corollary 4.3.7.

$$\left(\frac{2^n-1}{2^s-1} , \frac{2^k-1}{2^s-1}\right) = 1$$

so for q to be a positive integer, we require that $2^k-1/2^s-1$ wholly divides 2^p-1. Since $k = pc = ts$, $p = st/c$, and

$$q = \frac{(2^{st/c}-1)((2^s)^{f-1} + (2^s)^{f-2} + \dots + 2^s+1)}{((2^s)^{t-1} + (2^s)^{t-2} + \dots + 2^s+1)}$$

thus we require,

$$\frac{st}{c} > s(t-1)$$

$$\frac{t}{c} > (t-1)$$

which is only possible if $c = 1$.

Thus $k = pc = p$.

Q.E.D.

Therefore, if $p = k$,

i) $(a, 2^k-1) = 1$

or ii) k = prime

or iii) $a = 2^n-1, \quad n < k.$

And if $p < k$

i) $p|k$

ii) $2^p = q\,\dfrac{(2^k-1)}{a} + 1$

for some $q = 1,2,3,\ldots\ldots$.

In the following example we find all orders for all $a < 2^6-1$.

Example 4.3.2.

We find as an example, all circulants generated by integers $a < 2^6-1$. We begin with unity.

$\{2^o.1,\ 2^1.1,\ 2^2.1,\ 2^3.1,\ 2^4.1,\ 2^5.1\} = \{1,2,4,8,16,32\}$

since $2^6.1 \equiv 1 \bmod(2^6-1)$, $p = 6$ as expected since $(1,2^6-1) = 1$.

We proceed by taking the next lowest integer not contained in the generated set above, which is 3, giving

$\{3,6,12,24,48,33\}$ and $p = 6$, since $3 = (2^2-1)$.

The next lowest integer is 5.

$\{5,10,20,40,17,34\}$ and $p = 6$, since $(5,63) = 1$

$\{7,14,28,56,49,35\}$ and $p = 6$, since $7 = (2^3-1)$

$\{9,18,36\}$ and $p = 3$,

$\{11,22,44,25,50,37\}$ and $p = 6$, since $(11,63) = 1$

$\{13,26,52,41,19,38\}$ and $p = 6$, since $(13,63) = 1$

$\{15,30,60,57,51,39\}$ and $p = 6$, since $15 = (2^4-1)$

$\{21,42\}$ and $p = 2$,

$\{23,46,29,58,53,43\}$ and $p = 6$, since $(23,63) = 1$

$\{27,54,45\}$ and $p = 3$,

$\{31,62,61,59,55,47\}$ and $p = 6$, since $31 = (2^5-1)$

and $(31,63) = 1$.

Since a circulant consists of all cyclic shifted forms of the generating k-tuple, if we form the modulo 2 sum of two k-tuples from different circulants to obtain a check sum, the modulo 2 sum of the cylically shifted versions of the two k-tuples forms another check sum on another digit. Mathematically we can say, let $\{a\}$ be the set of integers generated by $2^p a \bmod (2^k-1)$ for $0 \leq p \leq k$, then if $m_1(x)$ and $m_2(x)$ sum to form a check sum,

$$B(m_1(x) + m_2(x)) \in \{1\} \tag{4.3.6.}$$

and

$$\begin{aligned} &B(x^p\, m_1(x) \bmod (x^k+1) + x^p\, m_2(x) \bmod (x^k+1)) \\ &\qquad \equiv 2^p\, B(m_1(x) + m_2(x)) \bmod (2^k-1) \in \{1\} \end{aligned}$$

and this applies generally for,

$$B\left(\sum_{i}^{m} m_i(x) \right) \in \{1\} \ , \tag{4.3.7.}$$

where $m_1(x), m_2(x), \ldots, m_m(x)$, sum to form a check sum.

4.4 Initial Code Construction.

By using the results of the previous sections we can build up codes, whose generator matrix is composed of circulant $k' = (k-1)$-tuples with an overall check on circulent columns. We know from theorem 4.2.2. that providing the generator matrix contains all $k' = (k-1)$-tuples of weight $(b+1)$, we can add circulants of order p, weight b, and increase the number of check sums by $J = p(k'-b)/k'$ with each additional circulant. Of course each time a circulant of order p is added, the code length, n, increases by p. Since all check sums are formed by adding pairs of received digits the overall parity-check cancels.

We can best consider the initial code construction by an example. If we use the circulants of example 4.3.2. then $k' = 6$, and we construct codes with k = 7 information digits. Table 4.4.1. below represents each circulant by its generating number, a, and gives its order p, binary weight b and the number of check sums it provides, assuming, for the moment, all (b+1), k'-tuple circulants are contained in the generator matrix. For completeness the two trivial generators are also given.

TABLE 4.4.1.

Generating Number a.	Order p	Weight b	Check-sums $J = p(k'-b)/k'$
63	1	6	0
31	6	5	1
27	3	4	1
15,23	6	4	2
21	2	3	1
7,11,13	6	3	3
9	3	2	2
3,5	6	2	4
1	6	1	5
0	1	0	1

Note:

Although the check sums indicated are on the first k' information digits, having decoded these, it is a simple matter to decode the k'th, as we shall see later.

A given code can therefore be represented by its generating numbers. We begin by assuming the generator matrix contains 63, that is, all the k'-tuples of weight 6. We then add our circulants and obtain the set of codes given below, in Table 4.4.2.

TABLE 4.4.2.

Code length n	Check-sums J	Generating numbers a
7	1	63,31.
13	3	63,31,15.
13	3	63,31,23.
10	2	63,31,27.
16	4	63,31,15,27.
16	4	63,31,23,27.
19	5	63,31,15,23.
22	6	63,31,15,23,27.
28	9	63,31,15,23,27,7.
28	9	63,31,15,23,27,11.
28	9	63,31,15,23,27,13.
24	7	63,31,15,23,27,21.
34	12	$\underbrace{63,31,15,23,27}$,7,11.
34	12	" 7,13.
34	12	" 11,13.
30	10	" 7,21.
30	10	" 11,21.
30	10	" 13,21.
36	13	" 7,11,21.
36	13	" 7,13,21.
36	13	" 11,13,21.
40	15	" 7,11,13.
42	16	$\underbrace{\text{" 7,11,13,21}}$.
45	18	" 9.
48	20	" 3.
48	20	" 5.
51	22	" 9,3.
51	22	" 9,5.
54	24	" 3,5.
57	26	" 3,5,9.
63	31	" 3,5,9,1.
64	32	" 3,5,9,1,0.

These conditions are called "initial" because they are sub-optimum. A slight improvement in the constructions above is possible in the following manner.

Consider the circulant generated by 27 of order p = 3. As a binary 6-tuple, 27 has the form 011011, of weight b = 4, so that it will combine with 6-tuples of weight b = 3 and b = 5 to form check sums. Let {27} be the set of 6-tuples generated by 27, then by replacing each single one in 011011, in turn, by a zero, we can see which weight 3, 6-tuples, 27 combines with.

$$011011 \Rightarrow \begin{array}{lcl} 001011 & = & 11 \in \{11\} \\ 010011 & = & 19 \in \{13\} \\ 011001 & = & 25 \in \{11\} \\ 011010 & = & 26 \in \{13\} \end{array}$$

Therefore any 6-tuple $\in$ {27} only combines with weight 3, 6-tuples $\in$ {11} and {13}. The circulant {27} provides one check sum, from theorem 4.2.2., when combined with the weight 5, 6-tuples so that in codes whose generator matrix contains {27} but does not contain {11} and {13}, if we remove {27} we will reduce J by one and n by three.

This is done in the following codes;

i) n = 28, J = 9, {63,31,15,23,27,7}

ii) n = 24, J = 7, {63,31,15,23,27,21}

iii) n = 30, J = 10, {63,31,15,23,27,21,7}

If we remove {27} we obtain

i)' n = 25, J = 8, {63,31,15,23,7}

ii)' n = 21, J = 6, {63,31,15,23,11}

iii)' n = 27, J = 9, {63,31,15,23,7,11}.

Since {27} combines with {11} and {13} we show this by writing,

$$\{27\} \xleftrightarrow{\quad c \quad} \{11\}, \{13\}.$$

Examining all generators we obtain:

$\{63\} \xleftrightarrow{c} \{31\}$ since {31} is all 6-tuples of weight 5 no alteration is applicable.

$\{31\} \xleftrightarrow{c} \{15\},\{23\},\{27\}$ no new code.

$\{15\} \xleftrightarrow{c} \{7\},\{11\},\{13\}$ remove {15} from codes containing {15} and {21}.

n = 18, J = 5 {63,31,23,27,21}

$\{23\} \xleftrightarrow{c} \{7\},\{11\},\{13\},\{21\}$ no new code.

$\{27\} \xleftrightarrow{c} \{11\},\{13\}$ see above example.

$\{7\} \xleftrightarrow{c} \{3\},\{5\}$ remove {7} from codes containing {7} and {9}

n = 39, J = 15 {63,31,15,23,27,11,13,21,9}

$\{11\} \xleftrightarrow{c} \{3\},\{5\},\{9\}$ no new code.

$\{13\} \xleftrightarrow{c} \{3\}\ \{5\}\ \{9\}$ no new code.

$\{21\} \longleftrightarrow \{5\}$ remove {21} from codes containing {21} and {3} and {9}

n = 46, J = 19 {63,31,15,23,27,7,11,13,3}

n = 43, J = 17 {" " " " " " " " 9}

n = 49, J = 21 {" " " " " " " " 3,9}

$\{3\} \xleftrightarrow{c} \{1\}$ no new code

$\{5\} \xleftrightarrow{c} \{1\}$ " " "

$\{9\} \xleftrightarrow{c} \{1\}$ " " "

$\{1\} \longleftrightarrow \{0\}$ no new code

The new codes are also sub-optimum since some improvement is still possible.

4.5 Utilization of Redundancy.

In the previous section we presented a set of codes whose generator matrix was composed of circulants of differing orders. The total number of check sums possible, by adding pairs of columns, (or similarly pairs of received digits) was equal to the sum of the number of zero's in the columns of all the circulants present. That is the number of zero's in a row of the generator matrix, except the row which is an overall parity-check. Because of this any k-tuple column not used to decode message-digit i, which is said to be redundant, must have the i'th and k'th digit as binary one. The codes in the previous section are sub-optimum because of the relatively high number of redundant k-tuple columns for each message-digit. To demonstrate the uses which can be made of redundancy is best illustrated by example. The examples which follow illustrate the uses of redundancy for the decoding of digit 1, but due to the cyclic property of circulants, can also be achieved on all $k' = k-1$ message-digits.

Consider the code; $n = 49$, $J = 21$,

$$G = \left[63,31,15,23,27,7,11,13,3,9\right]$$

The following redundancy is found on message-digit 1.

1000011
1100001
1100011
1101001
1010101
1001011
1001001

Adding all seven k-tuples modulo 2 we have 1111111. Ignoring digit k, this is the circulant {63} of order one. Thus 63 can be deleted from the generator matrix giving a code with, $n = 48$, $J = 21$. This usage will be denoted by deleting 63 and writing -I with the generating numbers.

If we divide the above redundancy into two sections we obtain the two forms A and B below.

$$\begin{array}{lllll} 1000011 & (A) & \qquad & 1101001 & (B) \\ 1100001 & & & 1010101 & \\ 1100011 & & & 1001011 & \\ \cline{1-1} 1000001 & & & 1001001 & \\ \cline{1-1} \cline{4-4} & & & 0111110 & \\ \cline{4-4} \end{array}$$

Note that for the message-digits other than 1 we will obtain 1000010, 1000100, 1001000, 1010000, 1100000 for A and 0111101, 0111011, 0110111, 0101111, 001111 for B. We make use of A by adding the circulant of order 1 generated by 0 with a parity-check on the k'th message-digit, that is the k-tuple 1000000. We denote this in the generating numbers by + 0. This gives the code n = 50, J = 22, {63,31,15,23,27,7,11,13,3,9,+0}. We make use of (B) by adding the circulant of order 1 generated by 63 but with no parity-check on the k'th digit, that is the k-tuple 0111111. We denote this in the generating numbers by +I. This gives the code,

n = 51, J = 23, {63,31,15,23,27,7,11,13,3,9,+0,+I}.

Outlined below are the best codes which could be found utilizing the redundancy in the codes given in section 4.4.

i) n = 7, J = 1 {63,31}.

$$\begin{array}{l} 1111101 \\ 1011111 \\ 1111011 \\ 1101111 \\ 1110111 \\ \hline 1000001 \\ \hline \end{array}$$

giving the code

n = 8, J = 2, {63,31,+0}.

ii) n = 13, J = 3 {63,31,15}.

$$\begin{array}{lll} 1111011 & \qquad & 1101111 \\ 1110011 & & 1100111 \\ 1111001 & & 1110111 \\ \cline{3-3} 1001111 & & 1111111 \\ \cline{1-1} \cline{3-3} 0111110 & & \\ \cline{1-1} \end{array}$$

giving codes

n = 12, J = 3 {31,15,-I}

n = 13, J = 4 {31,15,-I,+I}

Note that since {31} and {15} both have order six, the code $n = 12$, $J = 3$ is quasi-cyclic.

iii) $n = 18$, $J = 5$. {63,31,23,27,21}

1010101	1110101
1010111	1011101
$\underline{1111101}$	1011011
$\underline{1111111}$	1101101
	$\underline{1011111}$
	$\underline{1000001}$

giving codes

$n = 17$, $J = 5$. {31,23,27,21,-I}

$n = 18$, $J = 6$. {31,23,27,21,-I,+0}

iv) $n = 21$, $J = 6$. {63,31,15,23,21}

1110111	1110101	1100111
1001111	1010111	1110011
$\underline{1111001}$	$\underline{1011101}$	$\underline{1010101}$
$\underline{1000001}$	$\underline{1111111}$	$\underline{1000001}$

giving codes

$n = 20$, $J = 6$. {31,15,23,21,-I}

$n = 21$, $J = 7$. {31,15,23,21,-I,+0}

$n = 22$, $J = 8$. {31,15,23,21,-I,+0,+0}

v) $n = 25$, $J = 8$. {63,31,15,23,7}

1000111	1100011	1110001
1110011	1110111	1010111
$\underline{1110101}$	$\underline{1101011}$	$\underline{1100111}$
$\underline{1000001}$	$\underline{1111111}$	$\underline{1000001}$

giving codes

$n = 24$, $J = 8$. {31,15,23,7,-I}.

This code is also quasi-cyclic, since all the circulants have order 6.

n = 25, J = 9, {31,15,23,7,-I,+0}

n = 26, J = 10, {31,15,23,7,-I,+0,+0}

vi) n = 27, J = 9, {63,31,15,23,7,21}.

1010101	1000111	1010111
1110111	1110101	1100111
$\underline{1100011}$	$\underline{1110011}$	$\underline{1110001}$
$\underline{1000001}$	$\underline{1000001}$	$\underline{1000001}$

giving codes

n = 28, J = 10, {63,31,15,23,7,21,+0}

n = 29, J = 11, {63,31,15,23,7,21,+0,+0}

n = 30, J = 12, {63,31,15,23,7,21,+0,+0,+0}.

vii) n = 34, J = 12, {63,31,15,23,27,7,11}

1110001	1000111	1011001	1001011
1011011	1100011	$\underline{1100111}$	$\underline{1110101}$
$\underline{1101011}$	$\underline{1100101}$	$\underline{0111110}$	$\underline{0111110}$
$\underline{1000001}$	$\underline{1000001}$		

giving codes

n = 23, J = 13, {63,31,15,23,27,7,11,+0}

n = 36, J = 14, {63,31,15,23,27,7,11, +0,+0}

n = 37, J = 15, {63,31,15,23,27,7,11, +0,+0,+I}

n = 38, J = 16, {63,31,15,23,27,7,11, +0,+0,+I,+I} .

Alternatively we can rearrange the k-tuples and use the following sets.

1110001	1000111	1001011
1011011	1100011	$\underline{1110101}$
1101011	$\underline{1100101}$	$\underline{0111110}$
1011001	$\underline{1000001}$	
$\underline{1100111}$		
$\underline{1111111}$		

giving codes

n = 33, J = 12, {31,15,23,27,7,11,-I}

n = 34, J = 13, {31,15,23,27,7,11,-I,+0}

n = 35, J = 14, {31,15,23,27,7,11,-I,+0,+I}

viii) n = 40, J = 15, {63,31,15,23,27,7,11,13}

$$\begin{array}{ccc}
1001011 & 1100101 & 1000111 \\
1011001 & 1101001 & 1100011 \\
\underline{1010011} & \underline{1001101} & 1110001 \\
\underline{1000001} & \underline{1000001} & \underline{1101011} \\
 & & \underline{0111110}
\end{array}$$

giving codes

n = 41, J = 16, $\underbrace{\{63,31,15,23,27,7,11,13,}+0\}$

n = 42, J = 17, $\underbrace{\{\qquad '' \qquad ,}+0,+0\}$

n = 43, J = 18, { " ,+0,+0,+I}

ix) n = 43, J = 17, {63,31,15,23,27,7,11,13,9}

$$\begin{array}{ccc}
1101011 & 1101001 & 1001011 \\
1001001 & 1110001 & 1001101 \\
\underline{1100011} & \underline{1011001} & \underline{1000111} \\
\underline{1000001} & \underline{1000001} & \underline{1000001}
\end{array}$$

giving codes

n = 44, J = 18, $\underbrace{\{63,31,15,23,27,7,11,13,9,}+0\}$

n = 45, J = 19, $\underbrace{\{\qquad '' \qquad ,}+0,+0\}$

n = 46, J = 20, { " ,+0,+0,+0}

x) n = 49, J = 21.

See initial example.

xi) n = 54, J = 24, $\underbrace{\{63,31,15,23,27,7,11,13,21,3,5\}}$

$$\begin{array}{ll} 1000011 & 1100001 \\ 1010001 & 1000101 \\ \underline{1010011} & \underline{1100101} \\ \underline{1000001} & \underline{1000001} \end{array}$$

giving codes

$n = 55, \quad J = 25, \qquad \{\underbrace{63,31,15,23,27,7,11,13,21,3,5,}+0\}$

$n = 56, \quad J = 26, \qquad \{ \qquad \text{"} \qquad ,+0,+0\}$

In all constructions above the complete set of redundant k-tuples is shown. Most of the optimum codes are derived from the use of redundancy shown above. Generally any code where k' is a prime which uses the -I redundancy property only is quasi-cyclic. But, as we saw in the above constructions, quasi-cyclic codes can occur for composite k', though they may not be optimum codes for that k'.

There is an aspect of the utilization of redundancy which is not obvious in the examples presented. Of the three forms of redundancy possible, that is 111····111, 01111···110 and 1000···001, the most efficient form is 0111····110. The form 1000···001 requires the sum of at least three redundant k'-tuples and the form 1111····111 can only be used once. However the form 0111····110 can be obtained by the sum of two redundant k'-tuples in the most efficient case.

Example 4.5.1.

We set out below all those optimum codes developed using $a < 2^{k'}-1$, for k' = 6,5,4,3,2. Quasi-cyclic codes are denoted q.c.

k' = 6.

Code length n	Check sums J	Generating numbers a ∈ G
7	1	63,31.
8	2	63,31,+0.
q.c. 12	3	31,15,-I.
13	4	31,15,-I,+I.
17	5	31,23,27,21,-I.
18	6	31,23,27,27,-I,+0.
21	7	31,15,23,21,-I,+0.
22	8	31,15,23,21,-I,+0,+0.
25	9	31,15,23,7,-I,+0.
26	10	31,15,23,7,-I,+0,+0.
29	11	$\underbrace{63,31,15,23,7,21,+0,+0}$.
30	12	" ,+0.
34	13	$\underbrace{31,15,23,27,7,11,-I,+0}$.
35	14	" ,+I.
37	15	$\underbrace{63,31,15,23,27,7,11,+0,+0,+I}$.
38	16	" ,+I.
42	17	$\underbrace{63,31,15,23,27,7,11,13,+0,+0}$.
43	18	" ,+I.
45	19	$\underbrace{63,31,15,23,27,7,11,13,9,+0,+0}$.
46	20	" ,+0.
48	21	31,15,23,27,7,11,13,3,9,-I.
50	22	$\underbrace{63,31,15,23,27,7,11,13,3,9,+0}$.
51	23	" ,+I.
54	24	$\underbrace{63,31,15,23,27,7,11,13,21,3,5}$.
55	25	" ,+0.
56	26	" ,+0,+0.
63	31	$\underbrace{63,31,27,15,23,21,7,11,13,9,3,5,1}$.
64	32	" ,0.

k' = 5.

Code length n	Check sums J	Generating numbers $a \in G$
6	1	31,15.
7	2	31,15,+I.
q.c. 10	3	15,7,-I.
11	4	15,7,+I,-I.
q.c. 15	5	$\underbrace{15,7,11,-I}$.
16	6	" ,+0.
q.c. 20	8	15,7,11,3.
22	9	$\underbrace{31,15,7,11,3,+0}$.
23	10	" ,+I.
26	11	$\underbrace{31,15,7,11,3,5}$.
27	12	" ,+I.
31	15	$\underbrace{31,15,7,11,3,5,1}$.
32	16	" ,+0.

k' = 4.

n	J	$a \in G$.
5	1	15,7.
6	2	15,7,+0.
9	3	15,7,3.
10	4	15,7,3,+0.
13	5	15,7,3,1.
14	6	15,7,3,1,+0.
15	7	15,7,3,5,1.
16	8	15,7,3,5,1,0.

k' = 3.

n	J	$a \in G$.
4	1	7,3.
5	2	7,3,+0.
7	3	7,3,1.
8	4	7,3,1,0.

k' = 2.

n	J	$a \in G$.
3	1	3,1.
4	2	3,1,0.

4.6 Amount of Redundancy.

Let Rd(b) be defined as the number of columns of the generator matrix, or similarly the number of received digits, not used to decode message-digit m(i), $1 \leq i \leq k'$, where b is the smallest binary weight of the circulant sets used in G.

If the generator matrix comprises the complete sets of k'-tuples, of weights k', k'-1, k'-2,....,b, then the redundancy occurs only in the set of weight b. From theorem 4.2.1. we know that the number of k'-tuples of weight b used to form check sums is equal to the number of zero's in the column of b-weight k'-tuples. Therefore,

$$J = \frac{(k'-1)!}{b!\ (k'-1-b)!} \qquad 4.6.1.$$

Since there are a total of N_b k'-tuples of weight b, where

$$N_b = \frac{k'!}{b!\ (k-b)!}$$

then the redundancy in this case is given by,

$$\begin{aligned} Rd(b) &= N_b - J \\ &= \frac{k!}{b!\ (k-b)!} - \frac{(k'-1)!}{b!\ (k'-1-b)!} \\ &= \frac{(k'-1)!}{(b-1)!\ (k-b)!} \qquad 4.6.2.a. \end{aligned}$$

which is equal to the number of binary ones in a column of the column of b-weight k'-tuples. In example 4.2.1. the redundancy in the set b = 3, is

$$\begin{aligned} &= N_3 - J \\ &= 10 - 4 = 6. \end{aligned}$$

If the final set of k'-tuples used is not complete, that is circulant sets of weight b are used, we can write,

$$Rd(b) = Rd(b+1) - C(b). \qquad 4.6.2.b.$$

Here Rd(b+1) is the redundancy when the final k'-tuple set has weight (b+1), and C(b) is the change brought about by adding an incomplete set of weight b.

If the reduction, C(b), is effected by one circulant of order p, then from theorem 4.2.2. it has J_p zero's per column, where,

$$J_p = \frac{p(k'-b)}{k'} \qquad 4.6.3.$$

Thus, the number of binary one's per column, I_p, is given by,

$$I_p = p - \frac{p(k'-b)}{k'} .$$

Therefore,

$$\begin{aligned} C(b) &= J_p - I_p \\ &= \frac{2p(k'-b)}{k'} - p \\ &= p - \frac{2pb}{k'} \qquad 4.6.4. \end{aligned}$$

If there are m_i circulants of order p_i

$$C(b) = \sum_i m_i \left(p_i - \frac{2p_i b}{k'} \right) \qquad 4.6.5.$$

and generally,

$$Rd(b) = Rd(b+1) - \sum_i m_i \left(p_i - \frac{2p_i b}{k'} \right) \qquad 4.6.6.$$

Example 4.6.1.

Consider the code whose generator matrix columns utilize k'-tuples with k' = 17, down to b = 2. From equation 4.6.2.b.

$$Rd(2) = Rd(3) - C(2).$$

Since k' is a prime only circulants of order 17 are possible. The number of weight 2,17-tuples is,

$$N_2 = \frac{17!}{2!\ 18!} = 136.$$

Therefore, let

$$m_1 = \frac{136}{17} = 8$$

$$p_1 = 17$$

$m_i = 0$, for $i > 1$.

Let $1 \leqslant m \leqslant 8$ then from equation 4.6.5.

$$C(2) = m \left(p_1 - \frac{2p_1 . 2}{17}\right)$$

$$= m.13$$

and since, from equation 4.6.2.a

$$Rd(3) = \frac{16!}{2!\ 14!} = 120$$

from equation 4.6.6.

$$Rd(2) = 120 - m.13. \qquad 1 \leqslant m \leqslant 8$$

and we reduce the redundancy by 13 each time a circulant is added.

Before adding any circulants we will have a code with, length

$$n = \sum_{b=3}^{17} \frac{k'!}{b!\ (k'-b)!} = 130{,}918$$

check sums,

$$J = \sum_{b=3}^{16} \frac{(k'-1)!}{b!\ (k'-1-b)!} = 65{,}399$$

$$Rd(3) = 120.$$

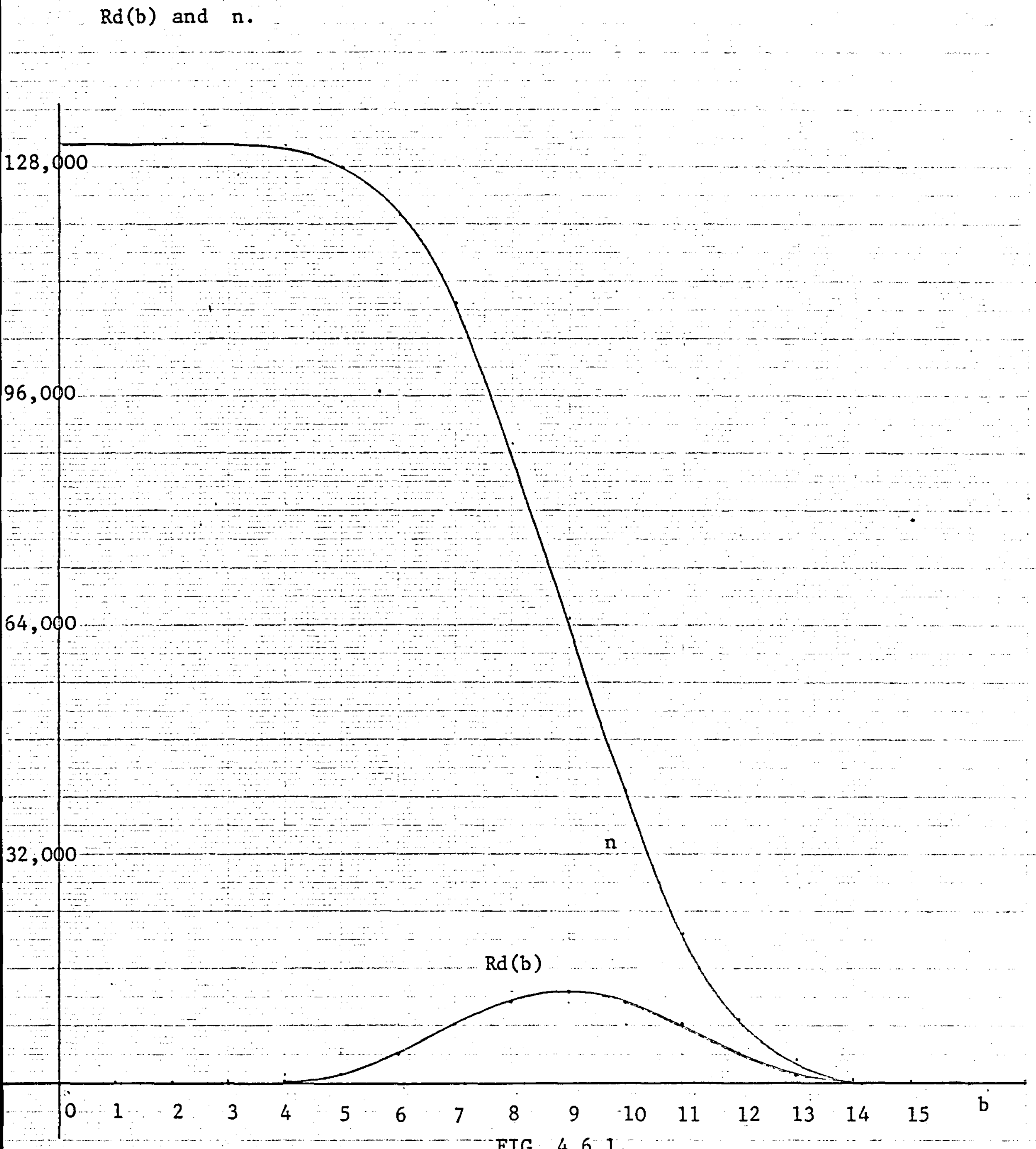


FIG. 4.6.1.

Rd(b) AND n AS A FUNCTION OF b.

Each time we add a circulant we increase n by 17, decrease Rd(b) by 13 and increase J by $J_p = \frac{17(17-2)}{17} = 15$, giving the set of codes below.

n	J	Rd(b)	k
130,918	65,399	120	18
130,935	65,414	107	18
130,952	65,429	94	18
130,969	65,444	81	18
130,986	65,459	68	18
131,003	65,474	55	18
131,020	65,489	42	18
131,037	65,504	29	18
131,054	65,519	16	18

We can extend our example and at the same time our investigation of redundancy by examining how Rd(b) and the code length vary with b. In fig. 4.6.1. we have plotted

$$Rd(b) = \frac{(k'-1)!}{(b-1)!\ (k'-b)!}$$

and

$$n = \sum_{b}^{k'} \frac{k'!}{b!\ (k'-b)!}$$

for values of, $k' = 17$, $0 \leqslant b \leqslant 17$, that is just codes where full sets of k'-tuples of weight b are used. Since addition of extra circulants increases or decreases Rd(b) linearly between these points and increases n linearly, the resultant curves apply to all codes with $k' = 17$, $n \leqslant 2^{k'}$.

Let n(R) be that value of n when Rd(b) is at its maximum, then it is apparent that the codes are more efficient when $n > n(R)$, in terms of redundancy. If k' is odd, Rd(b) will be a maximum when,

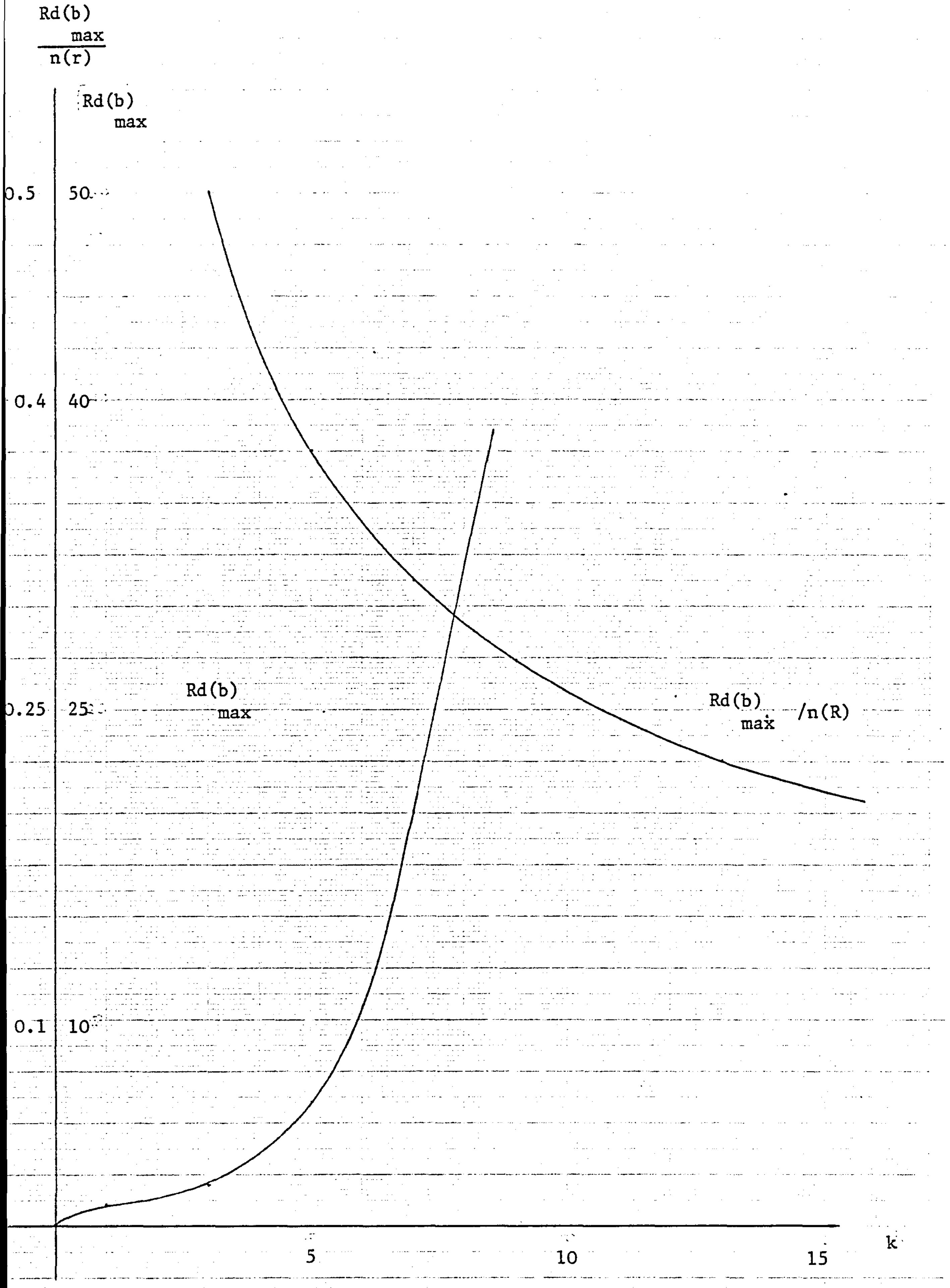


FIG. 4.6.2.

PLOT OF $Rd(b)_{max}$ /n(R) AGAINST k.

$$(b-1)! = (k'-b)! \quad \text{that is when}$$

$$b-1 = k'-b \quad \text{or}$$

$$b = \frac{k'+1}{2} \quad \text{or}$$

$$b-1 = \frac{k'-1}{2}\ . \qquad 4.6.7.$$

Therefore,

$$Rd(b)_{max} = \frac{(k'-1)!}{\left(\frac{k'-1}{2}\right)!\ \left(\frac{k'-1}{2}\right)!}$$

$$n(R) = \sum_{b=\frac{k'+1}{2}}^{k'} \frac{k'!}{b!\ (k'-b)!}$$

$$= 2^{k'-1}\ .$$

If k' is even, Rd(b) is a maximum when

$$(b-1)! = (k'-b\pm 1)!$$

say $b-1 = k'-b-1$

and $b = \frac{k'}{2}\ , \quad b-1 = \frac{k'}{2} - 1$

$$Rd(b)_{max} = \frac{(k'-1)!}{\left(\frac{k'}{2} - 1\right)!\ \left(\frac{k'}{2}\right)!}$$

$$n(R) = \sum_{b=\frac{k'}{2}}^{k'} \frac{k'!}{b!\ (k'-b)!}$$

this value of b gives the largest value for n(R).

Figure 4.6.2. shows a plot of the ratio $Rd(b)_{max}/n(R)$, when k' is various increasing odd values. Although this ratio decreases exponentially, $Rd(b)_{max}$ is increasing exponentially as figure 4.6.2. also shows. The length n is simply increasing faster than Rd(b).

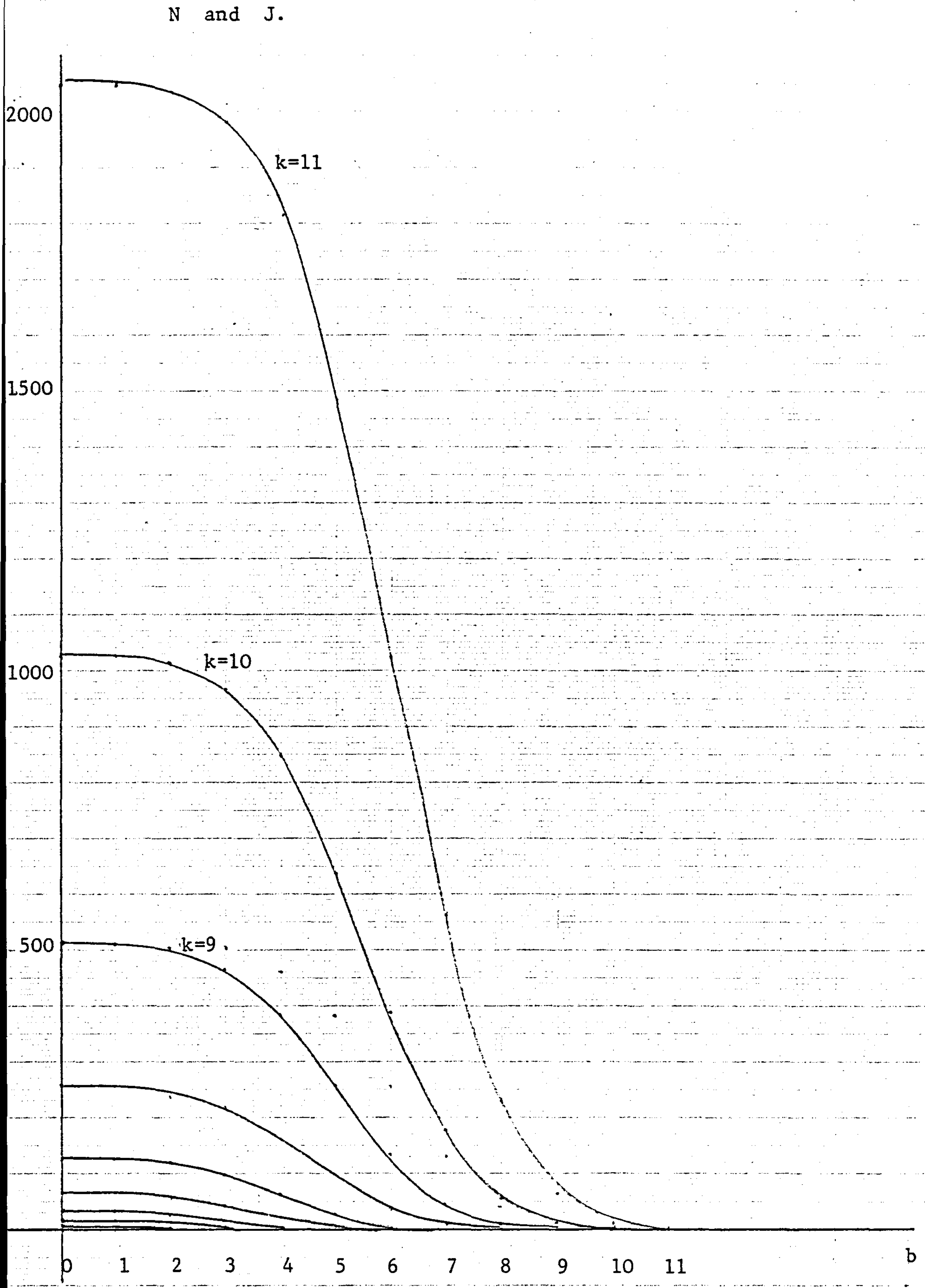


FIG. 4.6.3.

INCREASING LENGTH N AND CHECK-SUMS. J FOR INCREASING k, MESSAGE-LENGTH.

In figure 4.6.3. we show a family of curves for n increasing with b, for various values of k'. If one curve is plotted for some value of k', say n_1, then

$$n_1 = \sum_{b}^{k'} \frac{k'!}{b!\,(k'-b)!} \qquad 0 \leq b \leq k'$$

then the curve for (k'-1) is plotted for,

$$n_2 = \sum_{b}^{k'-1} \frac{(k'-1)!}{b!\,(k'-1-b)!}$$

But for the system using k',

$$J = \sum_{b}^{k'-1} \frac{(k'-1)!}{b!\,(k'-1-b)!}$$

and therefore,

$$n_2 = J$$

for the system of k'.

4.7 Minimum Distance of the Codes.

It is a property of all codes that if one can obtain J orthogonal check sums on the message-digits, then to decode correctly in the presence of t errors, one requires J = 2t+1. If, with this type of check sum, $J = d_m$, the minimum distance of the code, the code is said to be completely orthogonalizable up to its minimum distance.

Consider the codes developed in section 4.4. Since we have J check sums we, at least, are sure that $d_m \geq J$. The codes were developed by ensuring that each row of the generator matrix, except the row which forms an overall parity-check on the k'th digit, has J zero's. Thus the linear combination of any of these rows with the overall parity-check row will produce a code word, in the code, whose binary weight is equal to J. But if there exists a code word weight J, then $J \geqslant d_m$ for this code. Since, $J \geqslant d_m \geqslant J$, $d_m = J$.

So that the codes in section 4.4 are completely orthogonalizable up to their minimum distances.

We merely have to show that by utilizing redundancy, as in section 4.5., we maintain the same relationship between J and d_m.

We utilized redundancy in the following ways.

i)	deleted the k-tuple	$\overbrace{1111....111}^{\text{k-digits}}$
ii)	added the k-tuples	1000....000
iii)	added the k-tuples	0111....111

a) From (i) the value of J remains the same, so that $d_m \geqslant J$. In the original code adding the i'th and k'th rows of G mod 2, gave a code word of weight J, and since the mod 2 sum of the i'th and k'th digit of 1111....111 is zero, after deleting this k-tuple there still exists a code word of weight J. Therefore,

$$J \geqslant d_m \geqslant J \quad \text{and} \quad J = d_m.$$

b) From (ii) we increase J by one, so let the new $J' = J+1$, then $d_m \geqslant J'$. In the original code adding the i'th and k'th row of G mod 2 gave a code word of weight J. Adding the k-tuple 1000....000, to the generator matrix will produce a code word of weight J plus the weight of the mod 2 sum of the i'th and k'th digits of the additional k-tuple. This is always 1 so there exists a code word of weight $J' = J+1$, and since $J' \geqslant d_m \geqslant J'$, then $d_m = J'$.

c) The argument for the k-tuple 0111....111, is the same as above for 1000....000.

So that the optimum set of codes developed in section 4.5. are completely orthogonalizable also.

We can show in fact that

$$d_m = \frac{n - Rd(b)}{2} \qquad 4.7.1.$$

Consider the length of those codes whose generator matrix is composed of complete sets of k'-tuples, with overall parity-check, down to b-weight k'-tuple sets.

$$n = \sum_{t=b}^{k'} \frac{(k')!}{t!\ (k'-t)!}$$

$$= 1 + \sum_{t=b}^{k'-1} \frac{k'!}{t!\ (k'-t)!}$$

but

$$\frac{k'!}{t!\ (k'-t)!} = \frac{(k'-1)!}{t!\ (k'-1-t)!} + \frac{(k'-1)!}{(t-1)!\ (k'-1-(t-1))!}$$

and

$$n = 1 + \sum_{t=b}^{k'-1} \frac{(k'-1)!}{t!\ (k'-1-t)!} + \sum_{t=b}^{k'-1} \frac{(k'-1)!}{(t-1)!\ (k'-1-(t-1))!} \qquad 4.7.2.$$

But since the number of check sums obtainable, ignoring redundancy, is,

$$J = \sum_{t=b}^{k'-1} \frac{(k'-1)!}{t!\ (k'-1-t)!} \qquad 4.7.3.$$

then from equation 4.7.2.,

$$n = 1 + J + \sum_{t=b}^{k'-1} \frac{(k'-1)!}{(t-1)!\ (k'-1-(t-1))!} \qquad 4.7.4.$$

If we expand the final term, of equation 4.7.4.

$$= \frac{(k'-1)}{1!} + \frac{(k'-1)(k'-2)}{2!} + \ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots + \frac{(k'-1)(k'-2)\ldots\ldots\ldots(k'-b+1)}{(b-1)!}$$

Similarly if we expand J, equation 4.7.3.

$$= 1 + \frac{(k'-1)}{1!} + \frac{(k'-1)(k'-2)}{2!} + \ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots + \frac{(k'-1)(k'-2)\ldots\ldots\ldots(k'-b)}{b!}$$

So that

$$\sum_{t=b}^{k'-1} \frac{(k'-1)!}{(t-1)!\,(k'-1-(t-1))!} = J - 1 + \frac{(k'-1)(k'-2)\ldots\ldots(k'-b+1)}{(b-1)!}$$

$$= J - 1 + \frac{(k'-1)!}{(b-1)!\,(k'-b)!}$$

therefore putting in equation 4.7.4.

$$n = 2J + \frac{(k'-1)!}{(b-1)!\,(k'-b)!} \qquad 4.7.5.$$

but,

$$J = d_m$$

and

$$Rd(b) = \frac{(k'-1)!}{(b-1)!\,(k'-b)!}$$

therefore,

$$n = 2d_m + Rd(b). \qquad 4.7.6.$$

When incomplete sets of weight b are used, (that is circulants), Rd(b) is the number of k'-tuples unused after forming all possible check sums by adding pairs of k'-tuples. Therefore the above equation for n holds provided the modified form of Rd(b) is used, from equation 4.6.6.

$$n = 2d_m + Rd(b+1) - \sum_i m_i \left(p_i - \frac{2p_i b}{k'}\right).$$

When one utilizes redundancy this expression is modified slightly, in the following ways.

a) Using -I.

The length is reduced by one and the minimum distance remains the same, therefore, if n' is the new length,

$$n' = 2d_m + Rd(b)-1.$$

b) Using +0.

The length and minimum distance are increased by one, but

$$n' = 2(d_m+1) + Rd(b) = n+2, \text{ so that}$$

$$n' = 2d'_m + Rd(b) - 1 = n+1,$$

where d'_m is the new minimum distance. If we add generally m,+0 k-tuples, the length increases by m, and $d'_m = d_m + m$.

But

$$n' = 2(d_m+m) + Rd(b) = n + 2m, \text{ so that}$$

$$n' = 2d'_m + Rd(b) - m = n + m.$$

c) Using +I.

The results are the same as for +0, generally adding m,+I k-tuples, gives

$$n' = 2d'_m + Rd(b) - m.$$

Since for the codes listed in Example 4.5.1., J = d_m, we compare them with d_m from Helgert and Stinaff[40], below, in Table 4.7.1.

Hel gert and Stinaff[40] is a table of maximum minimum distances for all codes of length n $\leqslant$ 127 and information digits k $\leqslant$ 127. It effectively gives the maximum minimum distance possible, as specified by all known upper bounds at that time, for a given n and k.

TABLE 4.7.1.

n	k	d_m	d_m^{40}
7	7	1	1
8	"	2	2
12	"	3	4
13	"	4	4
17	"	5	6
18	"	6	7
21	"	7	8
22	"	8	8
25	"	9	10
26	"	10	11
29	"	11	12^{E}
30	"	12	12-13
34	"	13	14-16
35	"	14	15-16
37	"	15	16
38	"	16	16^{E}
42	"	17	17-19
43	"	18	18-20
45	"	19	20^{AE}
46	"	20	20-21
48	"	21	22^{E}
50	"	22	24
51	"	23	24
54	"	24	24-26
55	"	25	24-26
56	"	26	24-27

TABLE 4.7.1. (Contd)

n	k	d_m	d_m^{40}
63	7	31	31^I
64	"	32	32
6	6	1	1
7	"	2	2
10	"	3	3
11	"	4	4
15	"	5	6
16	"	6	6^E
20	"	8	8
22	"	9	9^G
23	"	10	10
26	"	11	12
27	"	12	12
31	"	15	15^I
32	"	16	16
5	5	1	1
6	"	2	2
9	"	3	3
10	"	4	4
13	"	5	5
14	"	6	6
15	"	7	7^B
16	"	8	8
4	4	1	1
5	"	2	2
7	"	3	3^I
8	"	4	4
3	3	1	1
4	"	2	2

The raised alphabetic indices refer to various bounds in the tables of Helegert and Stinaff[40].

15	8	4	4	(63,55,-I,+O)
14	8	3	4	(63,55,-I).

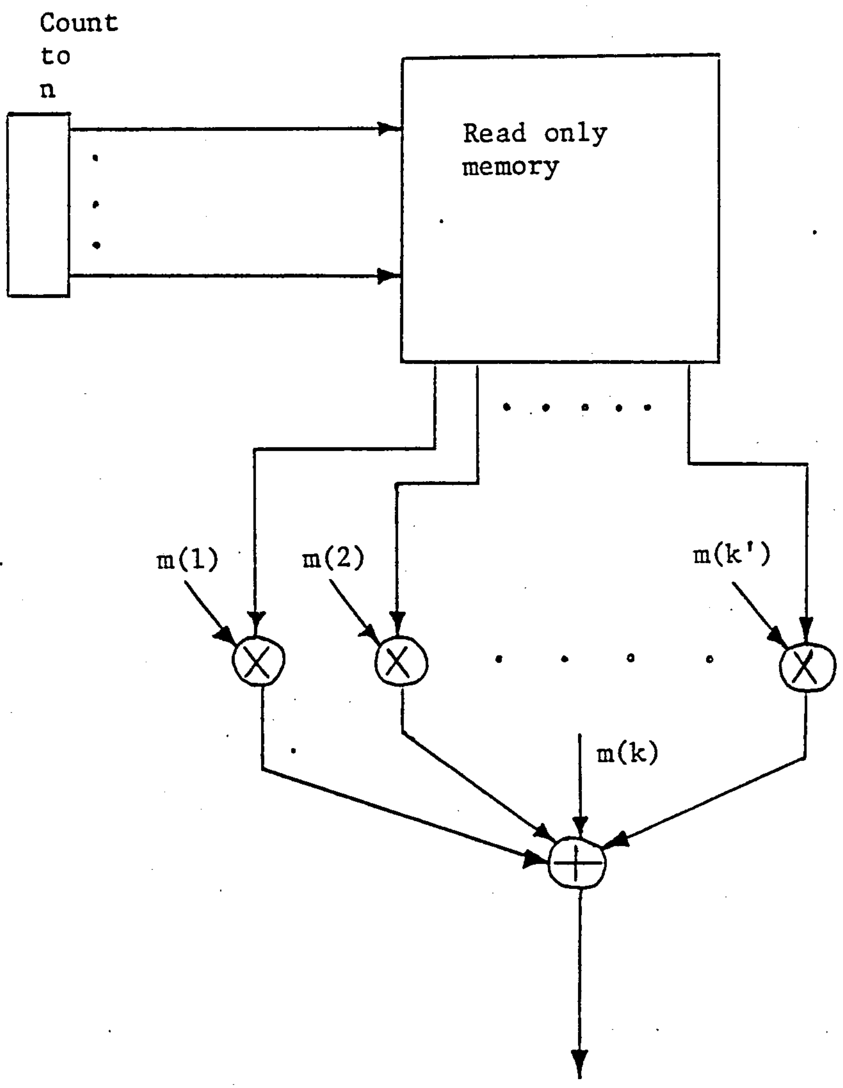


FIG. 4.8.1.

GENERAL ENCODER.

4.8 Encoding and Decoding.

Since it is not the purpose of this thesis to investigate the hardware implementation of encoders and decoders for error-correcting codes, a simple and informal method is offered.

4.8.1. Encoding.

Figure 4.8.1. offers an encoding method which has reasonable storage. A read-only-memory is addressed by a binary counter which counts up to n, the length of the code. For each value of n, the read-only-memory outputs a k'-digit binary k'-tuple corresponding to the n'th column of the generator matrix. Each digit of the k'-tuple is weighted, by multiplication modulo 2, by the corresponding message-digit. The k' weighted digits are summed and added to digit k, the resultant being an encoded digit.

The total storage then being S,

$$S = \left[{}_2\log_2 (n) \right] \cdot k' \text{ .}$$

One advantage being that if k' is relatively large but the code length short, then the storage is small. Read-only-memories exist at present that can be addressed by 12 input lines, so that this is limited to codes of length $n \leq 2^{12}$. The number of output lines is not critical since the output k'-tuple can be sub-divided among a number of read-only-memories, each of which is addressed by the $\left[\log_2 n \right]$ inputs.

4.8.2. Decoding.

In section 4.3. we saw that if a set of check sums can be obtained on a digit, then a similar set can be obtained by cyclically shifting

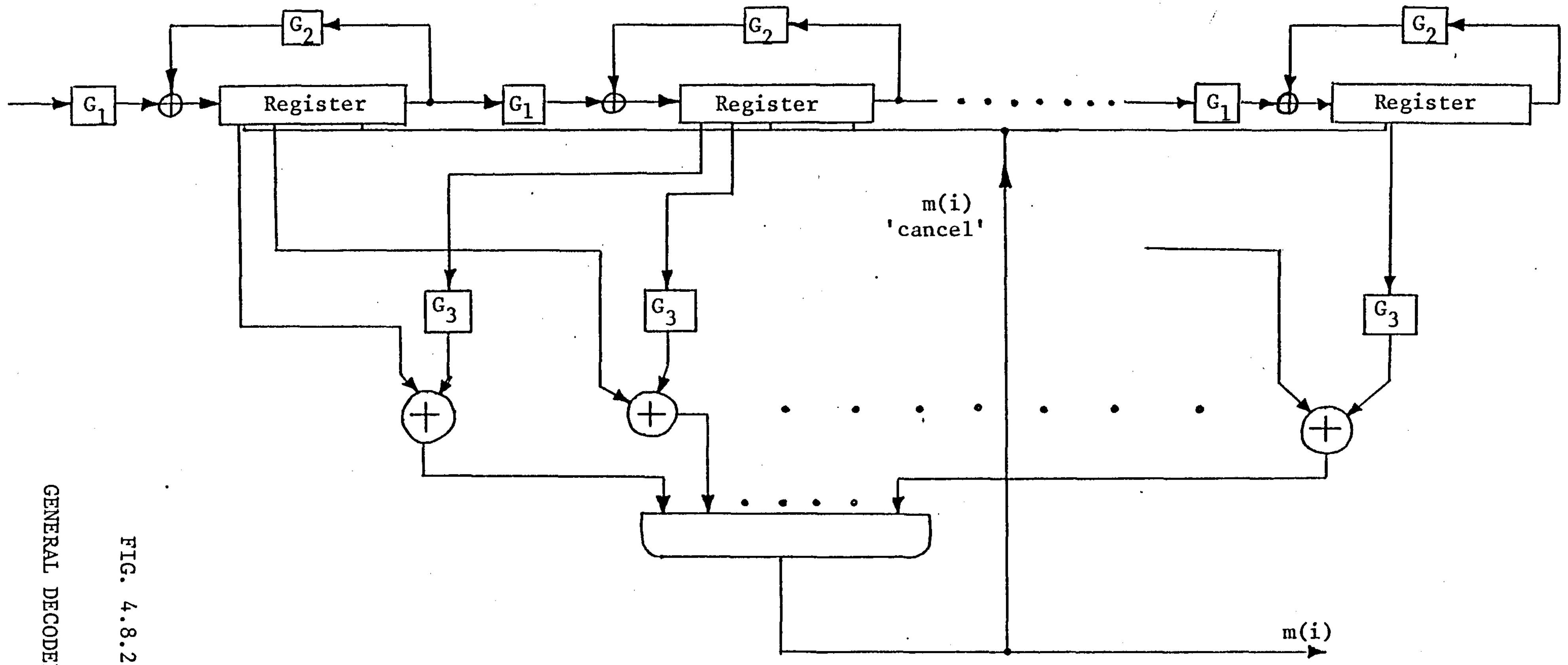


FIG. 4.8.2.

GENERAL DECODER.

all circulants one digit, with respect to themselves.

In the decoder then we have a shift-register of length p, corresponding to each circulant of order p, contained in the generator matrix. Connections are made from the shift-registers to obtain J check sums on the first message-digit. This is shown in figure 4.8.2. only in a general way. Decoding is then performed in the following way.

i) with gate G_2 closed and G_1, and G_3 open, the received word is fed in.

ii) digit m(1) is decoded, and its effect cancelled from the shift-registers.

iii) gate G_1 is closed, G_2, and G_3 opened and the shift-registers clocked once.

iv) the second digit m(2) is decoded and cancelled.

v) continue iii) and iv) until $k' = k-1$, of the message-digits have been decoded.

vi) at this point, provided $t \leqslant (d_m-1)/2$ errors occurred, the shift-registers contain approximately n estimates of the k'th digit, in the presence of the t errors.

vii) with G_1 closed, when the registers are clocked for the k'th time, G_3 is closed so that each adder sees only one input, and digit k is decoded and cancelled.

viii) steps ii) to vii) can be repeated to check for uncorrectable error patterns.

CHAPTER 5

5. CODES DERIVED FROM THE CLASS OF BINARY CODES.

5.1 Introduction.

The most useful class derived below from the constructions in Chapter 4 are those codes where $k = k'$, and the overall parity-check is removed. We show that it is possible to obtain a better code sometimes with this method.

We also examine codes derived from extending the generator matrix in various ways and obtain one class which is completely orthogonalizable with $n = 2\ d_m$.

However some classes will be seen not to be cyclically decodable and this puts a practical limitation on their use.

5.2 Codes without the Overall Parity-check.

The codes developed in Chapter 4 assumed the generator matrix contained an overall parity-check on the k'th message-digit. The codes were developed using circulants such that J orthogonal check sums could be guaranteed on the first k' message-digits and the k'th message-digit was decoded by cancelling the decoded estimates of the first k' message-digits, from the received code word.

Consider the codes developed in section 4.4., then, if the overall parity-check is removed we will have a code in which we can guarantee at least J check sums on all $k = k'$ message-digits. The code length n, will be the same, but we are no longer sure of d_m, based upon the arguments in section 4.7. The redundancy is determined by the circulants and is therefore the same, but a more efficient use can be made of it now.

The uses of redundancy can now be outlined as below.

a) The k-tuple 111....111.

As before we remove the circulant generated by 2^k-1, from the generator matrix. The length is reduced by one and J remains the same.

b) The k-tuple 1000....001.

Since the overall parity-check has been removed, this has the form 000....001 and is a valid check sum requiring no increase in length.

c) The k-tuple 0111....110.

Again this now appears as 111....110 and can be used by the addition of the circulant generated by 2^k-1.

Consider the codes developed in section 4.5., we will now re-examine the redundancy and develop codes, for k = 6.

i) n = 7, J = 1, {63,31}

```
111101
011111        this is now a valid check sum and gives the code,
111011        n = 7,  J = 2.
101111
110111
------
000001
------
```

ii) n = 13, J = 3, {63,31,15}

In Chapter 4 we obtained 1111111 and 0111110. Deleting the k'th digit gives 111111 and 111110, but from (a) and (b) above this would imply deleting and adding the circulant generated by 2^k-1, which is pointless, so we use,

111011	
110011	giving the code,
111001	n = 13, J = 4, {63,31,15}.
001111	
101111	
100111	
$\underline{110111}$	
$\underline{000001}$	

iii) n = 18, J = 5, {63,31,23,27,21}

010101	110101	
010111	011101	giving codes,
$\underline{111101}$	011011	n = 17, J = 6, {31,23,27,21}.
$\underline{111111}$	101101	
	$\underline{011111}$	
	$\underline{000001}$	

iv) n = 21, J = 6, {63,31,15,23,21}

110111	110101	100111	
001111	010111	110011	giving,
$\underline{111001}$	$\underline{011101}$	$\underline{010101}$	n = 20, J = 8, {31,15,23,21}.
$\underline{000001}$	$\underline{111111}$	$\underline{000001}$	

v) n = 25, J = 8 {63,31,15,23,7}

000111	100011	110001	
110011	110111	010111	giving,
$\underline{110101}$	$\underline{101011}$	$\underline{100111}$	n = 24, J = 10, {31,15,23,7}.
$\underline{000001}$	$\underline{111111}$	$\underline{000001}$	

Note, this code is quasi-cyclic, since all circulants have the same order.

vi) n = 27, J = 9, {63,31,15,23,7,21}

010101	000111	010111	
110111	110101	100111	giving,
$\underline{100011}$	$\underline{110011}$	$\underline{110001}$	n = 27, J = 12, {63,31,15,23,7,21}.
$\underline{000001}$	$\underline{000001}$	$\underline{000001}$	

vii) n = 34, J = 12, {63,31,15,23,27,7,11}

110001	000111	011001	001011
011011	100011	$\underline{100111}$	$\underline{110101}$
$\underline{101011}$	$\underline{100101}$	$\underline{111110}$	$\underline{111110}$
$\underline{000001}$	$\underline{000001}$		

giving codes,

n = 34, J = 14, $\underbrace{\{63,31,15,23,27,7,11\}}$

n = 35, J = 15, $\underbrace{\{ \qquad '' \qquad ,}$+I}

n = 36, J = 16, $\underbrace{\{ \qquad '' \qquad ,}$+I,+I}

Without repeating section 4.5. the following codes are also possible.

n = 40, J = 17, $\underbrace{\{63,31,15,23,27,7,11,13}$}

n = 41, J = 18, $\underbrace{\{ \qquad '' \qquad}$,+I}

n = 43, J = 20, $\underbrace{\{ \qquad '' \qquad}$,9}

n = 48, J = 22, $\underbrace{\{ \qquad '' \qquad}$,3}

n = 50, J = 23, $\underbrace{\{ \qquad '' \qquad}$,3,9,+I}

n = 54, J = 26, { " ,21,3,5}

Comparison with the optimum for k = 6, developed in Chapter 4, we can see that the codes above are never an improvement. Although at the points where the performance is equal, the above codes are easier to decode, since gating to decode the k'th digit is not required. However sometimes these codes are better than those of Chapter 4's optimum set. To show this consider the code below, for k = 7.

$n = 29$, $J = 7$, $\{127,63,55,47,31\}$

from redundancy we surprisingly obtain

1011101	1010111	1110101	1111001	0111101
0110111	0111011	1011011	1100111	1110011
1101011	1101101	0101111	0011111	1001111
0000001	0000001	0000001	0000001	0000001

giving,

$n = 29$, $J = 12$, $\{127,63,55,47,31\}$.

Note, if the generating number 127, is deleted we obtain the quasi-cyclic code.

$n = 28$, $J = 11$, $\{63,55,47,31\}$.

The best codes from Chapter 4 for $k = k'+1 = 7$, were,

$n = 29$, $J = 11$, and $n = 28$, $J = 10$.

Obviously then, when choosing a code with given k and error-correcting capability t, both constructions should be considered.

Encoding and decoding are implemented in the same manner as for Chapter 4 without special provision for the k'th digit, which is not now necessary.

The codes developed above, for the sake of clarity, are given in Table 5.2.1. below and compared with d_m from Helegert and Stinaff[40].

Another code is included in Table 5.2.1. which is better than the construction in Chapter 4 that is $n = 21$, $k = 7$, $J = 8$ compared with $n = 21$, $k = 7$, $d_m = 7$ in Chapter 4.

In section 4.5. we saw that the most efficient form of redundancy was the form 0111....110. However with the overall parity-check removed the most efficient form is that which required no increase in length, that is, the form 000....001.

TABLE 5.2.1.

n	k	J	d_m^{40}	
7	6	2	2	
13	6	4	4	
17		6	7	
20		8	8	
24		10	10^E	
27		12	12	
34		14	16	
35		15	16	
36		16	16	
40		17	18-19	
41		18	19-20	
43		20	20	
48		22	22	
50		23	24	
54		26	26	
29	7	12	12^E	
28	7	11	12	
21	7	8	8	{63,55,47,-I}
15	5	6	7	
10	5	4	4	
21	5	9	10	
9	4	4	4	
13	4	6	6	

5.3 Extending the Generator Matrix.

With the codes developed in Chapter 4 one eventually utilizes all circulants from the system of k'-tuples and obtains the (n, k = k'+1) code, with J check sums and redundancy Rd(b), where,

$$\left.\begin{aligned} n &= \sum_{b=0}^{k'} \frac{k'!}{b!\,(k-b)!} = 2^{k'} \\ J &= \sum_{b=0}^{k'-1} \frac{(k'-1)!}{b!\,(k'-1-b)!} = 2^{k'-1} \\ Rd(b) &= Rd(0) = 0. \end{aligned}\right\} \qquad 5.3.1.$$

Obviously this is a very efficient set of k-tuples and implies all check sums can be obtained by adding pairs of received digits. We will represent such sets of k-tuples, that is all $2^{k'}$ k'-tuples plus overall parity-check, by the symbol $S_1^{k'}$. We will show constructions where $S_1^{k'}$ is considered as the basic set present in the generator matrix and other sets are added to the basic set, to extend the generator matrix.

Let $S^{k'}$ be the complete set of k'-tuples with no overall parity-check. This set and others to be introduced are to be interpreted as follows. If a set S_1^d is used in the generator matrix of a code with k digits of information, then each (d+1)-tuple $\in S_1^d$ is considered as a k-tuple whose k-d-1 other digits are zero's.

Consider the set of k-tuples, S^k, which can be written,

$$S^k = S_1^{k-1} \cup S^{k-1} \qquad 5.3.2.$$

where $\cup$ is set union.

For example,

$$S^4 = S_1^3 \cup S^3$$

gives,

$$\begin{array}{cc} & S_1^3 \qquad\qquad \cup \qquad\qquad S^3 \\ & 1\,1\,1\,1\,1\,1\,1\,1 \quad 0\,0\,0\,0\,0\,0\,0\,0 \\ S^4 = & 1\,1\,1\,0\,1\,0\,0\,0 \quad 1\,1\,1\,0\,1\,0\,0\,0 \\ & 1\,1\,0\,1\,0\,1\,0\,0 \quad 1\,1\,0\,1\,0\,1\,0\,0 \\ & 1\,0\,1\,1\,0\,0\,1\,0 \quad 1\,0\,1\,1\,0\,0\,1\,0 \end{array}$$

However from equation 5.3.2.,

$$S^{k-1} = S_1^{k-2} \cup S^{k-2}$$

$$S^{k-2} = S_1^{k-3} \cup S^{k-3}$$

$$\vdots$$

$$S^{k-(k-2)} = S_1^{1} \cup S^{1}$$

So that we can rewrite equation 5.3.2. in the following form.

$$S^k = S_1^{k-1} \cup S_1^{k-2} \cup S_1^{k-3} \cup \ \ldots. \cup S_1^1 \cup S^1$$

To extend the generator matrix, for the codes of this section, we begin with the set $S_1^{k'} = S_1^{k-1}$, and extend by forming its union with other sets S_1^{k-2}, S_1^{k-3} etc. until we can extend no more, whereby S^k is contained in G.

Consider a code whose generator matrix contains the set,

$$S = S_1^{k'} \cup S_1^{k'-1} \cup S_1^{k'-2} \cup \ldots \cup S_1^{k'-i} .$$

then, $k = k'+1$ and from equations 5.3.1.

$$n = 2^{k'} + 2^{k'-1} + 2^{k'-2} + \ldots. + 2^{k'-i}$$

Also, from $S_1^{k'}$ we can obtain $2^{k'-1}$ check sums on the first k' message-digits, but, from equation 5.3.1.

from $S_1^{k'-1}$, $J = 2^{k'-2}$ on first k'-1 message-digits

from $S_1^{k'-2}$, $J = 2^{k'-3}$ on first k'-2 message-digits

$\vdots$

from $S_1^{k'-i}$, $J = 2^{k'-i-1}$ on first k'-i message-digits.

Therefore,

$$J = 2^{k'-1} + 2^{k'-2} + 2^{k'-3} + \ldots.. + 2^{k'-i-1} \qquad 5.3.3.$$

only for the first k'-i message-digits. If the columns of the

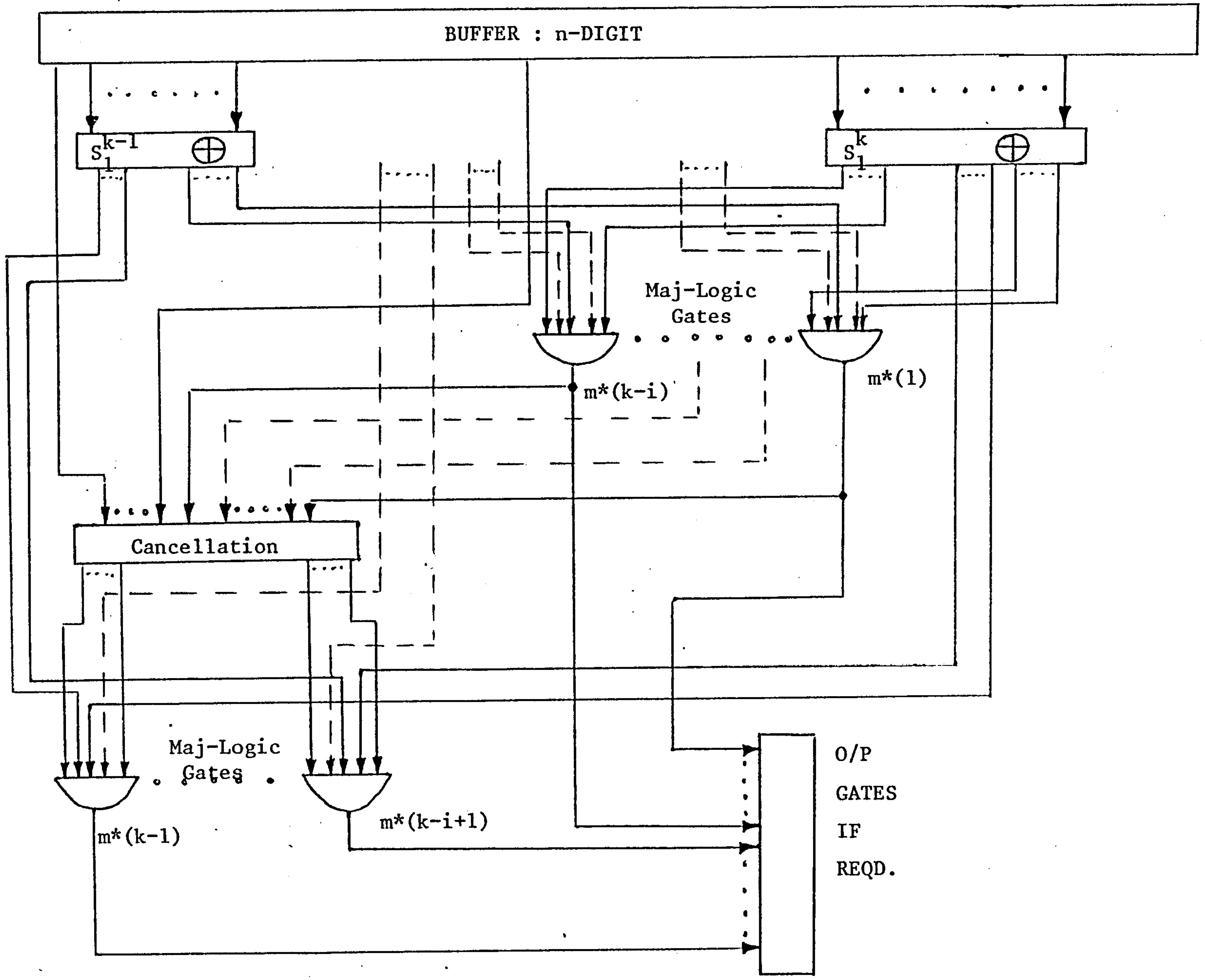


FIG. 5.3.1.

A GENERAL DECODER.

generator matrix are arranged such that the first k'-i digits of the columns are circulants of k'-i, tuples, then the first k'-i message-digits can be decoded cyclically. Once this is done special provision must be made for decoding the remaining i+1 message-digits. Alternatively one may decode using k majority-logic gates, and use the decoded estimates of the first k'-i digits to cancel their effect before decoding the final i+1 digits. A general decoder of this type is shown in figure 5.3.1.

Once the effect of the first k'-i digits have been cancelled from the generator matrix, there remains $2^{k'}$ columns, of G, with parity-checks on each of the remaining i+1 message-digits.

Obviously there are $2^{k'}$ columns with a parity-check on digit k in the subset $S_1^{k'}$. There are $2^{k'-1}$ columns in subset $S_1^{k'}$ parity-checking digit k' and $2^{k'-1}$ columns in subset $S_1^{k'-1}$ parity-checking digit k', giving a total of $2^{k'}$. Eventually, for digit k'-i-1, the number of columns giving a parity-check are, $2^{k'-1}$ from $S_1^{k'}$, $2^{k'-2}$ from $S_1^{k'-1}$, $2^{k'-3}$ from $S^{k'-2}$, ..., $2^{k'-i}$ from $S_1^{k'-i}$. This sums to $2^{k'}$ and the information is available to obtain the necessary $J < 2^{k'}$ orthogonal check sums, of equation 5.3.3. **after cancellation.**

We will call this construction 5.3.a. outlined as below.

Construction 5.3.a.

$$S = S_1^{k'} \cup S_1^{k'-1} \cup \ldots\ldots \cup S_1^{k'-i}$$

$$n = 2^{k'} + 2^{k'-1} + \ldots\ldots + 2^{k'-i}$$

$$J = 2^{k'-1} + 2^{k'-2} + \ldots\ldots + 2^{k'-i-1}$$

k = (k'+1) information digits.

Decode digits (1,2,..., k'-i), then (k-i+1,..., k). If we examine message-digit m(1) there are columns parity-checking this digit as follows,

$2^{k'-1}$ from $S_1^{k'}$, $2^{k'-2}$ from $S_1^{k'-1}$, down to $2^{k'-i-1}$ from $S_1^{k'-i}$.

Thus there exists a code word of binary weight, w(1), where

$$w(1) = 2^{k'-1} + 2^{k'-2} + \ldots\ldots + 2^{k'-i-1} = J$$

and since,

$$w(1) \geq d_m \geq J \quad , \quad d_m = J.$$

Therefore, $n = 2d_m$ for this construction.

Consider a code whose generator matrix contains the set,

$$S = S_1^{k'} \cup S_1^{k'-1} \cup \ldots\ldots\ldots \cup S_1^{*k'-i},$$

where $S_1^{*k'-i}$ is a subset of the set $S_1^{k'-i}$, and which is a code for

X k = k'-i+1. Let $n_{k'-i}$ and $J_{k'-i}$ be the length of the subset $S_1^{*k'-i}$ and the number of check sums respectively, including redundancy, obtainable on the first k'-i digits, from the subset $S_1^{*k'-i}$.

Then,

$$\begin{aligned} n &= 2^{k'} + 2^{k'-1} + \ldots\ldots + n_{k'-i} \\ J &= 2^{k'-1} + 2^{k'-2} + \ldots\ldots + J_{k'-i} \end{aligned} \qquad 5.3.4.$$

where again J is the total number of check sums obtainable on the first k'-i message-digits. After decoding and cancelling their effect, there are enough parity-checks on the remaining i+1 digits to form J check sums. Since the set,

$$S = S_1^{k'} \cup S_1^{k'-1} \cup \ldots\ldots \cup S_1^{k'-i+1}$$

has minimum distance

$$d'_m = 2^{k'-1} + 2^{k'-2} + \ldots\ldots + 2^{k'-i} \qquad 5.3.5.$$

then from equation 5.3.4. and 5.3.5.

$$J = d'_m + J_{k'-i}$$

but $J_{k'-i} = d_{m,k'-i}$ and

$$n_{k'-i} = 2d_{m,k-i} + Rd(b) - m$$

and so

$$n = 2d'_m + 2d_{m,k'-i} + Rd(b) - m$$

$$J = d_m + d_{m,k-i} \ .$$

We call this construction 5.3.b. outlined below.

Construction 5.3.b.

$$S = S_1^{k'} \cup S_1^{k'-1} \cup \ldots\ldots \cup S_1^{*k'-i}$$

$$n = 2^{k'} + 2^{k'-1} + \ldots\ldots + 2d_{m,k'-i} + Rd(b) - m$$

$$J = 2^{k'-1} + 2^{k'-2} + \ldots\ldots + d_{m,k'-i}$$

$k = (k'+1)$ information digits.

Decode digits $(1,2,\ldots, k'-i)$ then $(k'-i+1,\ldots, k)$.

A list of constructions is given in Table 5.3.1. and is preceeded by the notation (n,J) which denotes the code used as the subset S_1^{*k-i} from section 4.5.

Although construction 5.3.a. is a special case of 5.3.b. when $S_1^{*k'-i} = S_1^{k'-i}$, they have been presented separately because construction 5.3.a. is a class of codes with zero redundancy which can be constructed without reference to the constructions in Chapter 4.

From equations 5.3.1. we saw that if the complete set of k'-tuples $S_1^{k'}$ is used to form a code then we can obtain J check sums, where,

$$J = \sum_{b=0}^{k'-1} \frac{(k'-1)!}{b!\,(k-b)!} = 2^{k'-1}$$

Of course this only applies to the first k' message-digits, the k'th message-digit is decoded specially. However if the overall parity-check on message-digit m(k) is removed, we can still obtain J check sums on the k' message-digits, The set of k'-tuples without overall parity-check, $S^{k'}$, therefore has $J = 2^{k'-1}$. With this piece of information we can propose another construction.

Construction 5.3.c.

$$S = mS^{k} \cup S_1^{k-1} \cup S_1^{k-2} \cup \ldots . \cup S_1^{k-i}$$

$$n = m.2^{k} + 2^{k-1} + 2^{k-2} + \ldots . + 2^{k-i}$$

$$J = m.2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots . + 2^{k-i-1}$$

k information digits.

Decode digits (1,2,..., k-i) then (k-i+1,..., k).

m = 1,2,3,..........

In this way the generator matrix may be extended indefinitely.

Again, as in construction 5.3.a., there is a code word of weight J, therefore $J = d_m$ so that $n = 2d_m$.

One can generalize the construction by replacing S_1^{k-i} by S_1^{*k-i}, with results similar to construction 5.3.b.

For all constructions considered cyclic decoding is not too difficult providing i+1 is small.

TABLE 5.3.1.

Codes constructed using constructions 5.3.a. and b. are denoted by the letters 'a' and 'b' respectively.

$S = \underline{S_1^6 \cup S_1^{*5}}$; $\underline{k = 7}$; Decode $(1,2,\ldots,5) \longrightarrow (6,7)$

(0,0)	n = 64	d_m = 32	'a'	$\overset{40}{d_m}$ = 32
(6,1)	70	33	'b'	33^F
(7,2)	71	34	'b'	34
(11,3)	75	35	'b'	35-36
(12,4)	76	36	'b'	36^D
(15,5)	79	37	'b'	36-38
(16,6)	80	38	'b'	36-39
(20,8)	84	40	'b'	40^D
(22,9)	86	41	'b'	40-42
(23,10)	87	42	'b'	40-42
(26,11)	90	43	'b'	43-44
(27,12)	91	44	'b'	44^C
(31,15)	95	47	'b'	47
(32,16)	96	48	'a'	48^D

$S = \underline{S_1^6 \cup S_1^5 \cup S_1^{*4}}$; $\underline{k = 7}$; Decode $(1,2,3,4) \longrightarrow (5,6,7)$

(0,0)	96	48	'a'	48^D
(5,1)	101	49	'b'	49^F
(6,2)	102	50	'b'	50
(9,3)	105	51	'b'	51-52
(10,4)	106	52	'b'	52^D
(13,5)	109	53	'b'	53-54
(14,6)	110	54	'b'	54
(15,7)	111	55	'b'	55
(16,8)	112	56	'a'	56^D

$S = \underline{S_1^6 \cup S_1^5 \cup S_1^4 \cup S_1^{*3}}$; $\underline{k = 7}$; Decode $(1,2,3) \longrightarrow (4,5,6,7)$

(0,0)	112	56	'a'	56^D
(4,1)	116	57	'b'	57^F
(5,2)	117	58	'b'	58
(7,3)	119	59	'b'	59^F
(8,4)	120	60	'a'	60

$S = \underline{S_1^6 \cup S_1^5 \cup S_1^4 \cup S_1^3 \cup S_1^{*2}}$; $\underline{k = 7}$; Decode $(1,2) \longrightarrow (3,\ldots\ldots,7)$

(0,0)	120	60	'a'	60
(3,1)	123	61	'b'	62
(4,2)	124	62	'a'	62^D

$S = \underline{S_1^6 \cup S_1^5 \cup S_1^4 \cup S_1^3 \cup S_1^{*2} \cup S_1^1}$; $\underline{k = 7}$; Decode $(1,2,3,\ldots\ldots,7)$

(0,0)	124	62	'a'	62^D
(2,1)	126	63	'a'	63

This completes the codes for $k = 7$, $n \leqslant 2^7$.

$S = \underline{S_1^5 \cup S_1^{*4}}$; $\underline{k = 6}$; Decode $(1,2,3,4) \longrightarrow (5,6)$

(0,0)	32	16	'a'	16
(5,1)	37	17	'b'	17^F
(6,2)	38	18	'b'	18
(9,3)	41	19	'b'	19-20
(10,4)	42	20	'b'	20^D
(13,5)	45	21	'b'	21-22
(14,6)	46	22	'b'	22
(15,7)	47	23	'b'	23
(16,8)	48	24	'a'	24^D

$S = \underline{S_1^5 \cup S_1^4 \cup S_1^{*3}}$; $\underline{k = 6}$; Decode $(1,2,3) \longrightarrow (4,5,6)$

(0,0)	48	24	'a'	24^D
(4,1)	52	25	'b'	25^F
(5,2)	53	26	'b'	26
(7,3)	55	27	'b'	27
(8,4)	56	28	'a'	28^D

$S = \underline{S_1^5 \cup S_1^4 \cup S_1^3 \cup S_1^{*2}}$; $\underline{k = 6}$; Decode $(1,2) \longrightarrow (3,\ldots,6)$

(0,0)	56	28	'a'	28^D
(3,1)	59	29	'b'	29
(4,2)	60	30	'a'	30^D

$S = \underline{S_1^5 \cup S_1^4 \cup S_1^3 \cup S_1^2 \cup S_1^1}$; $\underline{k = 6}$; Decode $(1,2,\ldots,6)$

(0,0)	60	30	'a'	30^D
(2,1)	62	31	'a'	31

This completes the codes for $k = 6$, $n \leq 2^6$.

$S = \underline{S_1^4 \cup S_1^{*3}}$; $\underline{k = 5}$; Decode $(1,2,3) \longrightarrow (4,5)$

(0,0)	16	8	'a'	8
(4,1)	20	9	'b'	9^F
(5,2)	21	10	'b'	10
(7,3)	23	11	'b'	11
(8,4)	24	12	'a'	12^D

$S = \underline{S_1^4 \cup S_1^3 \cup S_1^{*2}}$; $\underline{k = 5}$; Decode $(1,2) \longrightarrow (3,4,5)$

(0,0)	24	12	'a'	12^D
(3,1)	27	13	'b'	13
(4,2)	28	14	'a'	14^D

$S = \underline{S_1^4 \cup S_1^3 \cup S_1^2 \cup S_1^1}$; $\underline{k = 5}$; Decode (1,2,3,4,5)

(0,0)	28	14	'a'	14^D
(2,1)	30	15	'a'	15

This completes the codes for $k = 5$, $n \leqslant 2^5$.

$S = \underline{S_1^3 \cup S_1^{*2}}$; $\underline{k = 4}$; Decode $(1,2) \longrightarrow (3)$

(0,0)	8	4	'a'	4
(3,1)	11	5	'b'	5^B
(4,2)	12	6	'a'	6

$S = \underline{S_1^3 \cup S_1^2 \cup S_1^1}$; $\underline{k = 4}$; Decode (1,2,3,4)

(0,0)	12	6	'a'	6
(2,1)	14	7	'a'	7

This completes the codes for $k = 4$, $n \leqslant 2^4$.

$S = \underline{S_1^2 \cup S_1^1}$; $\underline{k = 3}$; Decode (1,2,3)

(0,0)	4	2	'a'	2
(2,1)	6	3	'a'	3

And this completes the codes we can construct for $k \leqslant 7$, $n \leqslant 2^k$, using constructions 5.3.a and 5.3.b.

CHAPTER 6

6. NOTES ON THE MINIMUM DISTANCE OF GROUPS OF BINARY k-TUPLES.

6.1 Introduction.

We develop an expression for the minimum Hamming distance of a complete set of k-tuples of weight x, $0 \leq x \leq k$. We then show how to build up tables of minimum distances for binary k-tuple sets for any k. For large k even, it is seen to be not too difficult a task.

The minimum distance of codes, whose generator matrix is composed of complete sets of k-tuples of various weights, can be found from the tables. This introduces codes not seen in the previous sections and for which the author has not been able to discover a decoding procedure.

6.2 Tables of Minimum-distance.

There are 2^k distinct binary k-tuples which as a set, can be divided into groups of binary weight x, $0 \leq x \leq k$.

Consider the group of k-tuples of weight x arranged in a column. This column can be considered as k single digit columns so that each of the k columns can be considered as an N_x^k-tuple, where

$$N_x^k = \frac{k!}{x!\,(k-x)!} \quad .$$

Since all weight x k-tuples comprise the k columns, the binary weight of a single column, $W(N_x^k)$, is equal to the number of (k-1)-tuples whose binary weight is (x-1). Therefore,

$$W(N_x^k) = \frac{(k-1)!}{(x-1)!\,(k-x)!} \qquad 6.2.1.$$

If we consider all linear modulo 2 combinations of the k columns as a code space, then each column will form a row in the generator matrix of the code.

We can confine our attention for the moment to those code words resulting from the linear combinations of 'a' rows of such a generator matrix. In such a situation each k-tuple in the set can be divided into an 'a'-tuple and (k-a)-tuple. Since the set of all x weight k-tuples are present, each a-tuple of weight n must associate with all (k-a)-tuples of weight (x-n). Since there are,

$$N_n^a = \frac{a!}{n!\ (a-n)!}$$

a-tuples of weight n, then corresponding to each there are,

$$N_{x-n}^{k-a} = \frac{(k-a)!}{(x-n)\,(k-a-x+n)!} \qquad 6.2.2.$$

(k-a)-tuples of weight x-n. However if n is even the a-tuple will generate a code word digit of binary zero. Thus the binary weight of an N_x^k-tuple which is the linear combination of <u>ANY</u> 'a' columns, $W_x^k(a)$, is given by,

$$W_x^k(a) = \sum_{n=1}^{a} N_n^a \cdot N_{x-n}^{k-a} \cdot \frac{(1-(-1)^n)}{2} \quad , \qquad 6.2.3.$$

which is the number of odd-weight 'a'-tuples. Thus any code words that are the linear combination of 'a' rows of the generator matrix, have the same weight. We know that the minimum distance of a binary code is equal to the minimum binary weight of its code words, and therefore,

$$d_m = \min\ (W_x^k(a), \quad 1 \leqslant a \leqslant k).$$

Immediately we can see that, in equation 6.2.3.

i) if x is odd $\overset{\text{and}}{\wedge}$ a = k, then n = x only, and

$$W_x^k(k) = N_x^k = d_m(\max).$$

ii) if x is even and a = k, then n = x only, but

$$W_x^k(k) = 0 = d_m.$$

Therefore if x is even,

$$d_m = 0$$

and sets like this have no use as code spaces, when used alone.

Consider the codes whose generator matrix comprises the sets of k-tuples of weights x and x-1, then

$$d_m \geqslant \min\,(W_x^k(a)) + \min\,(W_{x-1}^k(a)) \qquad 1 \leqslant a \leqslant k$$

$$= \min\left(\sum_{n=1}^{a} N_n^a\, N_{x-n}^{k-a}\, \frac{(1-(-1)^n)}{2} + \sum_{n=1}^{a} N_n^a\, N_{x-1-n}^{k-a}\, \frac{(1-(-1)^n)}{2} \right)$$

$$= \min\left(\sum_{n=1}^{a} N_n^a \left\{ N_{x-n}^{k-a} + N_{x-1-n}^{k-a} \right\} \frac{(1-(-1)^n)}{2} \right) \qquad 6.2.4.$$

However,

$$N_{x-1-n}^{k-a} = N_{x-n}^{k-a} \cdot \frac{(x-n)}{(k-a-x+n+1)}$$

therefore,

$$N_{x-n}^{k-a} + N_{x-1-n}^{k-a} = N_{x-n}^{k-a}\left(1 + \frac{(x-n)}{k-a-x+n+1}\right)$$

but

$$N_{x-n}^{k-a} = N_{x-n}^{k+1-a} \cdot \frac{(k+1-a-x+n)}{(k+1-a)}$$

therefore,

$$N_{x-n}^{k-a} + N_{x-1-n}^{k-a} = N_{x-n}^{k+1-a}\left(\frac{k+1-a-x+n}{k+1-a} + \frac{x-n}{k+1-a}\right)$$

$$= N_{x-n}^{k+1-a}$$

and therefore, equation 6.2.4. becomes,

$$d_m = \min \left\{ \sum_{n=1}^{a} N_n^a \, N_{x-n}^{k+1-a} \quad \frac{(1-(-1)^n)}{2} \right\}$$

$1 \leqslant a \leqslant k.$

But this is the expression for the d_m of the set of (k+1)-tuples of weight x, except that $1 \leqslant a \leqslant (k+1)$. Thus if 'a' < (k+1), for a given 'a' the two systems generate code words of the same weight. If x is even, the system of (k+1)-tuples has $d_m = 0$, otherwise the two systems have the same minimum distance, since

$$W_x^k(k+1) = N_x^{k+1} = d_{m(max)}.$$

The above result has another significance, this being,

$$W_x^k(a) + W_{x-1}^k(a) = W_x^{k+1}(a) \qquad 6.2.5.$$

for $1 \leqslant a \leqslant k$.

Thus having constructed a table of weights for the k-tuple sets, of weights x, for various values of 'a', one can construct the table of weights for the (k+1)-tuple sets, of weights x, for the same values of 'a', up to a = k, x = k. The table for the (k+1)-tuples is then completed by the relationships,

$$\begin{aligned} W_{k+1}^{k+1}(a) &= 0 \quad \text{if} \quad a = \text{even}. \\ &= 1 \quad \text{if} \quad a = \text{odd}. \qquad 6.2.6. \\ W_x^{k+1}(k+1) &= N_x^{k+1} \quad \text{if} \quad x = \text{odd}. \\ &= 0 \quad \text{if} \quad x = \text{even}. \end{aligned}$$

Let the tables take the form below.

$$\begin{array}{c|cccc}
a \backslash x & 0 & 1 & \cdots\cdots & k \\
\hline
0 & 0 & 0 & \cdots\cdots & 0 \\
1 & 0 & & & \\
\vdots & \vdots & & W_x^k(a) & \\
k & 0 & & &
\end{array}$$

For 'a' > 0, let a row of the table be represented by $W^k(a)$, where,

$$W^k(a) = (W_0^k(a),\ W_1^k(a), \ldots\ldots,\ W_k^k(a)\ )$$

and let,

$$N^{k-a} = \left\{N^{k-a}_{0-n},\ N^{k-a}_{1-n}, \ldots\ldots,\ N^{k-a}_{0},\ N^{k-a}_{1}, \ldots\ldots\ldots \right.$$

$$\left. \ldots\ldots,\ N^{k-a}_{k-a},\ N^{k-a}_{k-a+1}, \ldots\ldots,\ N^{k-a}_{k} \right\}$$

but if,

$$x < n\ ,\ N^{k-a}_{x-n} = 0 \qquad 6.2.7.$$

$$x > k-a+n,\ N^{k-a}_{x-n} = 0$$

therefore,

$$N^{k-a} = (\overbrace{0,0,\ldots,0,}^{n}\ N^{k-a}_{0},\ N^{k-a}_{1}, \ldots,\ N^{k-a}_{k-a},\ \overbrace{0.\ldots\ldots.0}^{(a-n)}\ )$$

and the centre section can be seen to be the coefficients of the binomial expansion of $(y+1)^{k-a}$, where y is an arbitrary unknown.

Thus a row of the table can be represented by the expression below.

$$W^k(a) = \sum_{n=1}^{a} \left\{ N^a_n \left\{ \overbrace{0,\ldots.,0,}^{n}\ \text{b.e.c.}(y+1)^{k-a},\ \overbrace{0,\ldots,0}^{(a-n)} \right\} \frac{(1-(-1)^n)}{2} \right\} \qquad 6.2.8.$$

Where b.e.c. $(y+1)^{k-a}$ means, the binomial expansion coefficients of $(y+1)^{k-a}$.

Two examples of constructing the tables follow.

<u>Example 6.2.1.</u>

Let k = 2, then

i) if a = 1, from equation 6.2.8.

$$W^2(1) = N_1^1\left[0,\ \text{b.e.c.}(y+1)^{k-1}\right]$$

$$= 1\ (0,1,1) = (0,1,1)$$

which is the first row of the table.

ii) if a = 2

$$W^2(2) = N_1^2\left[0,\ \text{b.e.c.}(y+1)^{o},0\right]$$

$$= 2\ (0,1,0) = (0,2,0)$$

which is the second row of the table.

a \ x	0	1	2
0	0	0	0
1	0	1	1
2	0	2	0 .

<u>Example 6.2.2.</u>

Let k = 3, then

i) if a = 1, from equation 6.2.8.

$$W^3(1) = N_1^1\left[0,\ \text{b.e.c.}(y+1)^2\right]$$

$$= 1\ (0,1,2,1) = (0,1,2,1)$$

ii) if a = 2

$$W^3(2) = N_1^2 \left[0,\ \text{b.e.c.}(y+1)^1, 0\right]$$

$$= 2\ (0,1,1,0) = (0,2,2,0)$$

iii) if a = 3

$$W^3(3) = N_1^3 \left[0,\ \text{b.e.c.}(y+1)^0, 0, 0\right] + N_3^3 \left[0,0,0,\ \text{b.e.c.}(y+1)^0\right]$$

$$= 3\ (0,1,0,0) + 1\ (0,0,0,1)$$

$$= (0,3,0,0) + (0,0,0,1)$$

$$= (0,3,0,1).$$

giving the table

a \ x	0	1	2	3
0	0	0	0	0
1	0	1	2	1
2	0	2	2	0
3	0	3	0	1

Using the information so far presented the tables up to k = 10, have been constructed and are given as Table 6.2.1.

TABLE 6.2.1.

k = 2.

		$\underline{x}$		
	$W_x^k(a)$	0	1	2
	0	0	0	0
a)	1	0	1	1
	2	0	2	0
	N_x^2	1	2	1

k = 3.

		$\underline{x}$			
	$W_x^k(a)$	0	1	2	3
	0	0	0	0	0
a)	1	0	1	2	1
	2	0	2	2	0
	3	0	3	0	1
	N_x^3	1	3	3	1

k = 4.

		$\underline{x}$				
	$W_x^k(a)$	0	1	2	3	4
	0	0	0	0	0	0
	1	0	1	3	3	1
a)	2	0	2	4	2	0
	3	0	3	3	1	1
	4	0	4	0	4	0
	N_x^4	1	4	6	4	1

k = 5.

		$\underline{x}$					
	$W_x^k(a)$	0	1	2	3	4	5
	0	0	0	0	0	0	0
	1	0	1	4	6	4	1
a)	2	0	2	6	6	2	0
	3	0	3	6	4	2	1
	4	0	4	4	4	4	0
	5	0	5	0	10	0	1
	N_x^5	1	5	10	10	5	1

k = 6.

		$\underline{x}$						
	$W_x^k(a)$	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
	1	0	1	5	10	10	5	1
	2	0	2	8	12	8	2	0
a)	3	0	3	9	10	6	3	1
	4	0	4	8	8	8	4	0
	5	0	5	5	10	10	1	1
	6	0	6	0	20	0	6	0
	N_x^6	1	6	15	20	15	6	1

k = 7.

		$\underline{x}$							
	$W_x^k(a)$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
	1	0	1	6	15	20	15	6	1
	2	0	2	10	20	20	10	2	0
a)	3	0	3	12	19	16	9	4	1
	4	0	4	12	16	16	12	4	0
	5	0	5	10	15	20	11	2	1
	6	0	6	6	20	20	6	6	0
	7	0	7	0	35	0	21	0	1
	N_x^7	1	7	21	35	35	21	7	1

k = 8.

					x					
	$W_x^k(a)$	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
	1	0	1	7	21	35	35	21	7	1
	2	0	2	12	30	40	30	12	2	0
	3	0	3	15	31	35	25	13	5	1
a	4	0	4	16	28	32	28	16	4	0
	5	0	5	15	25	35	31	13	3	1
	6	0	6	12	26	40	26	12	6	0
	7	0	7	7	35	35	21	21	1	1
	8	0	8	0	56	0	56	0	8	0
	N_x^8	1	8	28	56	70	56	28	8	1

k = 9.

					x						
	W	0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	8	28	56	70	56	28	8	1
	2	0	2	14	42	70	70	42	14	2	0
	3	0	3	18	46	66	60	38	18	6	1
a	4	0	4	20	44	60	60	44	20	4	0
	5	0	5	20	40	60	66	44	16	4	1
	6	0	6	18	38	66	66	38	18	6	0
	7	0	7	14	42	70	56	42	22	2	1
	8	0	8	8	56	56	56	56	8	8	0
	9	0	9	0	84	0	126	0	36	0	1
	N_x^9	1	9	36	84	126	126	84	36	9	1

<u>k = 10</u>.

		x										
	W	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	9	36	84	126	126	84	36	9	1
	2	0	2	16	56	112	140	112	56	16	2	0
	3	0	3	21	64	110	126	98	56	24	7	1
a	4	0	4	24	64	104	120	104	64	24	4	0
	5	0	5	25	60	100	126	110	60	20	5	1
	6	0	6	24	56	104	132	104	56	24	6	0
	7	0	7	21	56	112	126	98	64	24	3	1
	8	0	8	16	64	112	112	112	64	16	8	0
	9	0	9	9	84	84	126	126	36	36	1	1
	10	0	10	0	120	0	252	0	120	0	10	0
	N_x^{10}	1	10	45	120	210	252	210	120	45	10	1

6.3 The Codes.

From Table 6.2.1. we can form codes by using the sets of weight x, x = odd, as the k-tuple columns of a generator matrix. The table column corresponding to x gives the different code word weights possible so that d_m is the minimum figure in a column. The table can also be used to find the weight enumerators of the code. There are k!/(a! . (k-a)!) code words of binary weight $W_x^k(a)$. One can also form codes by combining the weights of different columns. If we combine b columns, of weights $x_1, x_2, \ldots, x_b$, we can specify the generator matrix by the set $(x) = (x_1, x_2, \ldots, x_b)$, so that the resulting column of weights will be the values $W_{x_1}^k(a) + W_{x_2}^k(a) + \ldots + W_{x_b}^k(a)$ for $1 \leq a \leq k$. For each such value of resultant weight, there are again k!/a! (k-a)! code words, of this weight.

Table 6.3.1. below lists those codes, from the Table 6.2.1., for k = 7, whose lengths, n, are the shortest for the given d_m. Those codes marked with an asterisk are quasi-cyclic. They are compared with d_m from Hel gert and Stinaff.[40]

TABLE 6.3.1.

TABLE OF OPTIMUM CODES FOR k = 7

n		d_m	$d_{m_1}^{40}$	(x)
7	*	1	1	(1)
8		2	2	(1,7)
14	*	4	4	(1,6)
21	*	6	8	(5)
28	*	7	12	(1,2)
29		8	12	(1,2,7)
28	*	12	12	(1,5) or (5,6)
35	*	14	15-16	(1,5,6)
35	*	15	15-16	(3)
36		16	16^D	(3,7)
42	*	17	17-19	(3,6)
43		18	18-20	(3,6,7)
49	*	22	23	(1,3,6)
50		23	24	(1,3,6,7)
56	*	24	24-27	(1,2,5,6)
56	*	26	24-27	(3,5)
63	*	28	31	(1,4,5)
63	*	31	31^I	(1,3,5)
64		32	32	(1,3,5,7)
70	*	33	33^F	(1,3,5,6)
71		34	34	(1,3,5,6,7)
77	*	35	36-37	(3,4,6)
77	*	36	36-37	(1,3,4)
84	*	37	40	(1,2,3,5)
85		38	40-41	(1,2,3,5,7)
84	*	40	40^D	(1,3,4,6)
98	*	42	48	(1,2,3,4)
91	*	44	44^C	(3,4,5)
98	*	48	48	(3,4,5,6)
105	*	51	51-52	(1,3,4,5,6)
106		52	52^D	(1,3,4,5,6,7)
119	*	57	59	(1,2,3,4,5)
120		58	60	(1,2,3,4,5,7)
126	*	63	63	(1,2,3,4,5,6)
127		64	64	(1,2,3,4,5,6,7)

In theorem 4.3.3. we saw that when k is a prime the set of all k-tuples can be organised as circulants of order k, except the two trivial circulants which have order one.

We also know that a circulant set of k-tuples is comprised of k-tuples of the same weight. Thus the x-weight groups can be organised as sets of circulants.

Therefore the codes in this section, when k is a prime, are quasi-cyclic of order k if $(0), (k) \notin (x)$.

Hence the large number of quasi-cyclic codes of order 7 in Table 6.3.1.

The codes not given in Table 6.3.1. are,

i) $(x) = (x_1)$, x_1 = even and $d_m = 0$

ii) $(x) = (x_1, (x_1 - 1))$, x_1 = odd, since a code of the same length and d_m exists with $(k+1)$ message-digits.

If $(1) \in (x)$, then the codes can be arranged in systematic form.

An interesting point is that knowing the weight enumerators, we know all code word weights, thus if an overall parity-check is added, to each column of G, on a (k+1)'th digit we can find the resulting minimum distance.

We simply require to know, (if $G = (x_1)$) $\max (W^k_{x_1}(a))$ then we have,

$$d^{k+1}_m = N^k_{x_1} - \max (W^k_{x_1}(a)) \qquad 6.3.1.$$

But since,

$$W^k_{x_1}(k) = N^k_{x_1} = \max (W^k_{x_1}(a))$$

when x_1 is odd,

$$d^{k+1}_m = 0.$$

Also as seen earlier,

$$W^k_{x_1}(k) = 0 = d_m$$

when x_1 is even. So that adding an overall parity-check on a (k+1)'th digit is only useful if,

$$(x) = (x_1, x_2, \ldots, x_b)$$

$b > 1$ and all x_i not odd.

Let, $(x) = (x_1, x_2, \ldots, x_b)$, where $b \leq k$ and the x_i are not all odd, and let max $(w^k_{(x)}(a))$ be the maximum weight of any code word in the code space generated by (x). Let d^k_m and $N^k_{(x)}$ be the minimum distance and length of any code word from the code space, respectively.

Then if an overall parity-check is added on all columns of the generator matrix, on a (k+1)'th digit, the minimum distance d^{k+1}_m is given by,

$$\text{Min}\left[d^k_m\,,\ N^k_{(x)} - \max\,(W^k_{(x)}(a))\right] \qquad 6.3.2.$$

The following example illustrates, this idea for the case $k = 7$, $x = (3,4)$.

Also Table 6.3.2. gives codes with $k = 8$ developed from Tables 6.2.1. and 6.3.1., by adding an overall parity-check.

Only codes where

$$N^k_{(x)} - \max\,(W^k_{(x)}(a)) \geq (d^k_m - 1)$$

are shown.

TABLE 6.3.2.

CODES WITH k = 8 (OVERALL PARITY-CHECK).

n	d_m	d_m^{40}	(x)
28	7	11-12	(1,2)
29	8	12	(1,2,7)
56	24	24-26	(1,2,5,6)
63	27	28-30	(1,4,5)
98	42	46-48	(1,2,3,4)
119	56	56-58	(2,3,4,5,6)
126	62	62	(1,2,3,4,5,6)

Example 6.3.1.

Let (x) = (3,4) and k = 7., then from Tables 6.2.1.

$$W_3^7(1) + W_4^7(1) = 15 + 20 = 35$$

$$W_3^7(2) + W_4^7(2) = 20 + 20 = 40$$

$$W_3^7(3) + W_4^7(3) = 19 + 16 = 35$$

$$W_3^7(4) + W_4^7(4) = 16 + 16 = 32$$

$$W_3^7(5) + W_4^7(5) = 15 + 20 = 35$$

$$W_3^7(6) + W_4^7(6) = 20 + 20 = 40$$

$$W_3^7(7) + W_4^7(7) = 35 + 0 = 35 .$$

The code has length,

$$n = N_3^7 + N_4^7 = 35 + 35 = 70.$$

From the set of code word weights we have,

$$d_m^k = 32,$$

$$\max\ (W_{(3,4)}^7(a)) = 40.$$

Therefore if we add an extra message-digit and an overall parity-check, the minimum distance becomes,

$$d_m^{k+1} = 70 - 40 = 30$$

since $30 < d_m^k = 32$, from equation 6.3.2.

Chapter 7

7. CONCLUSIONS AND DISCUSSIONS.

7.1 "Good" Codes.

Before assessing the codes presented in the previous chapters it is worth discussing what determines if a code is "good" or not. There is, in effect, a theoretical form of assessment of "good" and a practical one, both of which are helpful in assessing the usefulness of a class of codes.

The theoretical criteria for a "good" code rests upon a statement which says that, for a given n and k there always exists an (n,k) code with minimum distance at least d'_m. A statement of this form is called a lower bound and the Varsharmov-Gilbert lower bound is widely used as a criteria for "good" codes. Basically this lower bound states that it is possible to find an (n,k) code with minimum distance at least d'_m for which the following inequality holds,

$$H\left(\frac{d'_m - 2}{n}\right) \geqslant 1 - R \qquad 7.1.1.$$

where;

$$H(x) = -x \log_2(x) - (1-x) \log_2(1-x)$$

= the entropy function.

$R = k/n$

= the code transmission rate.

An individual code then is classed as "good" if its parameters (n,k,d_m) satisfy inequality 7.1.1., since compared to the (n,k,d'_m) code guaranteed by the bound, then $d_m \geqslant d'_m$ for the same n and k.

This criteria is used to form an assessment of the performance of the codes in Chapter 4 for increasing k.

Though the Varsharmov-Gilbert lower bound can tell us if a code is "good" it does not tell us how "good" it is. Statements that indicate the best that can be achieved with codes, under certain assumptions, are called upper bounds. Typical upper bounds are the Plotkin upper bound, Hamming upper bound and Elias upper bound. For a simple discussion of their development, see Peterson and Weldon[2], Chapter 4.

A table of maximum minimum distances implied as possible, by all upper bounds (known at the time), was published by Hel gert and Stinaff[40] for all $n \geqslant 127$ and $k \geqslant 127$.

This table was used for comparison of minimum distances, for the codes developed in the previous chapters, as shown in Tables 4.7.1., 5.2.1., 5.3.1., 6.3.1. and 6.3.2.

Upper and lower bounds are very useful for indicating what is and is not possible in coding theory. Also of course they are useful for indicating how theoretically "good" a given class of codes is. However bounds do not, in general, show how one should construct the codes that they infer exist. Therefore, in practice many of the theoretical codes are yet to be discovered, so that a given code, though not as "good" as the upper bound, may be the best constructive code with these parameters and is therefore "good" from a practical viewpoint. Also one must consider decoder complexity when assessing how practically "good" a code or class of codes is. If a number of codes exist with similar values of n,k and d_m then the best code may be that which is simplest to decode. Therefore a code may be "good" because it is simple to decode compared with other codes of a similar or better capability particularly if a slight degradation of performance is acceptable on the basis of encoder-decoder implementation economics. Of course as mentioned in Chapter 1 the minimum distance of a code is a crude measure of its performance on a real channel. Nevertheless it is still useful to compare codes on the basis of rate, length, d_m and decoder complexity.

This is done for the codes of tables 4.7.1., 5.2.1., 5.3.1., and 6.3.1., in the following sections. Once a decision has been made regarding choice of decoding method, the best choice of practical code can only be determined from performance tests on a channel.

7.2 Performance of the Class of Binary Codes.

The codes developed in Chapter 4 have been shown to be cyclically decodable in one-step, with a single majority gate, so that in terms of decoder economics, they are very useful as a class. The codes were also shown to be completely orthogonalizable up to their minimum distance so one can deduce that the decoding procedure is efficient.

Those codes constructed are presented in table 4.7.1. where they are compared with the value of minimum distance which the tables of Helgert and Stinaff[40] state are maximum values (known or predicted by upper bounds) for that value of n and k. It can be seen that for these codes developed, with $k \leq 7$, the minimum distance of each (n,k) code is equal to, or slightly less than, the maximum given by the tables[40]. In table 7.2.1. the codes are compared directly with existing codes, where the comparison codes are always shown in their favourable light. The comparison codes are taken from Table 5.2.

TABLE 7.2.1.

Comparison of codes from Table 4.7.1. with other codes.

Codes from Table 4.7.1.			Cyclic Polynomial Codes. (n = odd)		
n	k	d_m	n	k	d_m
7	4	3	7	4	3 *
15	5	7	15	5	7 *
31	6	15	31	6	15 *
27	6	12	27	6	6
20	6	8	21	6	8
15	6	5	15	6	6
63	7	31	63	7	31 *
51	7	23	51	8	24
48	7	21	49	7	7
45	7	19	45	7	15
35	7	14	35	7	14
26	7	10	27	7	6
21	7	7	21	7	8
17	7	5	15	7	5

* These are also Euclidean Geometry codes but decodable in 2-steps, with majority-logic.

Codes from Table 4.7.1.			Quasi-cyclic Codes.		
n	k	d_m	n	k	d_m
8	4	4	8	4	4
10	5	4	10	5	4
11	6	4	12	6	4
13	7	4	14	7	4
15	8	4	16	8	5

			Quasi-perfect Codes.		
n	k	d_m	n	k	t
9	5	3	9	5	1
10	5	4	10	5	1
11	6	4	11	6	1
14	8	3	14	8	1

Quasi-perfect codes, Table 8.3. Quasi-cyclic codes and Appendix.D. List of Binary cyclic codes of odd length $n \leqslant 65$, from Peterson and Weldon[2], pp.122, pp.259 and pp.493-534 respectively. On the basis of these comparisons the codes appear to be very competitive with other codes and this is certainly true for small k. However we are also interested in how the codes perform for large k and some insight can be obtained from figures 4.6.1. and 4.6.2. We can see from these figures that redundancy, for a given k', must be made use of efficiently, if the code is to be useful. For small k' (and therefore k) this is not too difficult, but from figure 4.6.1., when $n = 2^{k'-1} = 65{,}536$, we have $Rd(9) = 12{,}870$ binary 17-tuples to try and combine into further check sums.

From Appendix A, equation A.1.,

$$b = \frac{k'+1}{2} = \frac{17+1}{2} = 9$$

and $x = k' - b + 1 = 9$, so that $x = b$, and the optimum case results in the 12870, k'tuples pairing off into 6,435 forms 0111...110. This is obviously very efficient use of redundancy, but is rather a special case. Generally, as Appendix A shows, most use of redundancy is obtained from the form 1000...001, which is only efficient for

$$b \leqslant \left[\frac{2k'}{3}\right],$$

when k' is large.

This shows that as k' increases the redundancy can only be used efficiently for long codes when b is small. Also as k' rises, we see from figure 4.6.2. that $Rd(b)_{max}$ increases exponentially. Other values of Rd(b), (except Rd(1) which is constant and Rd(2) which is linear, with k' increasing) will have correspondingly non-linear

increases as k' increases.

In fact Rd(3) and Rd(4) are close to $Rd(b)_{max}$ anyway. The results from Appendix A, figures 4.6.2. and 4.6.1. imply that as k' rises good codes will only be obtained with large n relative to $2^{k'-1}$. This is borne out by the Varsharmov-Gilbert lower bound which is shown for the codes with k = 7, 9 and 18, in figure 7.2.1. The value of J used, ignores redundancy, but since each extra check sum requires we add an extra column to G, thus increasing n by one, utilization of redundancy will only give a slight improvement.

As an example, the code mentioned above has n = 65,536, J = 26,333, Rd(9) = 12870, and from very efficient use of redundancy, we can obtain a new code by increasing n and J by 6,435 giving the code, n = 71971, J = 32768. But H(J/n) = H(32,768/41971) < 1-18/71971 and the code still does not qualify as "good".

It can be seen from figure 7.2.1. that for k = 7, H(J/n) > (1-R) and the codes can be considered as "good". However when k = 9, H(J/n) is approximately equal to (1-R) initially, until, as n approaches (2^8-1), the codes become "good". When k = 18, H(J/n) < (1-R) until n approaches $(2^{17}-1)$ when the codes become "good".

Figure 7.2.1. shows that there are some "good" codes to be obtained of all lengths $0 < n \leq 2^{k'}-1$, up to k = 9, but when k > 9, one is only going to obtain "good" codes when n approaches $2^{k'}-1$. This is a result of the distribution of Rd(b) as b increases, shown in figure 4.6.1. and the fact that the use of redundancy to form extra check sums becomes less efficient as k' rises, as Appendix A shows. Nevertheless the codes are useful as a class, particularly for low k.

7.3 Performance of the Codes derived from the Class of Binary Codes.

The codes derived by deleting the k'th digit and the overall parity-check on all columns of G, in section 5.2., are cyclically decodable in one-step of majority-logic using a single majority gate and are therefore economic to decode. The actual codes constructed in section 5.2., are shown in Table 5.2.1. where their minimum distances are compared with d_m from Helgert and Stinaff[40]. In Table 7.3.1. they are also compared with other codes and from these results are seen to compare reasonably well. As pointed out in section 5.2. sometimes codes from this class are better than those from Chapter 4.

For a given k, n and b, a code from section 5.2., ignoring redundancy, has the same value of J as the code from Chapter 4 with the same b, n and k+1.

Thus the curves for H(J/n) in figure 7.2.1. apply to both sets of codes. However the curves of (1-R) for the codes of section 5.2. are slightly different (since there is one less information digit) though not significantly so and are not drawn. Nevertheless there is one aspect of these codes which is significantly different to those of Chapter 4 and arises from use of redundancy.

The results from Appendix A and figures 4.6.1. and 4.6.2. apply equally to the codes of section 5.2. but we do not need to increase length n, to utilize one form of redundancy, that is, the form 000...001.

This may have the effect of raising H(J/n) so that more "good" codes exist. Nevertheless efficient use of this form of redundancy is still subject to the constraints given in Appendix A and as such

TABLE 7.3.1.

Comparison of codes from tables 5.2.1., and 5.3.1., with other codes.

Codes from Table 5.3.1.			Codes from Table 5.2.1.			Cyclic Polynomial Codes (n = odd).		
n	k	d_m	n	k	J	n	k	d_m
14	4	7				15	4	8
30	5	15				31	5	16
24	5	12				25	5	5
21	5	10				21	5	10
62	6	31				63	6	32
48	6	24	48	6	22	49	6	14
45	6	21	43	6	20	45	6	18
			35	6	15	35	6	10
			21	7	8	21	7	8
			28	7	11	27	7	6
			17	6	6	15	6	6
			20	6	8	21	6	8
			27	6	12	27	6	6
			34	6	14	31	6	15
						Quasi-perfect Codes.		
n	k	d_m				n	k	t
11	4	5				11	4	2
						Euclidean Geometry[58] Codes.		
n	k	d_m				n	k	t
48	6	24				48	6	10
						Quasi-cyclic Codes.		
n	k	d_m				n	k	d_m
6	3	3				6	3	3
			10	5	4	10	5	4
			9	4	4	8	4	4
			13	6	4	12	6	4

is limited to codes of long length for large k.

It was mentioned when developing the codes of Section 5.2. that we are no longer sure if $J = d_m$. In fact if we take the code,

$n = 42$, $J = 16$, $\{63,31,15,27,23,21,7,11,13\}$

Utilizing redundancy, we obtain the code,

$n = 42$, $J = 18$, $\{63,31,15,27,23,21,7,11,13\}$

However this code has a generator matrix exactly the same as that code constructed from Table 6.2.1. for k = 6 where, we have

$n = 42$, $d_m = 20$, $(x) = (3,4,5,6)$

so that it is not completely orthogonalizable. Provided J equals the maximum upper bound from Helgert and Stinaff[40], we can be sure the codes are competitive with others. Otherwise if $J < d_m$[40], we are not sure.

The codes presented in construction 5.3.a,b, and c, are not simply decoded, though one has two alternatives. One can use k majority gates and decode by cancellation as shown in the general decoder of figure 5.3.1. Alternatively one can arrange the columns of G as circulants of (k'-i)-tuples and decode the first (k'-i) digits cyclically, making special provision for the remaining (i+1) digits.

If k is small, general decoding may not be so costly though it suffers from a form of error propagation. However provided a correctable error pattern occurs, correct decoding always results. For cyclically decoding the first (k'-i) digits, provided (i+1) or i is small this is quite a useful procedure.

The codes of constructions 5.3.a and 5.3.c are classes of "good" codes since with $n = 2d_m$, $d_m = n/2$ and for large d_m,

$$H\left(\frac{d_m - 2}{n}\right) \approx H\left(\frac{1}{2}\right) = 1\cdot0$$

Since rate, $R > 0\cdot0$, then $(1-R) < 1\cdot0$ and we always have,

$$H\left(\frac{d_m - 2}{n}\right) > (1-R).$$

We can see from table 5.3.1. that constructions 5.3.a always meet the maximum upper bound on minimum distance, for those shown.

The codes of construction 5.3.b are not so easy to assess as they depend upon the amount of redundancy in the final set $S_1^{*k'-i}$. If i is small the redundancy could form a reasonable part of n, whereas if i is large the redundancy will be only a small part of n. Large i implies large n so that again for large k' the best codes will be obtained for large n.

However as Table 5.3.1. shows, for small k' there are some good codes to be obtained for all $0 < n \leqslant 2^k-1$, which are not too difficult to decode.

In Table 7.3.1. some codes from construction 5.3.b are compared with other codes.

7.4 The Weight Tables.

The tables of weights developed in Chapter 6 are useful in two ways. They can be used to construct codes whose minimum distance and weight spectrum can be obtained from the tables, and this is done. The best codes are presented in Table 6.3.1. and their minimum distances are compared with the maximum upper bound from Hel gert and Stinaff[40]. It can be seen that some very useful codes resulted, some better than the codes of Table 4.7.1. but none better than Table 5.3.1., though many are equal.

Secondly they can be used to obtain the minimum distance of some of the codes developed in section 5.2., that is where $(x) = (x_1, x_2, \dots, x_b)$ and $x_b = k$, $x_{b-1} = k-1, \dots, x_1 = k-b+1$, and this defines a consecutive set. This in turn can be used to show that some codes from section 5.2. may not be completely orthogonalizable, as we saw in section 7.3.

However for most of the best codes obtained in Table 6.3.1. the set (x) is not a consecutive one. This implies that the majority-logic procedure if applied would not utilize the full capability of the code, as we saw in the example given in section 7.3.

7.5 Further Work and Comments.

We have developed a majority-logic decoding procedure which involves check sums obtained from linear combinations of received digits or columns of the generator matrix. We then showed a number of different methods of choosing the columns of the generator matrix, such that the necessary check sums could be obtained. We saw that, in general from Chapters 4 and 5, some "good" and useful codes resulted for k up to about 9 but that for large k "good" codes only resulted for large n.

As a decoding procedure there is no reason why it cannot be applied to other codes and the following rate one half quasi-cyclic codes (from Peterson and Weldon, Table 8.3. pp.259) have been found to be majority-logic decodable in this way.

n	k	d_m	Generator of circulant
6	3	3	3
10	5	4	7
12	6	4	7
14	7	4	7

Appendix C shows each generator matrix for these codes and how the check sums are formed. In this appendix it is seen that the two circulants are not of consecutive weights, that is generally we have a circulant generated by a k-tuple of weight 3 and one of weight 1. Thus the decoding procedure is probably more widely applicable and this may be worth further investigation, perhaps in the decoding of the codes developed from the weight tables of Chapter 6. The codes developed from the weight tables in Chapter 6 may be worth further investigation, from the point of view of a decoding procedure and to obtain some bound on minimum distance for large k and n.

Most majority-logic codes either have length 2^x or 2^x-1, see Lin[1]., pp.176-177, Peterson[2], pp.326 and pp.332, Hartmann[16], Lin[20,17], Warren[11], and Lin[1]., pp.151-154.

The codes presented offer a wide range of lengths for a given k.

However the codes presented do not offer the solution to all coding theory problems. They are simple to decode and appear to have good minimum distance properties for small k. Nevertheless they are essentially low-rate codes, $R \leqslant 1/2$, and as such do not compete with the many good Euclidian and Projective Geometry codes, of high rate, which have appeared in the last decade.

APPENDIX A.

From the code (n = 37, k = 18+1) whose generator matrix is $G = \{2^{18}-1,\ 2^{17}-1,\ 2^{16}-1\}$ plus overall parity-check, we obtain the following uses of redundancy.

```
              (k'+1)'th digit.
              1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
              1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
              1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1
    (a)       1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
              -------------------------------------
              0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
              -------------------------------------

              1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    (b)       1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------

              1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
    (c)       1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------

              1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
    (d)       1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
              1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------
              1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
              -------------------------------------
```

$$\text{(e)}\qquad \begin{array}{c} \texttt{1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1} \\ \hline \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1} \\ \hline \end{array}$$

$$\text{(f)}\qquad \begin{array}{c} \texttt{1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1} \\ \hline \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1} \\ \hline \end{array}$$

$$\text{(g)}\qquad \begin{array}{c} \texttt{1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1} \\ \hline \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1} \\ \hline \end{array}$$

$$\text{(h)}\qquad \begin{array}{c} \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1} \\ \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1} \\ \hline \texttt{1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1} \\ \hline \end{array}$$

The original code was;

n	k	d	G
37	19	3	$\{2^{18}-1,\ 2^{17}-1,\ 2^{16}-1\}$

and redundancy gives

36	19	3	$\{2^{17}-1,\ 2^{16}-1,\ -I\}$
37	19	4	$\{2^{17}-1,\ 2^{16}-1,\ -I,\ +0\}$.

Without overall parity-check we combine (a) and (b) to give 00......001 and use (c) as well to give the code,

n	k	d	G
36	18	4	$\{2^{17}-1,\ 2^{16}-1,\ -I\}$

This is obviously not efficient, with or without parity-check.

An example of the most efficient use of redundancy for the codes of Chapter 4 is shown below for k' = 18. The redundancy is assumed on the 1st digit.

$$\begin{array}{c}
1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1 \\
1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1 \\
\hline
0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0 \\
\hline
\end{array}$$

If the lowest weight, k'-tuple, used in a code is b, then a b-weight k'-tuple has k'-b, zero's. Therefore it must be mod 2 added to a k'-tuple of weight x = k'-b+1, to form an efficient form as above. Since x = k'-b+1, if $x \neq b$, then from the constructions of Chapters 4 and 5, we must be bounded by $x \simeq b$, because we cannot have redundant k'-tuples of widely differing weights.

In the example above, ignoring the k'+1'th digit, k' = 18, b = 9, so that,

$$x = 18 - 9 + 1 = 10.$$

Since we require $x \simeq b$, in the optimum case x = b, giving

$$x = b = \left[\frac{k' + 1}{2}\right] \qquad \text{A.1.}$$

which is only possible if k' is odd.

Equation A.1. states that if two redundant k'-tuples, k' odd, sum mod 2. to give the form 0111....110, then if they have the same weight, this weight must be $\frac{k'+1}{2}$.

This implies that given the complete set of redundant k'-tuples of weight $\frac{k'+1}{2}$, for every k'-tuple of this weight there is a corresponding k'-tuple of this weight which seems to give the form 0111....110, and the complete set can be paired off.

For example, let k' = 5, then the complete set of redundant 5-tuples of weight $\frac{k'+1}{2}$ = 3, is given below paired off.

$$\begin{array}{c} 1\;0\;0\;1\;1\;1 \\ 1\;1\;1\;0\;0\;1 \\ \hline 0\;1\;1\;1\;1\;0 \\ \hline \end{array} \qquad \begin{array}{c} 1\;1\;0\;0\;1\;1 \\ 1\;0\;1\;1\;0\;1 \\ \hline 0\;1\;1\;1\;1\;0 \\ \hline \end{array} \qquad \begin{array}{c} 1\;0\;1\;0\;1\;1 \\ 1\;1\;0\;1\;0\;1 \\ \hline 0\;1\;1\;1\;1\;0 \\ \hline \end{array}$$

Therefore this form of redundancy is only possible, using two redundant k'-tuples, when $b \simeq \frac{k'+1}{2}$ and is as such limited. Hence the need for 10 × 18-tuples to obtain this form when $b = 16 >> \frac{k'+1}{2}$, in the initial example.

For values of $b \neq \frac{k'+1}{2}$, the code must resort to the use of the next most efficient form, 1000....001. In its most efficient construction it can be obtained with 3 redundant k'-tuples.

Since it is likely that the redundant k'-tuples will be composed of one or two weights b_1 and or b_2, we will assume $b_2 > b_1$ and $b_1 = b_2 - 1$, and these are the only weights present.

To be able to obtain the form 1000....001 by summing mod 2, three k'-tuples (which is the most efficient form for the codes of section 5.2.), we require the mod 2 sum of two k'-tuples to have weight b_1-1 or b_2-1 in order to combine with another, third k'-tuple. Let the two k'-tuples agree in y positions that are binary one, then we require the following conditions to be satisfied.

i) both k'-tuples have weight b_2.

$$(b_2-y) + (b_2-y) = b_2-1$$
$$\text{or } b_1-1$$

giving

$$y = \frac{b_2+1}{2} \quad \text{or} \quad \frac{2b_2-b_1+1}{2} = \frac{b_2+2}{2}$$

respectively.

ii) both k'-tuples have weight b_1.

$$(b_1-y) + (b_1-y) = b_1 - 1 \quad \text{or } b_2 - 1$$

giving,

$$y = \frac{b_1+1}{2} \quad \text{or} \quad \frac{2b_1-b_2+1}{2} = \frac{b_2-1}{2} = \frac{b_1}{2}$$

respectively.

iii) the k'-tuples have different weights.

$$(b_1-y) + (b_2-y) = b_2 - 1 \quad \text{or } b_1 - 1$$

giving,

$$y = \frac{b_1+1}{2} \quad \text{or} \quad \frac{b_2+1}{2}$$

respectively.

For all cases we must also satisfy

$$(b_i-y) + (b_j-y) + y \leqslant k' \qquad \text{A.2.}$$

If $i = j$,

$$b_i \leqslant \left[\frac{k' + y}{2}\right] \quad \text{, and therefore}$$

$$\begin{aligned}
b_1 &\leqslant \left[\frac{2k' + 1}{3}\right] & &\text{if } y = \frac{b_1+1}{2}, & i &= 1\\
b_1 &\leqslant \left[\frac{2k'}{3}\right] & &\text{if } y = \frac{b_1}{2}, & i &= 1\\
b_2 &\leqslant \left[\frac{2k' + 1}{3}\right] & &\text{if } y = \frac{b_2+1}{2}, & i &= 2\\
b_2 &\leqslant \left[\frac{2k' + 2}{3}\right] & &\text{if } y = \frac{b_2+2}{2}, & i &= 2
\end{aligned} \qquad \text{A.3.}$$

From equation A.2. let $i = 1$ and $j = 2$, then

$$b_1 + b_2 \leqslant k' + y$$

$$2b_1 + b_2 \leqslant 2k' + 1 \qquad \text{if } y = \frac{b_2+1}{2}$$

$$2b_2 + b_1 \leqslant 2k' + 1 \qquad \text{if } y = \frac{b_1+1}{2}$$

Since $b_2 - 1 = b_1$, we obtain

$$b_2 \leqslant \left[\frac{2k' + 3}{3}\right] \qquad \text{if } y = \frac{b_2+1}{2}$$

$$b_1 \leqslant \left[\frac{2k' - 1}{3}\right] \qquad \text{if } y = \frac{b_1+1}{2} \qquad \text{A.4.}$$

For all equations A.3. and A.4. if $k' \gg 1$

$$b_i \leqslant \left[\frac{2k'}{3}\right] , \qquad i = 1 \text{ or } 2.$$

is a reasonable approximation and implies that the use of redundancy to obtain the form 1000....001 is inefficient for b greater than this value, for both codes from Chapter 4 and section 5.2.

APPENDIX B.

The equations for length, n, and check sums, J, for the codes presented in Chapter 4 are;

$$J = \sum_{t=b}^{k'-1} \frac{(k'-1)!}{t!\ (k'-1-t)!}$$

$$n = \sum_{t=b}^{k'} \frac{k'!}{t!\ (k'-t)!}$$

where b is the smallest weight of the complete sets of k'-tuples used. Since

$$H(x) = -x \log_2(x) - (1-x) \log_2(1-x)$$

$$H\left(\frac{d_m-2}{n}\right) = \frac{d_m-2}{n}\left[\log_2 n\right] - \frac{d_m-2}{n}\left[\log(d_m-2)\right]$$

$$+ \frac{(n-d_m+2)}{n}\left[\log_2 n\right] - \frac{n-d_m+2}{n}\left[\log(n-d_m+2)\right]$$

Let,

k' = 17

b	n	J	H(J/n)	1 - (k/n)
16	18	1	0.3095434	0
15	154	17	0.50109	0.8831168
14	834	137	0.6444242	0.9784172
13	3214	697	0.7543993	0.9943995
12	9402	2517	0.8381577	0.998055
11	21778	6885	0.9001395	0.9991734
10	41226	14893	0.9437175	0.9995633
9	65536	26333	0.9719996	0.9997253
8	89846	39203	0.9882728	0.9997996
7	109294	50643	0.9961229	0.999835
6	121670	58651	0.9990692	0.999852
5	127858	63019	0.9998531	0.9998592
4	130238	64839	1.0004747	0.99986
3	130918	65399	1.00019	0.99986
2	131054	65519	0.999996	0.9998626
1	131071	65535	0.999999	0.9998626
0	131072	65536	0.999999	0.9998626

k' = 8

b	n	J	H(J/n)	1 - (k/n)
7	9	1	0.5032582	0
6	37	8	0.7531979	0.7567567
5	93	29	0.8952722	0.9032258
4	163	64	0.9177783	0.9447852
3	219	99	0.993357	0.9589041
2	247	120	0.9994205	0.9635627
1	255	127	0.9999889	0.9647058
0	256	128	1.0	0.9648437

$k' = 6$

b	n	J	H(J/n)	1 - (k/n)
5	7	1	0.5916727	0
4	22	6	0.8453508	0.6818181
3	42	16	0.9587118	0.833333
2	57	26	0.9944422	0.8771929
1	63	31	0.9998181	0.88888
0	64	32	1.0	0.890625

APPENDIX C.

Majority-logic decoding of some of the quasi-cyclic codes from Table 8.3. Peterson and Weldon[5], pp.259.

(a) The code, $(n, k, d_m) = (6, 3, 3)$ has generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Since this is two circulants, any check sums obtainable on one message-digit, from linear sums of columns of the matrix, can also be obtained on all message-digits. The following check sums on message-digit m(1) are obtained from the columns.

$$\begin{array}{ccc} \overline{\underline{0\ 0\ 1}} & 0\ 1\ 0 & 1\ 0\ 0 \\ & \underline{0\ 1\ 1} & \underline{1\ 0\ 1} \\ & \underline{0\ 0\ 1} & \underline{0\ 0\ 1} \end{array}$$

Thus $J = d_m = 3$ and the code is completely orthogonalizable in one-step.

(b) The code, (10, 5, 4) has generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The following check sums, from sums of columns, are possible on m(1).

$$\begin{array}{c} \hline 0\,0\,0\,0\,1 \\ \hline \\ \\ \\ \end{array} \qquad \begin{array}{c} 0\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,1\,1\,1 \\ \hline 0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,1\,1\,1\,0 \\ 1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,1 \\ \hline 0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,1\,0\,0\,0 \\ 1\,0\,0\,0\,0 \\ 1\,1\,0\,0\,1 \\ \hline 0\,0\,0\,0\,1 \\ \hline \end{array}$$

Again $J = d_m = 4$.

(c) The code, (12, 6, 4), has generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The following check sums are obtained.

$$\begin{array}{c} \hline 0\,0\,0\,0\,0\,1 \\ \hline \\ \\ \\ \\ \end{array} \qquad \begin{array}{c} 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,1\,1 \\ \hline 0\,0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,1\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,1 \\ \hline 0\,0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,0\,1\,1\,1\,0 \\ 0\,1\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,0 \\ 1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,0\,0\,0 \\ \hline 0\,0\,0\,0\,0\,1 \\ \hline \end{array}$$

And $J = d_m = 4$.

(d) The code, (14, 7, 4), has generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The following check sums are obtained.

$$\begin{array}{c} \hline 0\,0\,0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1\,1 \\ \hline 0\,0\,0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{c} 0\,1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0\,1 \\ \hline 0\,0\,0\,0\,0\,0\,1 \\ \hline \end{array}$$

$$\begin{array}{c} 0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,1\,1\,1\,0\,0 \\ 0\,1\,1\,1\,0\,0\,0 \\ 1\,1\,1\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,1\,1 \\ \hline 0\,0\,0\,0\,0\,0\,1 \\ \hline \end{array}$$

And $J = d_m = 4$.

REFERENCES

1. LIN, S., "An Introduction to Error-Correcting Codes", 1970, Prentice-Hall, Inc., Englewood Cliffs, N.J., pp.33-56.

2. PETERSON, W.W., and WELDON, E.J. Jnr., "Error-Correcting Codes", 2nd Ed. 1972, The M.I.T. Press, Cambridge, Massachusetts., pp.310-356.

3. LIN, S., "An Introduction to Error-Correcting Codes", 1970, Prentice-Hall, Inc., Englewood Cliffs, N.J., pp.87-109.

4. MASSEY, J.L., "Threshold Decoding", 1963, The M.I.T. Press, Cambridge, Massachusetts.,

5. PETERSON, W.W., and WELDON, E.J. Jnr., "Error-Correcting Codes", 2nd. Ed. 1972, The M.I.T. Press, Cambridge, Massachucetts., pp.237-261.

6. LIN, S., "An Introduction to Error-Correcting Codes, 1970, Prentice-Hall, Inc., Englewood Cliffs. N.J. pp.140-181.

7. NG, S.W., "On Rudolph's Majority-Logic Decoding Algorithm", Sept. 1970, I.E.E.E. Trans. Information Theory, Corresp., Vol. IT-16, pp.651-652.

8. RUDOLPH, L.D. "Threshold Decoding of Cyclic Codes", May 1969, I.E.E.E. Trans. Information Theory, Vol. IT-15, pp.414-418.

9. RUDOLPH, L.D. "A Class of Majority-Logic Decodable Codes", April 1967, I.E.E.E. Trans. Information Theory, Corresp., Vol. IT-13, pp.305-307.

10. CHEN, C.L., PETERSON, W.W. and WELDON, E.J. Jnr., "Some Results on Quasi-Cyclic Codes", 1969, Information and Control, Vol. 15, pp.407-423.

11. WARREN, W.T. and CHEN, C.L. "On Efficient Majority-Logic Decodable Codes", Nov. 1976, I.E.E.E. Trans. Information Theory, Vol. IT-22, pp.737-745.

12. RIEK, R.J., HARTMANN, C.R.P. and RUDOLPH, L.D. "Majority Decoding of Some Classes of Binary Cyclic Codes", Sept. 1974, I.E.E.E. Trans. Information Theory, Vol. IT-20, pp.637-643.

13. SHIVA, S.G.S. and TAVARES, S.E. "On Binary Majority-Logic Decodable Codes", Jan. 1974, I.E.E.E. Trans. Information Theory, Corresp., Vol. IT-20, pp.131-132.

14. LAFERRIERE, C. and SHIVA, S.G.E. "On 1-Step Majority-Logic Decoding", June 1977, Proceedings of I.E.E., Vol. 124, pp.527-528.

15. HASHIM, A.A. and CONSTANTINIDES, A.G. "Class of Linear Binary Codes", July 1974, Proceedings of I.E.E., Vol. 121, pp.555-558.

16. HARTMANN, C.R.P., DUCEY, J.B. and RUDOLPH, L.D. "On the Structure of Generalised Finite-Geometry Codes", Mar. 1974, I.E.E.E. Trans. Information Theory, Vol. IT-20, pp.240-252.

17. LIN, S. "Multifold Euclidean Geometry Codes", July 1973, I.E.E.E. Trans. Information Theory, Vol. IT-19, pp.537-548.

18. RUDOLPH, L.D. and ROBBINS, W.E. "One-Step Weighted-Majority Decoding", May 1972, I.E.E.E. Trans. Information Theory, Corresp., Vol. IT-18, pp.446-448.

19. DUC, N.Q. "Pseudostep Orthogonalisation: A New Threshold-Decoding Algorithm", Nov. 1971, I.E.E.E. Trans. Information Theory, Corresp. Vol. IT-17, pp.766-767.

20. LIN, S., and YIU, K-P. "An Improvement to Multifold Euclidean Geometry Codes", 1975, Information and Control, IC-28, pp.221-265.

21. CHEN, C.L. "On Majority-Logic Decoding of Finite-Geometry Codes", May 1971, I.E.E.E. Trans. Information Theory, Vol. IT-17, pp.332-336.

22. RUDOLPH, L.D., and MITCHELL, M.E. "Implementation of Decoders for Cyclic Codes", 1964, I.E.E.E. Trans. Information Theory, Vol. IT-10, pp.259-260.

23. KASAMI, T. "A Decoding Procedure for Multiple Error-Correcting Cyclic Codes", 1964, I.E.E.E. Trans. Information Theory, Vol. IT-10, pp.134-139.

24. MacWILLIAMS, J. "Permutation Decoding of Systematic Codes", 1964, Bell Syst. Tech. Jnl. 43, pp.485-505.

25. OMURA, J.K. "A Probabilistic Decoding Algorithm for Binary Group Codes", 1969, S.R.I. Technical Report, Project 664531-226.

26. KASAMI, T. and LIN, S. "On Majority-Logic Decoding for the Duals of Primitive Polynomial Codes", 1971, I.E.E.E. Trans. Information Theory, Vol. IT-17, pp.322-331.

27. KARLIN, M. "New Binary Coding Results by Circulants", 1969, I.E.E.E. Trans. Information Theory, Vol. IT-15, pp.81-92.

28. TOWNSEND, R.L., and WELDON, E.J. Jnr. "Self-orthogonal Quasi-cyclic Codes", 1967, I.E.E.E. Trans. Inf. Th., Vol. IT-13, pp.183-195.

29. PETERSON, W.N. and WELDON, E.J. Jnr. "Error-Correcting Codes", 2nd. Ed. 1972, The M.I.T. Press, Cambridge, Mass., pp.40-47.

30. REED, I.S. "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme", 1954, I.R.E. Trans., PGIT-4, pp.38-49.

31. MULLER, D.E. "Application of Boolean Algebra to Switching Circuit Design and Error Detection", 1954, I.R.E. Trans., EC-3, pp.6-12.

32. YALE, R.B. "Error-Correcting Codes and Linear Recurring Sequences", 1958, Lincoln Lab. Report 34-77, Lincoln Lab., M.I.T.

33. ZIERLER, N. "On a Variation of the First Order Reed-Muller Codes", 1958, Lincoln Lab. Report 34-80, Lexington, Mass., M.I.T.

34. HAMMING, R.W. "Error Detecting and Error-Correcting Codes", 1950, Bell Syst. Tech. Jnl. 29, pp.147-160.

35. BOSE, R.C. and RAY-CHAUDHURI, D.K. "On a Class of Error-Correcting Binary Group Codes", 1960, Information and Control, Vol. IC-3, pp.68-79.

36. HOCQUENGHEM, A. "Codes Correcteurs D'Erreurs", 1959, Chiffres, 2, pp.147-156.

37. MITCHELL, M.E. et al. "Coding and Decoding Operations Research", 1961, U.S.A.F. Cambridge Research Centre, Contract AF 19(604)-6183.

38. PETERSON, W.W. and WELDON, E.J. Jnr. "Error-Correcting Codes", 2nd Ed. 1972, The M.I.T. Press, Camb. Mass. Chapter 5.

39. WELDON, E.J. Jnr. "Difference-Set Cyclic Codes", 1966, Bell Syst. Tech. Jnl. 45, pp.1045-1055.

40. HELGERT, J.H. and STINAFF, R.D. "Minimum-Distance Bounds for Binary Linear Codes", 1973, I.E.E.E. Trans. Inf. Thry. Vol. IT-19, pp.344-356.

41. LeVEQUE, J.W. "Topics in Number Theory", 1958, Addison-Wesley Pub. o. Inc., Chapters 3,4 and 5.

42. McCOY, N.H. "The Theory of Numbers", 1965, The Macmillan Co. N.Y., Chapters 1,2 and 3.

43. BEILER, A.H. "Recreations in the Theory of Numbers: The Queen of Mathematics Entertains", 1966, Dover Pub. Inc. N.Y.

44. RUDOLPH, L.D. "Geometric Configurations and Majority-Logic Decodable Codes", 1964, M.E.E. Thesis, University of Oklahoma.

45. WELDON, E.J. Jnr. "Euclidean Geometry Cyclic Codes", 1967, Proceedings of Symp. of Combinat. Maths., Univ. of North Carolina, Chapel Hill, N.C.

46. WELDON, E.J. Jnr. "New Generalizations of the Reed-Muller Codes - Part II: Non-Primitive Codes", 1968, I.E.E.E. Trans. Inf. Th., Vol. IT-14, pp.199-206.

47. WELDON, E.J. Jnr. "Some Results on Majority-Logic Decoding", 1968, "Error-correcting Codes", H.B. Mann, John Wiley & Sons Inc.

48. KASAMI, T., LIN, S. and PETERSON, W.N. "New Generalizations of the Reed Muller Codes - Part I: Primitive Codes", 1968, I.E.E.E. Trans. Inf. Thry., Vol. IT-14, pp.189-199.

49. KASAMI, T., et al. "Some Results on Cyclic Codes which are invariant under the affine Group", 1966, U.S.A.F. Cambridge Research Lab. Report.

50. KASAMI, T., et al. "Polynomial Codes", 1968, I.E.E.E. Trans. Inf. Thry., Vol.IT-14, pp.807-814.

51. LIN, S. "On a Class of Cyclic Codes", 1968, "Error-Correcting Codes", Ed. H.B. Mann, John Wiley and Sons Inc., N.Y.

52. DELSARTE, P. "A Geometric Approach to a Class of Cyclic Codes", 1969, Journal of Combin. Thry. 6, pp.340-359.

53. LIN, S. and WELDON, E.J. Jnr. "New Efficient Majority-Logic Decodable Cyclic Codes", 1972, I.E.E.E. Int. Symp. on Inf. Th., Asilomar, Calif.

54. HARTMANN, C.R.P. and RUDOLPH, L.D. "Generalised Finite Geometry Codes", 1972, Proc. 10'th Ann. Conf. on Cct. and Syst. Th., Univ. Illinois, Urbania, Ill.

55. CHEN, C.L. "Note on Majority Logic Decoding of Finite Geometry Codes", 1972, I.E.E.E. Trans. Inf. Thy., Vol IT-18, pp.446-448.

56. CHEN, C.L. "On Majority-Logic Decoding of Finite Geometry Codes", 1971, I.E.E.E. Trans. Inf. Thy., Vol. IT-17, pp.332-336.

57. CHEN, C.L. "On Shortened Finite Geometry Codes", 1972, Information and Control, Vol.20, pp.216-221.

58. LIN, S. "Shortened Finite Geometry Codes", 1972, I.E.E.E. Trans. Inf. Thy., Vol. IT-18, pp.692-696.

59. CHEN, C.L. and WARREN, W.T. "A Note on One-step Majority-Logic Decodable Codes", 1973, I.E.E.E. Trans. Inf. Thy., Vol. IT-19, pp.135-137.

60. RUDOLPH, L.D. and HARTMANN, C.R.P. "Decoding by Sequential Code Reduction", 1973, I.E.E.E. Trans. Inf. Thy., Vol. IT-18, pp.549-555.

61. CHOW, D.K. "On Threshold Decoding of Cyclic Codes", 1968, Inf. and Control, Vol. 13, pp.471-483.

62. GORE, W.C. "Generalised Threshold Decoding of Linear Codes", 1969, I.E.E.E. Trans. Inf. Thy., Vol. IT-15, pp.590-592.

63. GORE, W.C. "Generalised Threshold Decoding and the Reed-Solomon Codes", 1969, I.E.E.E. Trans. Inf. Thy., Vol. IT-15, pp.78-81.

64. DUC, N.Q. "On a Necessary Condition for L-step Orthogonalisation of Linear Codes and its Applications", 1973, Inf. and Cont., Vol. 22, pp.123-131.

65. LIN, S and WELDON, E.J. Jnr. "Further Results on Cyclic Product Codes", 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.452-459.

66. GORE, W.C. "Further Results on Product Codes". 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.446-451.

67. DUC, N.Q. and SKATTERBOL, L.V. "Further Results on Majority-Logic Decoding of Product Codes", 1972, I.E.E.E. Trans. Inf. Thy., Vol. LT-18, pp.308-310.

68. DUC, N.Q. "On the Lin-Weldon Majority-Logic Decoding Algorithm for Product Codes", 1973, I.E.E.E. Trans. Inf. Thy., Vol. IT-19, pp.581-583.

69. BOBROW, L.S. "Decoding Augmented Cut-Set Codes", 1971, I.E.E.E. Trans. Inf. Thy., Vol. IT-17, pp.218-220.

70. KASAMI, T. and LIN, S. "On the Construction of a Class of Majority-Logic Decodable Codes", 1971, I.E.E.E. Trans. Inf. Thy., Vol. IT-17, pp.600-610.

71. ASSMUS, E.F. Jnr. GOETHALS, J.M. and MATTSON, H.F. Jnr. "Generalised t-Designs and Majority-Logic Decoding of Linear Codes", 1976, Inf. and Contr., Vol. 32, pp.43-60.

72. ASSMUS, E.F. and MATTSON, H.F. "t-Design Decoding and the (48, 24) Binary Q.R Code", 1¼70, G.T.E. Sylvania, Rep. AFCRL-71-0013.

73. GOETHALS, J.M. "On t-Design and Threshold Decoding", 1970, Inst. Statist., Univ. North Carolina, Mimeo Series 600.29.

74. ASSMUS, E.F. and MATTSON, H.F. "Majority Decoding of the (24, 12) Binary Golay Code", 1969, G.T.E. Sylvania, Rep. Contract F-19628-69-C-0068.

75. RAHMAN, M. and BLAKE, I.F. "Majority-Logic Decoding Using Combinatorial Designs", 1975, I.E.E.E. Trans. Inf. Thy., Vol. IT-21, pp.585-587.

76. RAHMAN, M. and BLAKE, I.F. "Combinatorial Aspects of Orthogonal Parity Checks", 1976, I.E.E.E. Trans. Inf. Thy., Vol. IT-22, pp.759-762.

77. SUNDBERG, C. "One-step Majority-Logic Decoding with Symbol Reliability Information", 1975, I.E.E.E. Trans. Inf. Thy., Vol. IT-21, pp.236-242.

78. HARRISON, C.N. "Application of Soft Decision Techniques to Block Codes", 1977, I.E.R.E. Conf. on Digital Process, Univ. of Loughborough, Leics., England.

PART II

RANDOM ERROR-CORRECTING MAJORITY-LOGIC DECODABLE CONVOLUTIONAL CODES.

CHAPTER 8

8. INTRODUCTION.

8.1 Outstanding Problems.

It would be fair to say that the state of the art, with regard to random error-correcting majority-logic decodable convolutional codes, is still in its infancy. Particularly so compared to block codes. Very little has been achieved towards devising large constructive classes of majority-logic codes and one of the outstanding problems still awaiting a solution is the development of a constructive class of orthogonalizable codes. Those convolutional orthogonalizable codes which are known have been developed, using trial and error methods or hybrid use of block codes, are relatively small classes or of low transmission rate.

Of those majority-logic codes that have so far been devised, namely, orthogonalizable and self-orthogonal, the orthogonalizable have exhibited far better minimum distances, d_m, for a given length and rate, than the self-orthogonal. However, when decoded in the feedback mode the orthogonalizable codes have no inherent protection against propagated errors due to decoding failures in previous blocks. This is a problem encountered in all other known types of convolutional codes too, except self-orthogonal codes which are the only known type of convolutional code that have an inherent ability to automatically recover from error propagation.

However, regardless of which type of majority-logic convolutional code we consider there has, up to this time, been another limiting factor in their design. We saw in Part I that when a majority-logic block code is cyclic we can decode with a single majority-logic gate. There is nothing analagous to a cyclic code with convolutional codes* so that to decode a block of digits containing k_o information digits we always require k_o majority-logic gates. This has lead to the development of codes with small k_o (in fact for many existing codes $k_o \leq 5$). If k_o is small then the transmission rate cannot be very high, so that there is no constructive class of majority-logic decodable convolutional codes that are multiple error-correcting and very high rate.

8.2 Project Resumé.

The codes presented in Chapter 11 offer a solution to this second problem, their properties including,

i) k_o the number of message-digits in a sub-block can be very high, in particular if p^α , $\alpha = 1,2,\ldots\ldots$ is a power of an odd prime, $k_o = p^\alpha - 1$.

ii) the codes are cyclically encodable and decodable with a consequent reduction in the number of majority-logic gates. In particular if $k_o = p-1$, p an odd prime, and if the rate, $R = 1/2$, only one majority-gate is required and all k_o digits can be decoded cyclically.

iii) the codes are self-orthogonal and therefore majority-logic decodable in one-step.

iv) the codes are high rate with $R \geq 1/2$. In particular it is shown that a single error-correcting code exists for every

* Majority-logic codes that is.

positive integer m, with rate

$$R = k_o/k_o+2 = (m-1)/(m+1).$$

v) since the codes are self-orthogonal they have the ability to automatically recover from error propagation. In particular it is shown that the codes have the ability to recover from multiple error propagation.

The codes are compared with existing majority-logic codes on the basis of minimum distance, rate and length. An examination of their unbounded performance, when the number of errors exceeds the codes capability, is shown in Chapter 12, and this is compared with the unbounded performance of another well known class of self-orthogonal codes[7].

The idea of error-propagation efficiency is introduced to enable subjective comparison of different codes.

In Chapter 13 a pseudostep orthogonalization algorithm is introduced, which, though it may be applied to Reed - Massey[1] algorithm majority-logic codes, is applied to a well known class of orthogonalizable convolutional codes[1] with a consequent improvement in their unbounded performance.

Finally in Chapter 14 an assessment of the original work presented in Part II is conducted.

CHAPTER 9

9. SURVEY OF MAJORITY-LOGIC CONVOLUTIONAL CODES.

9.1 Non-Self-orthogonal Codes.

Massey[1] was the first to show that convolutional codes could be majority-logic decoded and he presented three classes of non-self-orthogonal codes. These three classes were,

1) Trial and Error, a large class constructed, as the name implies, by trial, with low rate $\leq 1/2$,

2) Uniform, a class of low rate $1/n_o$,

3) "Reed-Muller" like, another class of low rate $1/n_o$ codes.

In 1968, Reddy and Robinson[9] showed that Massey's Reed-Muller like codes[1] can be constructed directly from Reed-Muller block codes. They also showed that any linear Uniform code can be constructed using a MacDonald block code. Several algorithms were presented for constructing convolutional codes from block codes and resulted in long low rate codes. In the same year Reddy[10] presented new algorithms for constructing convolutional short high rate codes from block codes.

In another paper, in the same year, Reddy and Robinson[11] presented decoding algorithms for the codes presented previously. If the block code is one-step majority-logic decodable then the resulting convolutional code is one-step decodable also. In general the decoder construction is closely related to the decoder of the block code. In the general decoder a number of block decoders are required before processing and making a majority decision. It is also shown that the codes can decode with limited error propagation provided a reduction in capability can be tolerated. In 1972 Reddy and Robinson[20] presented further results on orthogonalisable convolutional codes, this time constructed from majority-logic decodable self-orthogonal convolutional codes, into which are imbedded block codes. Two constructions are presented giving low and high rate codes with decoders related to the block code decoders. The low rate construction has limited error propagation provided a reduction in capability can be tolerated whilst the position for the high rate construction is stated as being "not clearly understood".

9.2 Self-orthogonal Codes.

Again Massey[1] first presented a small class of rate 1/2 self-orthogonal codes and he noted that the large difference between actual and effective constraint lengths made the encoder and decoder larger than necessary.

In 1967, Robinson and Bernstein[7] presented a class of self-orthogonal codes whose construction was based upon difference triangles. More importantly, they showed that any convolutional self-orthogonal code, C.S.O.C., could recover automatically from error propagation when decoded in the feedback mode. In a later paper Robinson[12] showed that a code could recover from error propagation within a limited number of bits, using Feedback Decoding, if and only if that same code can be decoded using Definite Decoding. Significantly self-orthogonal codes remain the only known class of convolutional codes with this property inherent in their structure. This class of codes started by Robinson[7] et al. was later extended by Klieber[14] and virtually completed by Wu[22,23,24]. In Wu's first paper[22] he tabulated new C.S.O.C.'s of rates 2/3 up to 13/14 and in addition presented three examples of C.S.O.C.'s used in commercial satallite systems. For one example a significant result of an extensive evaluation including B.C.H. codes, cyclic difference set codes and both Viterbi and Sequential Decoding pointed to a rate 7/8 C.S.O.C. In a later paper Wu[23] extended the tabulation of high rate C.S.O.C.'s from rate 14/15 up to 49/50. He introduced a unified construction algorithm and extended Massey's work on A Posteriori (APP) decoding to C.S.O.C.'s.

In another paper, in 1976, Wu[24] tabulated further rate 1/2 C.S.O.C.'s and examined the concatenation of C.S.O.C.'s generally. Wu states that one of the problems of concatenation is that the inner codes' decoder, such as Reed-Solomon, B.C.H. or Viterbi, produces extra and bursty errors at its output when confronted with error patterns whose weight exceeds the capability of the code at its input. Wu found[24] that C.S.O.C. decoders exhibit properties which suppress both of these undesirable properties and make them attractive as inner concatenated codes.

9.3 Further Results.

Other results in the field of majority-logic decodable convolutional codes include results on burst error-correcting codes which can also correct random errors, from Tong[15] and then Ferguson[18]. Tong's[15] codes are high rate C.S.O.C.'s while Ferguson's[18] are rate 1/2 orthogonalizable.

Tong's[15] approach was to set up a coarse design and finalize it with a trial and error iterative procedure. This was done by designing a burst error-correcting convolutional code and then modifying it to correct random errors too.

Rudolph[13] showed that any linear convolutional code can be maximum-likelihood decoded, with respect to the decoding constraint length, by a one-step threshold decoder. The decoder utilized an exponentiation operator and threshold element for binary codes. In a later paper Rudolph and Robbins[21] showed that the correct statement for binary codes was to replace the exponentiation operator and threshold element by a weighted-majority element.

Goodman and Ng[25] recently presented results on soft-decision threshold decoding of convolutional codes. A random error and diffuse decoding scheme are proposed and an expected improvement over a hard decision decoder claimed. The ability to soft-decision decode majority-logic convolutional codes makes them highly competitive with other decoding methods such as Viterbi[8] and Sequential[2] decoding.

CHAPTER 10

10. CONVOLUTIONAL CODING THEORY.

10.1 Basic Description.[17]

With binary block codes, we saw in Chapter 3 that each block of n_o binary digits was uniquely determined by a block of $k_o < n_o$ binary digits, called the information block. Thus the check-digits of any n_o-digit block contain information regarding the k_o message (information)-digits which determined the block and no other message-digits. With a convolutional code this restriction is lifted and convolutional codes can be considered as a generalization of block codes. That is, we allow information, regarding the k_o binary digits of a message block, to be present in the check-digits of N different n_o-digit blocks called sub-blocks, with $n_o > k_o$.

The parameters N and $n_o N$ are called the constraint length of the code, in units of blocks and binary digits respectively.

The encoder functions by sub-dividing the incoming information stream, $\overline{m}$, into blocks of length k_o and stores N of these blocks. Having done this the encoder can then form the $n_o - k_o$ check-digits of some sub-block, say the e'th, to form an n_o-digit sub-block $\overline{c}_e$. Each binary digit of $\overline{c}_e$ can be represented by the following equations, which assume a systematic code,

$$c_e(i) = m_e(i), \qquad i = 1,2,\ldots,k_o \qquad 10.1.1.$$

$$c_e(k_o+j) = \sum_{i=1}^{k_o} m_e(i)\ g_o(i,j) \oplus \sum_{i=1}^{k_o} m_{e-1}(i)\ g_1(i,j) \oplus$$

$$\ldots\ldots\ldots\ldots \oplus \sum_{i=1}^{k_o} m_{e-N+1}(i)\ g_{N-1}(i,j) \qquad 10.1.2.$$

for $j = 1,2,\ldots,n_o-k_o$,

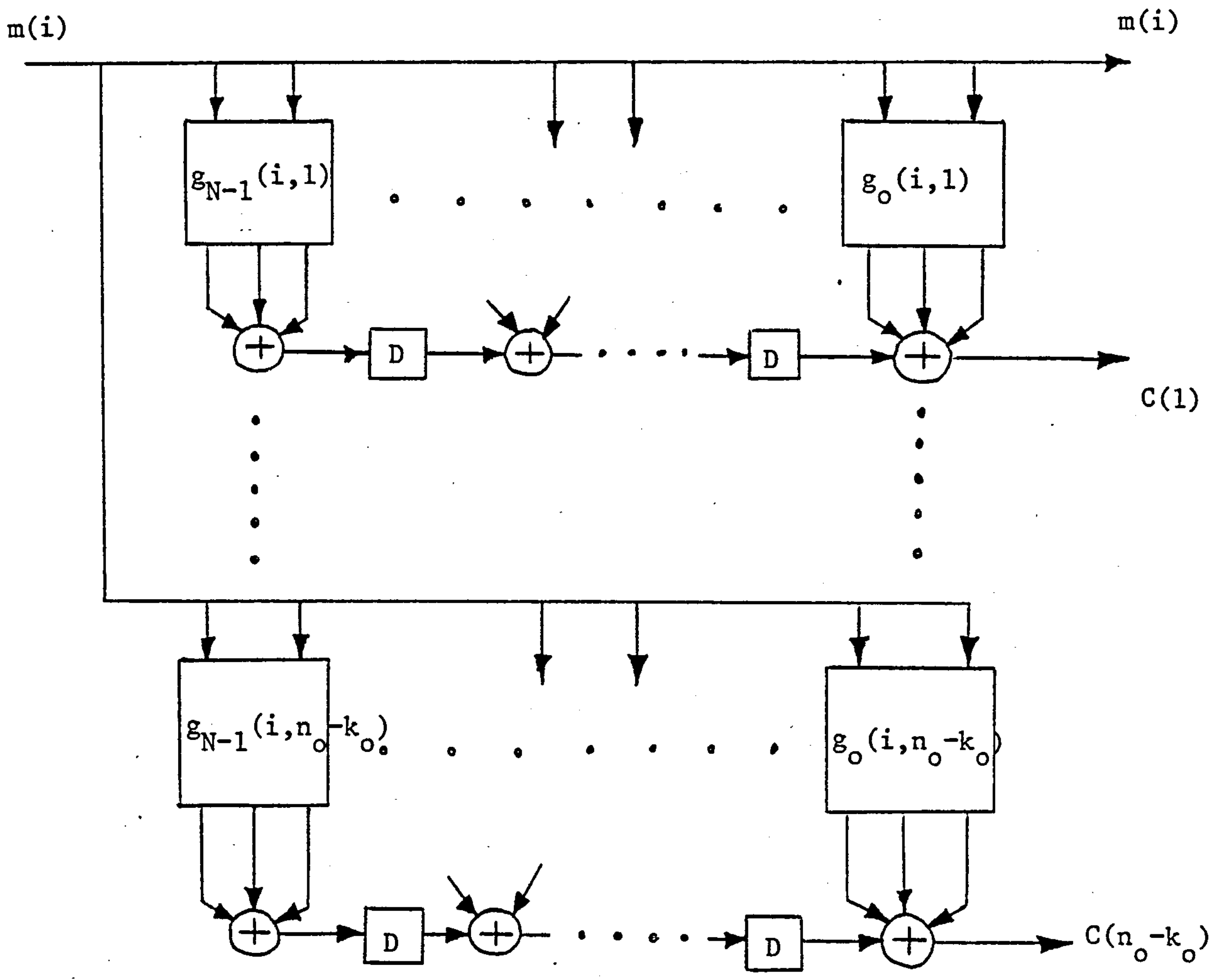


FIG. 10.1.1.

A GENERAL CONVOLUTIONAL ENCODER.

where;

$c_e(b)$ = b'th digit of $\bar{c}_e \in GF\ (2)$

$m_e(b)$ = b'th digit of $\bar{m}_e \in GF\ (2)$

$g_x(i,j)$ = 1 or 0, depending upon whether we do or do not respectively require $m_{e-x}(i)$ to be present in the check-digit $c_e(k+j)$. Therefore the encoder must store sub-blocks $\bar{m}_e, \bar{m}_{e-1}, \bar{m}_{e-2},\ldots,\bar{m}_{e-N+1}$ to form the encoded n_o-digit sub-block $\bar{c}_e$.

Let

$$g(i,j) = (g_o(i,j),\ g_1(i,j),\ldots,g_{N-1}(i,j)).$$

There are $k_o(n_o-k_o)$, N-digit binary vectors $g(i,j)$ and from equations 10.1.1. and 10.1.2. it can be seen that these vectors determine the n_o-digit sub-blocks. They therefore specify the code and are called the "sub-generators" of the code. Equations 10.1.1. and 10.1.2. specify a systematic code, but if the code is non-systematic we incorporate equation 10.1.1. into 10.1.2. and then require $k_o.n_o$ sub-generators $g(i,j)$, $i = 1,2,\ldots,k_o$, $j = 1,2,\ldots,n_o$.

The encoder must also store the $k_o.(n_o-k_o)$ sub-generators of the code. A general encoder for a convolutional code is shown in figure 10.1.1. where the blocks containing each $g_x(i,j)$ perform modulo 2 multiplication, the adders sum modulo 2 and the blocks containing D are simple delays to enable the storing of the N, k_o-digit message blocks.

10.2 General Decoding.[17]

We consider all data sequences as semi-infinite. This is because one cannot take out a section of data and say this is independent of any other data in the sequence, because any block is always related to the following N-1 sub-blocks. Also it is semi-infinite because at some time transmission must start. Therefore we represent the generator and parity-check matrices by the symbols G_∞ and H_∞ respectively.

Therefore we have,

$$G_\infty \cdot H_\infty^T = 0. \qquad 10.2.1.$$

Let $\bar{c}$ be a semi-infinite transmitted code sequence, and $\bar{m}$ a semi-infinite information sequence, then

$$\bar{c} = \bar{m} \cdot G_\infty .$$

Let $\bar{r}$ and $\bar{e}$ be semi-infinite received and noise sequences, respectively, then

$$\bar{r} = \bar{c} \oplus \bar{e}.$$

The syndrome is the semi-infinite sequence, $\bar{s}$, obtained by multiplying $\bar{r}$ by H_∞^T,

$$\begin{aligned}\bar{s} &= \bar{r} \cdot H_\infty^T \\ &= (\bar{c} \oplus \bar{e})H_\infty^T \\ &= \bar{c} \cdot H_\infty^T \oplus \bar{e} \cdot H_\infty^T \\ &= \bar{e} \cdot H_\infty^T \qquad 10.2.2.\end{aligned}$$

If we assume the first received sub-block of $\bar{r}$ is the 0'th block, we can represent $\bar{s}$ by,

$$\bar{s} = (s_o(1),\ s_o(2),\ldots\ldots,s_o(n_o-k_o),\ s_1(1),\ldots\ldots,s_1(n_o-k_o),$$
$$\ldots\ldots\ldots,\ s_e(1),\ldots\ldots,s_e(n-k),\ \ldots\ldots\ldots\ldots\ldots\ )$$

where, $s_e(1),s_e(2),\ldots\ldots,s_e(n_o-k_o)$ is the syndrome of the e'th block.

Therefore in order to obtain all the relevant information required to decode a single sub-block, the decoder must form and store the (n_o-k_o)-digit syndromes of the N blocks, whose check-digits contain information digits from the block to be decoded.

Let $\overline{r}_e$ be the received n_o-digit sub-block $\overline{c}_e$ plus errors. Then from equations 10.1.1. and 10.1.2. we can write,

$$r_e(i) = c_e(i) \oplus e_e(i) \qquad 10.2.3.$$

$$r_e(k_o+j) = c_e(k_o+j) + e_e(k_o+j) \qquad 10.2.4.$$

$$i = 1,2,\ldots,k_o \; ; \quad j = 1,2,\ldots,n_o-k_o,$$

where,

$c_e(k_o+j)$ is given by equation 10.1.2.

$e_e(k_o+j)$ is the single check-digit error acquired during transmission.

At the decoder $r_e(i)$, the received message-digit section, is re-encoded into the (n_o-k_o)-digit section $r'_e(k_o+j)$, where,

$$r'_e(k_o+j) = c_e(k_o+j) \oplus \sum_{i=1}^{k_o} e_e(i)\, g_o(i,j) \oplus \sum_{i=1}^{k_o} e_{e-1}(i)\, g_1(i,j) \oplus \ldots \oplus \sum_{i=1}^{k_o} e_{e-N+1}(i)\, g_{N-1}(i,j) \qquad 10.2.5.$$

$$i = 1,2,\ldots,k_o \; ; \quad j = 1,2,\ldots,n_o-k_o.$$

The decoder then forms the syndrome of block e, $\overline{s}_e$, by adding mod 2 the digits of equations 10.2.4. and 10.2.5. so that for each digit, $s_e(j)$, of $\overline{s}_e$, we obtain

$$\begin{aligned} s_e(j) &= r_e(k_o+j) \oplus r'_e(k_o+j) \\ &= e_e(k_o+j) \oplus \sum_{i=1}^{k_o} e_e(i)\ g_o(i,j) \oplus \sum_{i=1}^{k_o} e_{e-1}(i)\ g_1(i,j) \\ &\oplus \cdots\cdots \oplus \sum_{i=1}^{k_o} e_{e-N+1}(i)\ g_{N-1}(i,j) \end{aligned} \qquad 10.2.6.$$

where i and j are as in equation 10.2.5.

Of course the decoder performs this operation N times before it is ready to decode a block. Let us assume we are about to decode block 0, then to decode we require N, (n_o-k_o)-digit, syndromes $s_o(j),\ldots,s_{N-1}(j)$ and the equations relating to these syndromes are shown in equation 10.2.7. It can be seen that the errors in the message-digits of block 0 are present in all N syndromes.

$$\begin{aligned} s_o(j) &= e_o(k_o+j) \oplus \sum_{i=1}^{k_o} e_o(i)\ g_o(i,j) \\ s_1(j) &= e_1(k_o+j) \oplus \sum_{i=1}^{k_o} e_1(i)\ g_o(i,j) \oplus \sum_{i=1}^{k_o} e_o(i)\ g_1(i,j) \\ &\vdots \\ s_{N-1}(j) &= e_{N-1}(k_o+j) \oplus \sum_{i=1}^{k_o} e_{N-1}(i)\ g_o(i,j) \oplus \cdots\cdots \\ &\cdots\cdots \oplus \sum_{i=1}^{k_o} e_o(i)\ g_{N-1}(i,j) \\ j &= 1,2,\ldots,n_o-k_o. \end{aligned} \qquad 10.2.7.$$

If we assume the 0'th block is correctly decoded and the errors in block 0 are cancelled from the syndromes, then we obtain the set of syndrome equations below.

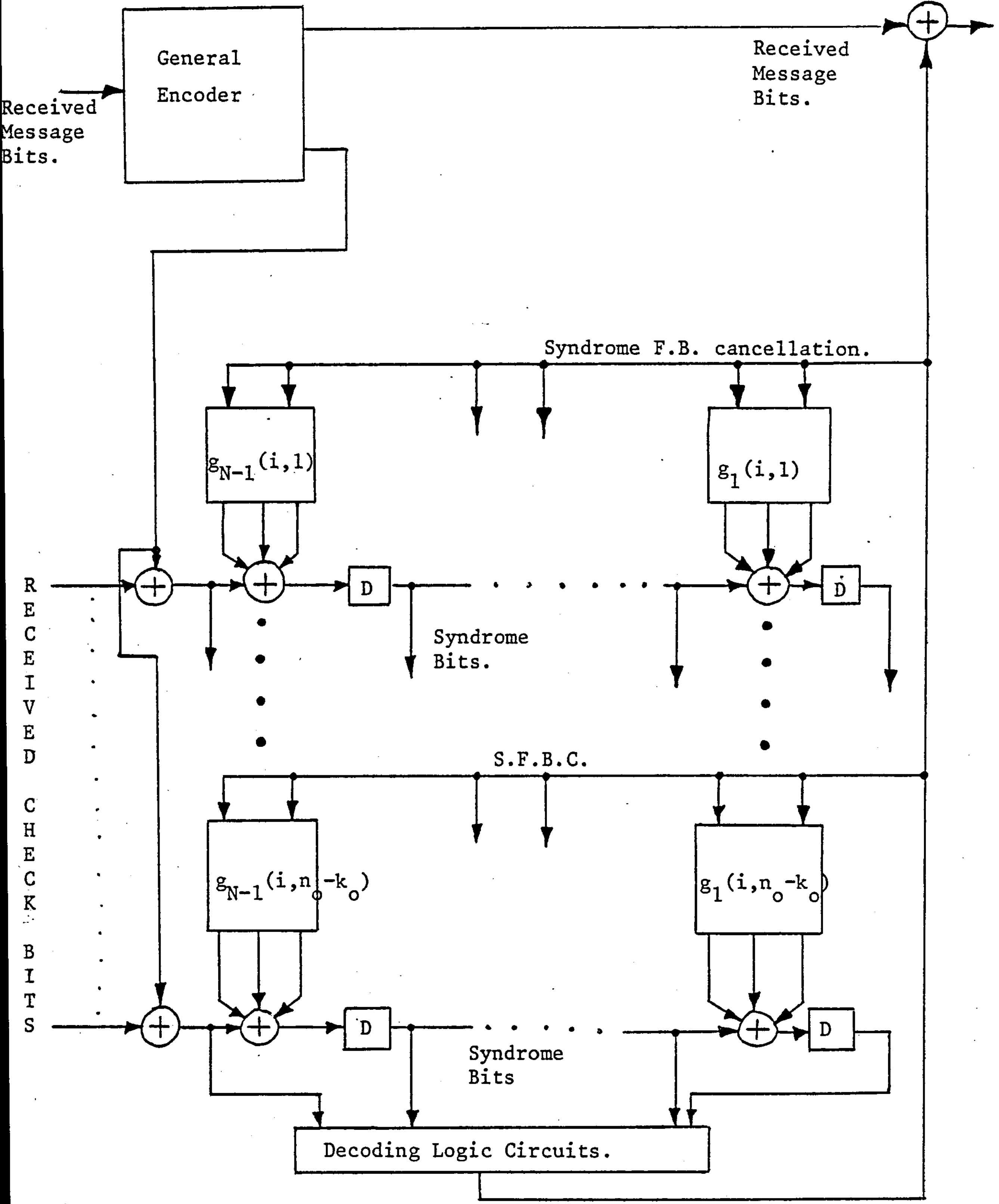


FIG. 10.2.1.

A GENERAL CONVOLUTIONAL DECODER WITH FEEDBACK.

$$\begin{aligned}
s_o(j) &= 0 \\
s_1(j) &= e_1(k_o+j) \oplus \sum_{i=1}^{k_o} e_1(i)\ g_o(i,j) \\
s_2(j) &= e_2(k_o+j) \oplus \sum_{i=1}^{k_o} e_2(i)\ g_o(i,j) \oplus \sum_{i=1}^{k_o} e_1(i)\ g_1(i,j) \\
&\ \vdots \\
s_N(j) &= e_N(k_o+j) \oplus \sum_{i=1}^{k_o} e_N(i)\ g_o(i,j) \oplus \ldots\ldots \oplus \sum_{i=1}^{k_o} e_1(i)\ g_{N-1}(i,j).
\end{aligned} \qquad 10.2.8.$$

So that we can decode the errors in block 1 using $s_1(j),\ldots,s_N(j)$ since they are identical in form, on block 1, as w ere $s_o(j),\ldots,s_{N-1}(j)$, on block 0, previously.

Once one has chosen the rules for encoding the k_o message-digits of a certain sub-block (which is the same as choosing the sub-generators), the same encoding rules apply to the k_o message-digits of every sub-block.

Also, therefore, if one can formulate rules for decoding a given sub-block, the same rules will decode every sub-block. However equations 10.2.8. were obtained assuming we cancelled the effects of the errors in block 0. This process is called syndrome cancellation and the procedure generally called feedback decoding. Figure 10.2.1. shows a general decoder for a convolutional code, decoded in the feedback mode.

10.3 Decoding and Error Propagation.[12,7,17]

The assumption in section 10.2 that the previous block was successfully decoded was useful to illustrate the general decoding procedure. However, in practice, this cannot be guaranteed and when a decoding failure occurs, due to the practice of syndrome cancelling the decoding failure manifests itself as errors in the syndromes of the following N-1 blocks. This property of convolutional codes, when decoded using syndrome cancellation, is referred to as error propagation.

It is possible for propagated errors to cause further decoding failures even in the absence of natural errors and is therefore an undesirable quality. Several methods for controlling the effect of propagation are known and also one (though perhaps obvious) method of eliminating it.

Periodic Resynchronization.[17]

Since a propagated error affects the decoding of the following N-1 blocks, if we can arrange periodically to guarantee to correctly decode a successive set of N-1 blocks, then propagation will be terminated. This is done by periodically encoding a set of $k_o(N-1)$ known digits (usually the all zero sequence). The price paid is a reduction in the transmission rate R to some figure R'. In particular if the known message-digits are encoded after every L blocks,

$$R' = \frac{k_o L}{L n_o + (N-1) n_o}$$

$$= \left(\frac{L}{L + N - 1}\right) R \qquad 10.3.1.$$

Assuming a worst case situation, that is a propagated error occurs in the decoding of the L'th block, since the following N-1 blocks, which the propagated error affects, are known, all errors in the N-1 blocks can be corrected and this must include the propagated error from block L.

Error Counting.[17]

This method involves counting the number of errors the decoder has estimated to have occurred, over a specified number of digits. If the count exceeds the error-correcting capability of the code, a retransmission is requested, on the basis that there is too much noise present at that time. Again the transmission rate will fall but not as much as with periodic resynchronization, provided the code is chosen sensibly and the channel noise is not too high compared to the error-correcting capability of the code.

Automatic Recovery.[12,7]

This involves the design of codes and decoders which have the ability to recover from error propagation without any external assistance. It is assumed that there are occasions, during which a long enough error free transmission period occurs, which allows the system to recover automatically. It has been shown[7] that self-orthogonal majority-logic decodable convolutional codes have the property of automatic recovery from error propagation.

It has also been shown[19] that if a code has the property that syndrome cancellation always acts to reduce the weight of the syndrome, then error propagation cannot occur.

Definite Decoding.[17,12]

Obviously if one does not use syndrome cancellation there is no feedback and error propagation cannot occur. This is called definite decoding and the price paid for this is a reduction in the error-correcting capability of the code. It has been shown[12] that self-orthogonal codes can be decoded in the definite decoding mode, simply by disconnecting the feedback connection.

Apart from the reduction in error-correcting capability, it has been shown[16] that on a binary symmetric channel (B.S.C.) with crossover probability P, that

$$P_{FD} < P_{DD} \quad \text{for} \quad 0 < P < \frac{1}{2}$$

where P_{FD} and P_{DD} are the probabilities of a decoding error using feedback decoding and definite decoding respectively, **and the decoder is a maximum likelihood decoder.**

10.4 Random Error-correcting Capability.[17]

We begin by defining the minimum distance of a convolutional code.

Definition 10.4.1.

The minimum distance d_m of an (n_oN, k_oN) convolutional code is equal to the smallest Hamming distance $d(\overline{u},\overline{v})$ between two initial n_oN-digit code sequences which disagree in the 0'th block.

An initial n_oN-digit code sequence is one of the 2^{k_oN} possible sequences due to the N blocks of message digits in the constraint length of n_oN digits. Let $\overline{z} = \overline{u} \oplus \overline{v}$, then $\overline{z}$ is an initial code sequence whose message blocks, other than the 0'th, are zero under modulo 2 addition since $\overline{u}$ and $\overline{v}$ only disagree in the 0'th block. Thus the minimum distance of a convolutional code is equal to the minimum weight, $w(\overline{z})$, of an initial n_oN-digit code sequence, $\overline{z}$, whose 0'th block is non-zero, written,

$$d_m(\overline{z}) = d_{min}(\overline{u},\overline{v}) = w(\overline{z}) \qquad 10.4.1.$$

We can show that any error sequence, with non-zero 0'th block, cannot prevent the correct decoding of the 0'th block providing its weight, t, is such that,

$$t \leqslant \left[\frac{d_m - 1}{2} \right]$$

That is, provided there are $\leqslant t$ errors in a span of n_oN digits, we can decode the 0'th block correctly.

To show this let $\bar{e}_1$ and $\bar{e}_2$ be two error sequences, of length n_oN digits, that disagree in the 0'th block. Then if both error sequences have weight $\left[(d_m - 1)/2 \right]$ their syndromes must be distinct, since if

$$\bar{s}_1 = \bar{e}_1 H_\infty^T = \bar{s}_2 = \bar{e}_2 H_\infty^T$$

then, $$(\bar{e}_1 \oplus \bar{e}_2) H_\infty^T = 0 \ . \qquad 10.4.2.$$

But this implies $(\bar{e}_1 \oplus \bar{e}_2)$ is a code sequence with non-zero 0'th block and therefore by the definition of a code sequence must have weight,

$$w(\bar{e}_1 \oplus \bar{e}_2) \geqslant d_m \ .$$

This is impossible by definition of the two sequences.

Thus a convolutional code with minimum distance d_m can correctly decode the 0'th block of a set of N blocks, provided there are $t \leqslant \left[(d_m - 1)/2 \right]$ errors in the span of n_oN digits, including the 0'th block itself.

10.5 Decoding Methods.

An outline is given of two decoding procedures which can be applied to any convolutional code and which are alternatives to the general decoding of section 10.2.

Viterbi-decoding.[8,19]

This is a systematic search algorithm which successively generates all 2^{k_o} code segments and compares each of these with the received code segments.

The decoder then assumes the 0'th block of the decoded sequence is equal to the 0'th block of the code word that is closest to the received word. Since approximately $2^{k_o N}$ calculations are required to decode an n_o-digit section the algorithm is limited to codes of small length $n_o N$ and low rate k_o/n_o.

The advantages are that it is a maximum-likelihood decoding procedure and performs definite decoding, thus eliminating the problem of error propagation.

Without going too deeply into Viterbi[8] decoding, which is an active branch of research on its own, it is worth noting that a parameter of interest is the "free distance" of a convolutional code. The decoding constraint length of a Viterbi decoder, $n_o M$, is several times larger than the encoding constraint length, $n_o N$. And the free distance is given by,

$$d_{free} = \lim_{M \to \infty} d_{n_o M}$$

where $d_{n_o M}$ is the minimum distance of the code with extended decoding constraint length. The probability of error is strongly dependent on d_{free}. Viterbi decoders can be used for longer codes than the simple systematic procedure outlined initially, although it is still limited to codes of moderate length.

Sequential Decoding.

Sequential decoding is an alternative to Viterbi decoding in as much as it operates on the principle that many of the 2^{k_oN} code words, [illegible] are highly improbable anyway, given the received code word. If it is possible to avoid considering highly improbable code words then the amount of computation may be manageable even for long codes.

Sequential decoding was introduced by Wozencraft.[2] and his ideas have been extended by other researchers. For a basic introduction see Peterson and Weldon[19] pp.412-425.

10.6 Majority-logic-Decoding.[1,20,7]

The majority-logic decoding algorithms in Chapter 3, are readily applicable to convolutional codes, except L-step decoding which Massey[1] showed could not be used. Also to this date no codes have been developed which utilize non-orthogonal check-sums or pseudostep orthogonalization, though there is no known restriction to their use. In a later chapter pseudostep orthogonalization is used to improve a class of codes usually decoded with the Reed-Massey[1] one-step decoding algorithm.

It is one-step decoding using the Reed-Massey algorithm which most known majority-logic convolutional codes utilize.

One-step Decoding.

The decoder of a convolutional code must store $N.(n_o-k_o)$ syndrome digits to decode the O'th block. Provided an error pattern $\overline{e}$ with $\leqslant$ t errors in a span of n_oN digits occurs, the O'th block can be one-step majority-logic decoded, if 2t orthogonal check-sums can be found on each error digit in the O'th block.

Of course the 2t check sums can be formed by any linear combinations of the $N.(n_o-k_o)$ syndrome digits. Also there is no need to correct the check-digit errors of the block being decoded, indeed from equation 10.2.4. it is difficult to see how it could be done even if one wished to do so.

There is no majority-logic equivalent of a cyclic code in the strict sense of the definition given for block codes. Thus all majority-logic codes so far devised require k_o majority-logic gates to decode a sub-block of k_o message-digits.

CHAPTER 11

11. A CLASS OF CYCLICALLY DECODABLE CONVOLUTIONAL CODES.

11.1 Introduction to Code Structure.

It was shown in Chapter 10 that a convolutional code can be completely specified by the coefficients of its sub-generators. It is the purpose of this introduction to show that a code, which is a member of the class of codes to be introduced, can be completely specified by an array of positive integers.

The codes to be presented are systematic so that the check-digits of an e'th sub-block can be represented by equation 10.1.2. given again below,

$$c_e(k_o+j) = \sum_{i=1}^{k_o} m_e(i)\ g_o(i,j) \oplus \sum_{i=1}^{k_o} m_{e-1}(i)\ g_1(i,j) \oplus \ldots\ldots$$

$$\ldots\ldots \oplus \sum_{i=1}^{k_o} m_{e-N+1}(i)\ g_{N-1}(i,j) \qquad 11.1.1.$$

for $j = 1,2,\ldots,n_o-k_o$,

where $c_e(b)$, $m_e(b)$ and $g_x(i,j)$ are as given in equation 10.1.2.

We impose the following conditions upon equation 11.1.1.

$$g_o(i,j) = \begin{cases} 0 & \text{for } i \neq a_j \\ 1 & \text{for } i = a_j \end{cases}$$
$$\vdots \qquad\qquad 11.1.2.$$
$$g_{N-1}(i,j) = \begin{cases} 0 & \text{for } i \neq n_j \\ 1 & \text{for } i = n_j \end{cases}$$

and, $1 \leqslant a_j, b_j,\ldots,h_j,\ldots,n_j \leqslant k_o$, are positive integers representing the numbers of the message-digits concerned.

We can thus rewrite equation 11.1.1. in the following form,

$$c_e(k_o+j) = m_e(a_j) \oplus m_{e-1}(b_j) \oplus \ldots\ldots \oplus m_{e-N+1}(n_j) \qquad 11.1.3.$$

where, $m_{e-x}(h_j)$ implies $g_x(h_{j,j}) = 1$.

For example,

$$c_e(k_o+1) = m_e(4) \oplus m_{e-1}(1) \oplus \ldots\ldots \oplus m_{e-N+1}(6)$$

shows that the first check-digit of block e contains the 4'th message-digit from block e, the 1'st message-digit from block e-1,........, the 6'th message-digit from block e-N+1, so that, $a_1 = 4$, $b_1 = 1, \ldots,$ $n_1 = 6$.

Consider the array below;

Array 1

$$\begin{array}{c|cccccc}
\text{rows} \backslash \text{Col's} & 0 & 1 & 2 \ldots & x & \ldots & N-1 \\
\hline
1 & a_1 & b_1 & c_1 \ldots & h_1 & \ldots & n_1 \\
2 & a_2 & b_2 & c_2 \ldots & h_2 & \ldots & n_2 \\
3 & a_3 & b_3 & c_3 \ldots & h_3 & \ldots & n_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\
i & a_i & b_i & c_i \ldots & h_i & \ldots & n_i \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\
n_o-k_o & a_{n_o-k_o} & b_{n_o-k_o} & c_{n_o-k_o} \ldots & h_{n_o-k_o} & \ldots & n_{n_o-k_o}
\end{array}$$

We can see that row i shows the numbers of the message-digits involved in check-digit $c_e(k_o+i)$, from equation 11.1.3. More specifically, from row i, the integer in column x gives the number of the message-digit from block e-x present in check-digit $c_e(k_o+i)$.

For the purposes of encoding, we can see from the array what form the encoding equations will take for the check-digits of the encoded current block in time.

From equation 11.1.3., if we vary the reference block suffix e we obtain the set of equations below.

$$\begin{aligned}
c_{e-3}(k_o+j) &= m_{e-3}(a_j) \oplus m_{e-4}(b_j) \oplus \ldots\ldots \oplus m_{e-N-2}(n_j) \\
c_{e-2}(k_o+j) &= m_{e-2}(a_j) \oplus m_{e-3}(b_j) \oplus \ldots\ldots \oplus m_{e-N-1}(n_j) \\
&\vdots \\
c_{e+x-3}(k_o+j) &= m_{e+x-3}(a_j) \oplus \ldots\ldots \oplus m_{e-3}(h_j) \oplus \ldots\ldots \oplus m_{e+x-N-2}(n_j) \\
&\vdots \\
c_{e+N-4}(k_o+j) &= m_{e+N-4}(a_j) \oplus m_{e+N-5}(b_j) \oplus \ldots\ldots \oplus m_{e-3}(n_j)
\end{aligned} \qquad 11.1.4.$$

If one traces the distribution of the message-digits from block e-3 through the equations 11.1.4. above, the following statement becomes meaningful, assuming block e-3 is the current block.
"Column x of Array 1, shows the distribution of the message-digits from the current block e-3 in the check-digits of block e-3+x."

So the array not only gives the form of the encoding equations for the check-digits of some arbitrary present block in time, it also shows how the message-digits of that block are distributed in the check-digits of itself and the N-1 blocks that will follow in time.

In section 10.2, Chapter 10, we saw that at the decoder we form the syndrome by effectively replacing a given message-digit, in the check-digit equations, by the error in that digit. This resulted in a general syndrome equation 10.2.6. which if we impose the restrictions of equations 11.1.2., becomes,

$$s_e(j) = e_e(k_o+j) \oplus e_e(a_j) \oplus e_{e-1}(b_j) \oplus \ldots\ldots \oplus e_{e-N+1}(n_j) \qquad 11.1.5.$$

Ignoring the check-digit error $e_e(k_o+j)$, this equation is given by row j of the same array of integers, bearing in mind of course, that the integers now represent errors in the message-digits.

If we consider block e is to be decoded we require the syndromes of the following N-1 blocks also, as below.

$$\begin{aligned}
s_e(j) &= e_e(k_o+j) \oplus e_e(a_j) \oplus e_{e-1}(b_j) \oplus \ldots\ldots \oplus e_{e-N+1}(n_j) \\
s_{e+1}(j) &= e_{e+1}(k_o+j) \oplus e_{e+1}(a_j) \oplus e_e(b_j) \oplus \ldots\ldots \oplus e_{e-N+2}(n_j) \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 11.1.6. \\
s_{e+x}(j) &= e_{e+x}(k_o+j) \oplus e_{e+x}(a_j) \oplus e_{e+x-1}(b_j) \oplus \ldots\ldots \oplus e_e(h_j) + \ldots\ldots \\
&\qquad \ldots \oplus e_{e+x-N+1}(n_j) \\
&\vdots \\
s_{e+N-1}(j) &= e_{e+N-1}(k_o+j) \oplus e_{e+N-1}(a_j) \oplus e_{e+N-2}(b_j) \oplus \ldots\ldots \oplus e_e(n_j)
\end{aligned}$$

But if feedback decoding is used, then all errors e_{e-x}, where $(e-x) < e$, have been cancelled. Therefore to decode some error, say $e_e(p)$, $1 \leq p \leq k_o$, we locate the integer p in each column of the array. Since $e_{e-x} = 0$ for $(e-x) < e$, from equations 11.1.6., the set of integers to the left of each integer p, gives the errors from other message-digit positions which interfere with these check sums on $e_e(p)$.

<u>Example 11.1.1</u>.

Consider the code specified by the array of integers below;

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \\ 4 & 3 & 2 & 1 \end{matrix}$$

When we encode a general block e, from equation 11.1.3. we obtain the following check-digit equations, from the rows of the array,

$$\begin{aligned} c_e(k_o+1) &= m_e(1) \oplus m_{e-1}(2) \oplus m_{e-2}(3) \oplus m_{e-3}(4) \\ c_e(k_o+2) &= m_e(2) \oplus m_{e-1}(4) \oplus m_{e-2}(1) \oplus m_{e-3}(3) \\ c_e(k_o+3) &= m_e(3) \oplus m_{e-1}(1) \oplus m_{e-2}(4) \oplus m_{e-3}(2) \\ c_e(k_o+4) &= m_e(4) \oplus m_{e-1}(3) \oplus m_{e-2}(2) \oplus m_{e-3}(1) \end{aligned} \qquad 11.1.7.$$

Since there are only four distinct integers in the array, there are only four message-digits per sub-block, so that $k_o = 4$. Also since there are four rows in the array there are four check-digits per sub-block, so that $n_o - k_o = 4$, and $n_o = 8$. This is therefore a rate 1/2 code. Only 3 other blocks are involved as well as block e, so that the constraint length of the code, in blocks, is N = 4.

From equations 11.1.4., if we increase the suffix's in equations 11.1.7., we obtain sets of check-digit equations for the other 3 blocks.

If we collect those equations involving $m_e(1)$ we obtain the following,

$$\begin{aligned} c_e(k_o+1) &= m_e(1) \oplus m_{e-1}(2) \oplus m_{e-2}(3) \oplus m_{e-3}(4) \\ c_{e+2}(k_o+2) &= m_{e+2}(2) \oplus m_{e+1}(4) \oplus m_e(1) \oplus m_{e-1}(3) \\ c_{e+1}(k_o+3) &= m_{e+1}(3) \oplus m_e(1) \oplus m_{e-1}(4) \oplus m_{e-2}(2) \\ c_{e+3}(k_o+4) &= m_{e+3}(4) \oplus m_{e+2}(3) \oplus m_{e+1}(2) \oplus m_e(1) \end{aligned} \qquad 11.1.8.$$

At the decoder these check-digits are changed to syndrome digits, the message-digits become errors from those message-digits and assuming feedback decoding, we obtain the set of equations below for the syndromes.

$$\begin{aligned} s_e(1) &= e_e(1) && \oplus e_e(k_o+1) \\ s_{e+2}(2) &= e_{e+2}(2) \oplus e_{e+1}(4) \oplus e_e(1) && \oplus e_{e+2}(k_o+2) \\ s_{e+1}(3) &= e_{e+1}(3) \oplus e_e(1) && \oplus e_{e+1}(k_o+3) \\ s_{e+3}(4) &= e_{e+3}(4) \oplus e_{e+2}(3) \oplus e_{e+1}(2) \oplus e_e(1) && \oplus e_{e+3}(k_o+4) \end{aligned}$$

Under Definition 3.6.1. these are orthogonal check sums on error digit $e_e(1)$. Note that if we locate the integer 1 in each column of the array, the leftwise sequence determines a check sum. For the check-sums to be orthogonal for all integers, we require the leftwise sequences to be unique, where this implies each element of a sequence is distinct from the corresponding element in another sequence.

The leftwise sequences for $e_e(1)$ are;

1			
1	3		
1	4	2	
1	2	3	4

Note the columns are unique as defined.

Being leftwise unique guarantees that some error from another block, will not appear more than once. Being leftwise unique also guarantees that the orthogonal check sums are inherent in the structure of the code, and as such the code is self-orthogonal. The sequences for the other message-digits are;

2				3				4			
2	1			3	4			4	2		
2	3	4		3	2	1		4	1	3	
2	4	1	3	3	1	4	2	4	3	2	1

Since we can obtain J = 4, orthogonal estimates of each error in the current block e, we can decode in the presence of $t \leqslant 2$, errors in a span of $n_o N = 32$, digits.

In the work which follows it will be shown that this array can be arranged in a 'cyclic' form such that the four information digits from a sub-block can be decoded cyclically with one majority-logic gate.

Note that if we guarantee unique leftwise sequences for some integer p, we also guarantee unique rightwise sequences, under our definition of unique. Therefore the syndrome equations obtained from equations 11.1.8. would still be self-orthogonal on $e_e(1)$ even if we did not use feedback decoding and used definite decoding instead. This is a property of all self-orthogonal codes.[12]

Before leaving this section, it is important that the reader understands exactly the information imparted by the array of integers which specifies the code. To this end the following statements directly describe the arrays' properties.

(a) If we are examining block e, then the integer in row i and column x represents both,

(i) the number of the message-digit or message-digit error from block e in check-digit i of block e + x.

(ii) the number of the message-digit or message digit error from block e − x in check-digit i of block e.

Note that if x = 0 both (i) and (ii) are identical.

11.2 Existence of Arrays.

11.2.1. Introduction to the theory.[4,5,27]

It is assumed that the reader has a knowledge of the basic properties of congruence relations. To that extent, the following brief resumé is given without proofs.

Definition 11.2.1.1.

Any two positive integers a,b whose remainder, r, upon division by some positive integer m, is the same, are said to be congruent or equivalent and can be written,

$$a \equiv b \text{ modulo } (m) \qquad 11.2.1.1$$

or

$$a \equiv b \equiv r \text{ modulo } (m).$$

11.2.1.(a)

If $a \equiv b$ modulo (m), then for any positive integer d,

$$d \cdot m + a \equiv b \text{ modulo } (m) \qquad 11.2.1.2$$

$$a + d \equiv b + d \text{ modulo } (m) \qquad 11.2.1.3$$

$$a \cdot d \equiv b \cdot d \text{ modulo } (m) \qquad 11.2.1.4$$

11.2.1.(b)

If $a \equiv b$ modulo (m), and also with c and d positive integers $c \equiv d$ modulo (m), then,

$$a + c \equiv b + d \text{ modulo } (m) \qquad 11.2.1.5$$

$$a - c \equiv b - d \text{ modulo } (m) \qquad 11.2.1.6$$

$$ac \equiv bd \quad \text{modulo } (m) \qquad 11.2.1.7$$

11.2.1.(c)

If c is a positive integer such that, $ca \equiv cb$ modulo (m), then if H.C.F. (a,b), means the highest common factor of a and b,

H.C.F. $(c,m) = d$ and $m = m_1 d$, then

$a \equiv b$ modulo (m_1),

and in particular, if

H.C.F. $(c,m) = 1$, $m = m_1$ and

$a \equiv b$ modulo (m).

In text books, the remainder is conventionally referred to as a "residue" and the set of all positive integers which have the same residue modulo (m) is called a "Residue Class".

Let $\boxed{r}$ represent the residue class containing all positive integers whose residue is r modulo (m).

We can represent the complete set of residue classes by R , where

$$\boxed{R} = \left\{ \boxed{0},\ \boxed{1},\ \boxed{2},\ \ldots\ldots,\boxed{m-1} \right\}$$

Definition 11.2.1.2.

A set of B positive integers is a complete residue system, if and only if, both of the following conditions are satisfied:

(i) B has m elements,

(ii) if a and b are contained in B, written

$a,b \in B$, and $a \equiv b$ modulo (m)

then $a = b$.

That is the set B contains one integer from each of the m residue classes contained in $\boxed{R}$.

In particular we are interested in what are known as "Reduced Residue Systems". But before we define what a reduced residue system is, we must define a function used widely in number theory and known as Eulers ϕ-function.

Definition 11.2.1.3.

For every positive integer m, the number of integers less than m and relatively prime to m, is $\phi(m)$, where

$$\phi(m) = m \prod_{p|m} \left(\frac{p-1}{p}\right)$$

where the notation indicates a product over all the distinct primes $p > 1$ which divide m.

For example;

if $m = p$, $\phi(m) = p-1$

if $m = p^{\alpha}$, $\phi(m) = p^{\alpha-1}(p-1)$

if $m = p_1^{\alpha_1} \; p_2^{\alpha_2} \; \ldots\ldots \; p_r^{\alpha_r}$

expanding $\phi(m)$, generally

$$\phi(m) = p_1^{\alpha_1-1} \; p_2^{\alpha_2-2} \; \ldots\ldots \; p_r^{\alpha_r-1} (p_1-1)(p_2-1)\ldots\ldots (p_r-1).$$

Note that;

$\phi(2m) = \phi(m)$.

Definition 11.2.1.4.

A set of S positive integers is a reduced residue system, modulo (m), if and only if, the following conditions are satisfied:

(i) S has $\phi(m)$ elements.

(ii) H.C.F. $(a,m) = 1$ for each $a \in S$

(iii) if $a,b \in S$ and $a \equiv b$ modulo (m) then $a = b$.

If two integers a,b are relatively prime, we will write, $(a,b) = 1$.

As a complete residue system, we could choose the set B, to be all m integers less than m, i.e.

$$B = \{0,1,2,\ldots, m-1\}.$$

A reduced residue system could then be all $\phi(m)$ integers $\in B$ above, that are relatively prime to m, and thus $\in S$. But in general any $\phi(m)$ integers whose residues are distinct and relatively prime to m, is a reduced residue system, as our definition implies.

The following theorems will be found useful, in later work.

Theorem 11.2.1.1.

If there exist positive integers a,b and m, such that

$$a \equiv b \text{ modulo } (m)$$

then if $(a,m) = 1$, then $(b,m) = 1$ also.

Proof.

If we assume the contrary, then there exists a positive integer $s > 1$, such that, $(b,m) = s$, (where from now on H.C.F.$(a,b) = (a,b)$, unless specified otherwise)

$$m = m_1 s \quad , \quad b = qs,$$

but since $a = \ell m + b$, then this implies

$$a = (\ell m_1 + q)s,$$

and implies s is a divisor of a, which is not possible since $(a,m) = 1$, and $s > 1$, therefore $(b,m) = 1$.

Q.E.D.

Thus if an integer's residue is relatively prime to m, then the integer itself is also relatively prime to m. Therefore any residue class, represented by $\boxed{a}$, with $(a,m) = 1$, contains a set of positive integers, which can be represented, from equation 11.2.1.2, by

$$\boxed{a} = \left\{(d \cdot m + a), \quad d = 0,1,2,\ldots\right\},$$

and $((dm + a), m) = 1$ for $d = 0,1,2,\ldots\ldots$.

<u>Theorem 11.2.1.2.</u>

If m_i, $(i = 1,2,\ldots\ldots,r)$ are positive integers, then

$$a \equiv b \text{ modulo } (m_i) \qquad i = 1,2,\ldots, r ,$$

if and only if,

$$a \equiv b \text{ modulo } (\text{L.C.M. } \{m_1,m_a,\ldots\ldots, m_r\}),$$

where, L.C.M. $\{a,b,\ldots,r\}$ means the Lowest Common Multiple of $a,b,\ldots,r$.

11.2.2. The Array.

In section 11.1. we showed that a code could be specified by an array of positive integers representing the numbers of message-digits or message-digit errors. We also saw that the array must be structured so that the leftwise sequences of any integer must form a unique set in order that orthogonal check sums exist for that integer. This condition must hold for every distinct integer in the array if a code is to be specified by the array.

In this section we develop bounds and conditions on the form of the array which guarantees unique leftwise sequences for all distinct integers. The following theorem determines the form of a single row of an array.

Theorem 11.2.2.1.[28]

If there exists positive integers a,m such that $(a,m) = 1$, then the quantities

$$a,\ 2a,\ 3a,\ldots..,\ (m-1)a$$

have $(m-1)$ distinct residues modulo (m).

Since for any integer m, there are $\phi(m)$ integers a_i, $(i = 1,2,\ldots,\phi(m))$, with $(a_i,m) = 1$, we can obtain $\phi(m)$ sets, S_i, where

$$S_i = \left\{a_i \bmod(m),\ 2a_i \bmod(m),\ldots,(m-1)a_i \bmod(m)\right\}$$

A general array (A) is defined as the set of sets, S_i, $i = 1,2,\ldots, \phi(m)$, where

$$(A) = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{\phi(m)} \end{bmatrix} \qquad 11.2.2.1.$$

and each S_i constitutes a row of the array which therefore has $\phi(m)$ rows, $(m-1)$ columns and $(m-1) \cdot \phi(m)$ elements.

It will be seen later that it is possible to construct arrays with $S < \phi(m)$ rows provided S divides $\phi(m)$, written $S|\phi(m)$. However a row of any array always contains the complete set of elements

1,2,...,(m-1), though in different arrangements or permutations.

From section 11.1. we saw that to obtain self-orthogonal estimates on a particular error-digit, we require the leftwise sequences of that integer representing the error, to be unique. See Example 11.1.1. and equations 11.1.6.

Consider the example below.

Example 11.2.2.1.

The array below is generated mod (8) by the two integers 5 and 7, relatively prime to 8.

$$\begin{array}{ccccccc} 5 & 2 & 7 & 4 & 1 & 6 & 3 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{array}$$

All leftwise sequences, for all integers except 3, are unique as defined. For 3 we obtain,

$$\begin{array}{ccccccc} 3 & 6 & 1 & 4 & 7 & 2 & 5 \\ 3 & 4 & 5 & 6 & 7 & & \end{array}$$

Assuming we are considering errors in the current block as block 0, the syndrome equations from example 11.1.1. are,

$$s_6(1) = e_6(8) \oplus e_0(3) \oplus e_1(6) \oplus e_2(1) \oplus e_3(4) \oplus e_4(7) \oplus e_5(2) \oplus e_6(5)$$

$$s_4(2) = e_4(9) \oplus e_0(3) \oplus e_1(4) \oplus e_2(5) \oplus e_3(6) \oplus e_4(7) ,$$

and these are orthogonal on,

$$e_0(3) \oplus e_4(7) .$$

The situation in example 11.2.2.1. above must be avoided if we are to decode all digits and we can gain some insight into how this can be done by considering what conditions caused the results in the above example.

Let, $a_1 = 5$ and $a_2 = 7$ then the two equations $s_6(1)$ and $s_4(2)$ are determined by the following equations from the array of integers,

$$7 + 4\cdot(5) \equiv 3 \bmod (8)$$

from the row generated by a_1.

$$7 + 4\cdot(7) \equiv 3 \bmod (8)$$

from the row generated by a_2.

So that 7 and 3 are separated by 4 multiples of a_1 and a_2. We can represent this situation generally by the equations below.

$$\begin{aligned} x + n(a_1) &\equiv v \bmod (m) \\ x + n(a_2) &\equiv v \bmod (m) \end{aligned} \qquad 11.2.2.2.$$

Obviously n must be an integer and from these equations,

$$x - v = q\cdot m - n(a_1)$$

$$x - v = r\cdot m - n(a_2)$$

so that, assuming $a_2 > a_i$, $r > q$

$$m(r - q) = n(a_2 - a_1)$$

$$n = \frac{m(r - q)}{(a_2 - a_1)} \qquad 11.2.2.3.$$

where n must be an integer. When this equation is satisfied we will call the condition "Array row equivalence" (or A.R.E.). We can obtain a maximum for n by considering x to be in the first and second columns of the array, with v in the next to last and last columns respectively.

That is, if x is in the first column of row one, $x = a_1$ and also since v, in row one, is in the next to last column, giving

$$(m - 2)a_1 \equiv v \bmod (m)$$

therefore, from equation 11.2.2.2. with $x = a_1$

$$a_1 + n \cdot a_1 = (m-2)\ a_1$$

and $$n_{max} = (m-3). \qquad 11.2.2.4.$$

The same result is derived from the second row also. We can also determine some useful condition on $(r - q)_{max}$ in the following way.

From equation 11.2.2.3,

$$(r-q) = \frac{n(a_2 - a_1)}{m}$$

and since (a_2-a_1) and m are fixed, $(r-q)$ is determined by n. Thus,

$$(r-q)_{max} = \frac{n_{max}(a_2-a_1)}{m}$$

$$= \frac{(m-3)(a_2-a_1)}{m} \quad .$$

Therefore,

$$(r-q)_{max} < (a_2-a_1) \qquad 11.2.2.5.$$

and if $n < n_{max}$, $(r-q) < (r-q)_{max}$ for all n.

We are now in a position to prove the following theorem.

Theorem 11.2.2.2.

Any array generated mod (m), by a set of relative primes a_i, where $(a_i,m) = 1$, $a_i < m$, is free from A.R.E., if all positive differences (a_j-a_i), $a_j > a_i$, are also relatively prime to m.

Proof.

From equation 11.2.2.3., if A.R.E. occurs

$$n = \frac{m(r-q)}{(a_j-a_i)}$$

must be an integer.

But from inequality 11.2.2.5.,

$$(r-q) < (a_j-a_i)$$

for all n, and since $((a_j-a_i),m) = 1$, for n to be an integer (a_j-a_i) must wholly divide $(r-q)$, which is not possible.

Q.E.D.

In example 11.1.1. the array was generated by the set of integers 1,2,3 and 4, mod (5), and any difference is also relatively prime to 5.

In example 11.2.2.1. the array was generated by the integers 5 and 7, mod (8), but their difference, $(7-5) = 2$, is not relatively prime to 8 and thus A.R.E. occurs.

If (a_j-a_i) is not relatively prime to m, then, if m_1 is the greatest common divisor,

$$((a_j-a_i),m) = m_1$$

and the following theorem shows that column duplication occurs under these circumstances.

<u>Theorem 11.2.2.3.</u>

If any array is generated mod (m), by two integers a_j and a_i, such that,

$$(a_j,m) = 1 \quad , \quad (a_i,m) = 1 \quad \text{and}$$

$$(a_j-a_i) = bm_1 \quad , \quad a_j > a_i$$

where b is an arbitrary positive integer and m_1 is some divisor of m, then column duplication will occur in the array.

Proof.

Assume that at the h'th column of the array, we have,

$$h\ a_i = q_1 m + r' \qquad r' < m$$

$$h\ a_j = q_2 m + r'' \qquad r'' < m$$

then, subtracting,

$$h(a_j - a_i) = m(q_1 - q_2) + r' - r'' \qquad 11.2.2.6(a)$$

but if $(a_j - a_i) = b\ m_1$

$$h\ b\ m_1 = m(q_1 - q_2) + r' - r'' \qquad 11.2.2.6(b)$$

since h can assume any value between 1 and its maximum (m-3), if

$$h = \frac{dm}{m_1} \quad , \quad d = 1,2,\ldots,\ g < m_1$$

$$= \frac{m}{m_1} \ , \ \frac{2m}{m_1} \ ,\ldots,\ \frac{gm}{m_1} < m$$

then equation 11.2.2.6., becomes,

$$dbm = m(q_1 - q_2) + r' - r''$$

but since

$r' < m$ and $r'' < m$

then $(r' - r'') < m$ and therefore

$r' = r''$

and column h has the same integer values at all $h = \frac{dm}{m_1}$, causing column duplication.

Q.E.D.

Two results are now immediately obvious;

i) if $m_1 > 2$, column duplication must occur in more than one column. It is also obvious that column duplication in more than one column causes A.R.E. This eliminates pairs of relative primes with this property.

ii) if $m_1 = 2$, then since $d < m_1$, column duplication can only occur in one column

$$h = \frac{m}{m_1} = \frac{m}{2} ,$$

that is the centre column. However this can only occur if m is an even integer and is impossible if m is an odd integer. This will be examined in more detail later.

We can now prove that whether or not column duplication occurs, all other columns contain distinct integers.

From equation 11.2.2.6(a) and theorem 11.2.2.3., if two rows of an array contain the same integer, in column h, $1 \leq h \leq m-1$, $r' = r''$ and

$$h(a_j - a_i) = m(q_1 - q_2)$$

$$h = \frac{m(q_1 - q_2)}{(a_j - a_i)} \quad .$$

There are two cases to consider;

i) if $((a_j - a_i), m) = 1$, this cannot occur for any integer value of h and all columns contain distinct integers.

ii) if $((a_j - a_i), m) = m_1$, then $(a_j - a_i) = bm_1$

$$h = \frac{m}{m_1} \frac{(q_1 - q_2)}{b}$$

but since $(b,m) = 1$, for h to be an integer b must wholly divide $(q_1 - q_2)$.

Let $(q_1 - q_2) = cb$, $c = 1,2,3,\ldots\ldots$

Regardless of the value of c, h will be a multiple of m/m_1, which is a column where duplication is known to occur. Thus all other columns contain distinct integers.

Example 11.2.2.2.

Let $m = 9$, then the set of relative primes is;

$$\{1, 2, 4, 5, 7, 8\}.$$

Let $a_j = 8$ and $a_i = 2$, then

$$(a_j - a_i) = (8 - 2) = 6 = 2 \cdot 3$$

therefore $m_1 = 3$ and $b = 2$ and column duplication should occur in columns that are multiples of,

$$h = \frac{m}{m_1} = \frac{9}{3} = 3$$

that is in columns,

$$h = d \cdot 3 \quad , \quad d = 1,2,\ldots, t < 3$$

$$= 3 \text{ and } 6.$$

The two rows of the array are,

```
2 4 6 8 1 3 5 7
8 7 6 5 4 3 2 1
```

and column duplication in columns 3 and 6 verified.

Therefore when decoding digit 3, of say block 0, we will obtain two syndrome equations orthogonal on $e_0(3) \oplus e_3(6)$, see example 11.1.1. Note, all other columns contain distinct integers.

However this does not mean that the type of A.R.E. indicated in example 11.2.2.1. does not occur when column duplication occurs, it can occur simultaneously with column duplication.

In fact A.R.E. also occurs for digits 1 and 5 as we can see from the leftwise sequences for these,

1	8	6	4	2			
1	2	3	4	5	6	7	8
5	6	7	8				
5	3	1	8	6	4	2	

And the above sequences represent syndrome equations orthogonal on $e_0(1) \oplus e_3(4)$ and $e_0(5) \oplus e_3(8)$ respectively.

Before examining the cases when m is odd and even, the following theorem establishes an upper bound on the number of relative primes one can utilize, to be sure that their differences are also relatively prime to m.

Theorem 11.2.2.4.

Given any integer m,

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots\cdot p_r^{\alpha_r}$$

$$p_1 = \text{smallest prime factor} > 1.$$

Then, the maximum number of relative primes $a_i < m$, one can choose, such that their differences, $(a_j - a_i)$, are also relatively prime to m, is upper bounded by $(p_1 - 1)$.

Proof.

Any difference, not relatively prime satisfies

$$(a_j - a_i) = bp_i$$

for some p_i, divisor of m.

Then $\quad a_j = bp_i + a_i$

or $\quad a_j \equiv a_i \bmod (p_i)$.

To guarantee this does not occur we require

$$a_j \not\equiv a_i \bmod (p_i)$$

for any divisor p_i of m. That is the set must have distinct residues modulo any divisor of m. Since p_1 is the smallest divisor of m, the set cannot exceed (p_1-1) in number otherwise, reducing mod (p_1) will produce equal residues for some numbers.

Q.E.D.

Almost as important as the upper bound and of great use later is the fact that the proof of theorem 11.2.2.4. shows that we can determine if a set of relative primes has relatively prime differences by determining if the set has distinct residues modulo (any divisor of m).

We can now examine how the preceding theorems affect arrays when m is odd and even.

m = ODD.

When m is an odd integer then its smallest divisor must be $\geqslant 3$. From theorem 11.2.2.3, if an array generated mod (m), has rows generated by relative primes $a_i < m$, such that $(a_i - a_j) = bm_1$, then column duplication will occur in columns,

$$h = \frac{dm}{m_1} \quad , \quad d = 1,2,\ldots., \; g < m_1$$

but since m is odd, $m_1 \geqslant 3$, we will have

$$h = \frac{m}{m_1} \; , \; \frac{2m}{m_1} \; , \; \ldots\ldots.$$

and column duplication will always occur in at least two columns.

An example follows, with m = 21, which is more illustrative than that of example 11.2.2.2. when m = 9.

<u>Example 11.2.2.3.</u>

Let $m = 3 \cdot 7 = 21$, then the set of relative primes, A, is

$$A = \{1,\ 2,\ 4,\ 5,\ 8,\ 10,\ 11,\ 13,\ 16,\ 17,\ 19,\ 20\}$$

Let, $a_3 = 8, \quad a_2 = 11, \quad a_1 = 4$

from $(a_2-a_3) = 11 - 8 = 3.$

$b_1 = 1, \quad m_1 = \underline{\underline{3}}$

and $\frac{dm}{m_1} = \frac{21}{3} \quad , \quad \frac{42}{3} = \underline{\underline{7, \ 14,}}$

duplication between rows 8 and 11 occurs in columns 7 and 14.

from $(a_2-a_1) = 11 - 4 = \underline{\underline{7}}$

$b_2 = 1, \quad m_1 = 7$

and $\frac{dm}{m_1} = \frac{21}{7}, \ \frac{42}{7}, \ \frac{63}{7}, \ \frac{84}{7}, \ \ldots\ldots.$

$= \underline{\underline{3, \ 6, \ 9, \ 12, \ 15, \ 18,}}$

and duplication between rows 11 and 4 occurs in six columns, 3, 6, 9, 12, 15, 18.

The array produced is

8	16	3	11	19	6	14	1	9	17	4	12	20	7	15	2	10	18	5	13
11	1	12	2	13	3	14	4	15	5	16	6	17	7	18	8	19	9	20	10
4	8	12	16	20	3	7	11	15	19	2	6	10	14	18	1	5	9	13	17

Duplication also occurs between rows generated by the following pairs of relative primes.

(1,4),(2,5),(5,8),(8,11),(2,8),(1,8),(4,10),(1,10),(10,13),(4,13), (1,13),(10,16),(13,16),(4,16),(2,16),(1,16),(11,17),(10,17),(8,17),(5,17) (2,17),(16,19),(13,19),(10,19),(5,19),(4,19),(1,19),(17,20),(13,20),(11,20), (8,20),(5,20),(2,20).

Note, all other columns contain distinct integers.

We have shown that if m is an odd integer, any array generated mod (m), which contains two rows, generated mod (m), by two relative primes a_j and a_i, such that,

$$((a_j - a_i), m) \neq 1$$

is an array with A.R.E. between these rows.

This along with theorem 11.2.2.2. proves the following theorem.

Theorem 11.2.2.5.

If m is an odd positive integer, then any array generated mod (m), by a set of relative primes $\{a_i\}$, such that $a_i < m$, $(a_i, m) = 1$ for each i, is free from A.R.E., if and only if, all differences $(a_j - a_i)$ are also relatively prime to m, where $a_j > a_i$.

Therefore the upper bound imposed on arrays generated by relative primes of an odd integer, is given completely by theorem 11.2.2.4.

As an Example of an array free from A.R.E. consider the following.

Example 11.2.2.4.

Let m = 7, then the set of relative primes is,

$$\{1, 2, 3, 4, 5, 6\}$$

and all differences, from this set, are also relatively prime to 7, and we obtain

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 1 & 3 & 5 \\ 3 & 6 & 2 & 5 & 1 & 4 \\ 4 & 1 & 5 & 2 & 6 & 3 \\ 5 & 3 & 1 & 6 & 4 & 2 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{array}$$

Six self-orthogonal equations (or unique leftwise sequences) are obtainable on all six distinct integers. Thus the array specifies a convolutional code, with

$k_o = 6$

$n_o = 12$

$N = 6$

$t = 3$.

We will show later how to arrange this array in a 'cyclic' form so that all six digits can be decoded cyclically with one majority-logic gate.

Since all differences are relatively prime all columns contain distinct integers.

m = EVEN.

All relative primes of an even integer are odd and therefore any difference (a_j-a_i) is divisible by 2 and cannot be relatively prime to m. This follows from theorem 11.2.2.4. We know from theorem 11.2.2.2. if

$$(a_j-a_i) = bm_1 \quad , \quad m_1 > 2$$

then column duplication occurs in more than one column and so A.R.E. exists.

We can therefore restrict our enquiries to subsets of relative primes, where,

$$(a_j-a_i) = b\cdot 2 \quad , \quad \text{where } \left(b, \frac{m}{2}\right) = 1 \ .$$

If A.R.E. occurs under these conditions, then, from equation 11.2.2.3.,

$$n = \frac{m(r-q)}{2\cdot b} \qquad 11.2.2.7.$$

and since $\left(b,\frac{m}{2}\right) = 1$, b must wholly divide (r-q) if n is an integer.

That is, $(r-q) = cb$, $c = 1,2,\ldots..$

but if $c > 2$, $n > m$ which is impossible,

if $c = 2$, $n = m$ which is impossible, and therefore, $c = 1$, $(r-q) = b$ and $n = m/2$ is the only value for which A.R.E. can occur. We know from theorem 11.2.2.3. that column duplication will occur in the centre column, but this alone will not cause A.R.E. on the duplicated integer. However from example 11.2.2.1., in which column duplication occurred only in the centre column, A.R.E. still occurred. We must determine under what conditions it occurs and if there are any conditions under which the array can be free from A.R.E.

Returning to the original equations under which A.R.E. was established we have,

$$x + na_i \equiv v \bmod (m)$$

$$x + na_j \equiv v \bmod (m)$$

But these equations take no account of the columns the two x's occur in. Let us assume that the x in the row generated by a_i occurs in column g, and the x in the row generated by a_j occurs in column f, then we have,

$$\left.\begin{array}{l} ga_i \equiv x \bmod (m) \\ fa_j \equiv x \bmod (m) \end{array}\right\} \qquad 11.2.2.8.$$

We can obtain a maximum value for g, since we know for A.R.E. to occur n must equal m/2. Let us write,

$$\left.\begin{array}{l} ga_i + na_i = (m-2)\ a_i \\ fa_j + na_j = (m-1)\ a_j \end{array}\right\} \qquad 11.2.2.9.$$

so that, $g < f$, giving

$$(g+n) = (m-2)\ , \quad \text{and}$$

$$g_{max} = m - 2 - \frac{m}{2}$$

$$= \frac{m}{2} - 2\ . \qquad 11.2.2.10.$$

Also from equation 11.2.2.9.,

$$(f+n) = (m-1) \quad , \quad \text{so that}$$

$$f_{max} = m - 1 - \frac{m}{2}$$

$$= \frac{m}{2} - 1 \qquad 11.2.2.11.$$

That is both x's must be located left of the centre column, where duplication occurs. To the left (or right) of the centre column there are (m/2) - 1 columns and the following theorem shows that if two x's exist left of centre, then two v's always exist m/2 columns away from both x's, right of centre causing A.R.E. .

Theorem 11.2.2.6.

If m is an even positive integer and an array is generated mod (m) by a set of relative primes a_i, such that any difference $(a_j - a_i) = 2 \cdot b$, with $(b,m) = 1$, then A.R.E. will occur, if and only if, there is an integer x, such that, with $g < f$

$$\begin{aligned} ga_i &\equiv x \bmod (m) \\ fa_j &\equiv x \bmod (m) \end{aligned} \qquad 11.2.2.12(a)$$

with both,

$$g \leqslant \frac{m}{2} - 2 \quad \text{and} \quad f \leqslant \frac{m}{2} - 1 \; .$$

Proof.

Let us assume there exists an integer r' in column g of row a_i and an integer r" in column f of row a_j, where $g \leqslant \frac{m}{2} - 2$, $f \leqslant \frac{m}{2} - 1$ then,

$$ga_i \equiv r' \bmod (m) \quad , \quad r' < m$$

$$fa_j \equiv r'' \bmod (m) \quad , \quad r'' < m$$

and if $f > g$

$$fa_j - ga_i = m(q_2-q_1) + r'' - r'$$

but if $(a_j-a_i) = 2b$, then

$$a_j = 2b + a_i \quad \text{and}$$

$$f(2b+a_i) - ga_i = m(q_2-q_1) + r'' - r'$$

$$2bf + a_i(f-g) = m(q_2-q_1) + r'' - r' \qquad 11.2.2.12(b)$$

If we move $n = \frac{m}{2}$ columns down the array, then we will have,

$$(g + \tfrac{m}{2})\, a_i = q_3m + p' \quad , p' < m$$

$$(f + \tfrac{m}{2})\, a_j = q_4m + p'' \quad , p'' < m \qquad 11.2.2.13.$$

where,

$$r' + \frac{m}{2}\, a_i \equiv p' \bmod (m)$$

$$r'' + \frac{m}{2}\, a_j \equiv p'' \bmod (m)$$

Subtracting equations 11.2.2.13.,

$$fa_j - ga_i + \frac{m}{2}\,(a_j-a_i) = m(q_4-q_3) + p'' - p'$$

and since $(a_j-a_i) = 2b$

$$fa_j - ga_i = m(q_4-q_3-b) + p'' - p'$$

$$2bf + a_i\,(f-g) = m(q_4-q_3-b) + p'' - p' \qquad 11.2.2.14.$$

Comparing equations 11.2.2.12(b) and 11.2.2.14,

$$m(q_2-q_1) + r'' - r' = m(q_4-q_3-b) + p'' - p'$$

and since $(r'' - r')$ and $(p'' - p')$ are less than m, we must have,

$$(q_2-q_1) = (q_4-q_3-b) \quad , \quad \text{and}$$

$$r'' - r' = p'' - p'.$$

Let $r' = x$ and $p' = v$, then if

$r'' = r' = x$, we also have,

$p'' = p' = v$ and A.R.E. occurs.

However this assumes g and f are less than their maximums, given in equations 11.2.2.10. and 11.2.2.11. respectively.

If we assume, $f > \frac{m}{2} - 1$, then

$$(f + \frac{m}{2}) > (m-1)$$

and since there are only (m-1) columns in the array, v, cannot exist in the row generated by a_i and A.R.E. cannot occur.

Also if $g > \frac{m}{2} - 2$, then this also implies $f > \frac{m}{2} - 1$ and again A.R.E. cannot occur.

Q.E.D.

Therefore to prevent A.R.E. we require all integers left of the centre column to be distinct and this obviously places a limitation on the size of the array.

The following example illustrates the results of theorem 11.2.2.6.

<u>Example 11.2.2.5.</u>

Let $m = 2 \cdot 5 = 10$, then the set of $\phi(10) = 4$, relative primes are

$$A = \{1, 3, 7, 9\}.$$

Let $a_1 = 1$, $a_2 = 3$, from equation 11.2.2.12(a)

therefore $\quad 2 \cdot 1 \equiv 2 \bmod (10)$

$4 \cdot 3 \equiv 2 \bmod (10)$

thus $g = 2$, $f = 4$, $\leq \frac{m}{2} - 2$, $\frac{m}{2} - 1$ respectively.

A.R.E. occurs when $n = \frac{m}{2} = 5$.

$$2 + 5 \cdot 1 \equiv 7 \bmod (10)$$

$$2 + 5 \cdot 3 \equiv 7 \bmod (10)$$

The full array is shown below.

1	2	3	4	5	6	7	8	9
3	6	9	2	5	8	1	4	7
7	4	1	8	5	2	9	6	3
9	8	7	6	5	4	3	2	1

A.R.E. occurs between rows (1,3), (1,7), (3,9), (7,9). Note that no A.R.E. occurs between rows (3,7) and (1,9) since all integers to the left of centre column are distinct.

Let $a_1 = 3$, $a_2 = 7$, and

$$\left.\begin{aligned} 6\cdot 3 &\equiv 8 \bmod (10) \\ 4\cdot 7 &\equiv 8 \bmod (10) \end{aligned}\right\} \qquad 11.2.2.15.$$

thus $g = 4 > \frac{m}{2} - 2 = g_{max}$

and $f = 6 > \frac{m}{2} - 1 = f_{max}$

and $(g + \frac{m}{2}) = 9$, $(f + \frac{m}{2}) = 11$

but column 11 does not exist.

Alternatively,

$$\left.\begin{aligned} 8\cdot 3 &\equiv 4 \bmod (10) \\ 2\cdot 7 &\equiv 4 \bmod (10) \end{aligned}\right\} \qquad 11.2.2.16.$$

thus $g = 2 < \frac{m}{2} - 2 = g_{max}$

and $f = 8 > \frac{m}{2} - 1 = f_{max}$

but though $(g+n) = 7$, we have $(f+n) = 13$, and column 13 does not exist.

In equations 11.2.2.15 both g and f exceed their maximums, but in equations 11.2.2.16. only f exceeds its maximum. Yet A.R.E. does not occur in both cases.

From the construction of an array we know that each row contains (m-1) distinct integers, of which (m/2)-1 are left of centre, and if we assume the array has S rows, then to the left of centre we require

$$S\left(\frac{m}{2} - 1\right)$$

distinct integers. Since there are only (m-1) distinct integers in the whole array this limits S to 2, or, (m-2) distinct integers left (and right) of the centre column which contains the (m-1)'th duplicated integer.

Consider the following theorem.

Theorem 11.2.2.7.

If m is an even positive integer, any array generated mod (m) by two relative primes a_i and a_j, is free from A.R.E. if $(a_i+a_j) = m$.

Proof.

Since, for A.R.E. to occur,

$$\begin{aligned} ga_i &\equiv x \bmod (m) \\ fa_j &\equiv x \bmod (m) \end{aligned} \qquad 11.2.2.17$$

where

$$g \leq \frac{m}{2} - 2$$

and

$$f \leq \frac{m}{2} - 1$$

then from equations 11.2.2.17.

$$ga_i \equiv fa_j \bmod (m)$$

$$ga_i = km + fa_j$$

but if $a_j + a_i = m$, then

$a_j = m - a_i$ and

$$ga_i = km + f(m-a_i)$$

$$m(f+k) = a_i(g+f)$$

and since $(a_i,m) = 1$, m must be a factor of $(g+f)$.

Let, $(g+f) = cm$ $c = 1,2,\ldots\ldots$

if $c \geqslant 1$, (i) g or f are $> \frac{m}{2}$

or (ii) $g = f = \frac{m}{2}$

and in both cases at least one of them exceeds their maximum and A.R.E. cannot occur.

Q.E.D.

It will be seen later that a 'cyclic' array always exists generated by a_j and a_i such that $(a_j+a_i) = m$.

The following example gives a general solution, when $m = 14$.

Example 11.2.2.4.

Let $m = 2\cdot 7 = 14$, then the set of $\phi(14) = 6$, relative primes are,

$$A = \{1, 3, 5, 9, 11, 13\}$$

Let $a_1 = 1$, $a_2 = 3$, from equation 11.2.2.12(a)

$$4\cdot 1 \equiv 4 \bmod (14)$$

$$6\cdot 3 \equiv 4 \bmod (14)$$

thus $g = 4$, $f = 6$, $\leqslant \frac{m}{2} - 2$, $\frac{m}{2} - 1$, respectively.

A.R.E. occurs when $n = \frac{m}{2} = 7$,

$$4 + 7\cdot 1 \equiv 11 \bmod (14)$$

$$4 + 7\cdot 3 \equiv 11 \bmod (14)$$

The full array is shown below.

1	2	3	4	5	6	7	8	9	10	11	12	13
3	6	9	12	1	4	7	10	13	2	5	8	11
5	10	1	6	11	2	7	12	3	8	13	4	9
9	4	13	8	3	12	7	2	11	6	1	10	5
11	8	5	2	13	10	7	4	1	12	9	6	3
13	12	11	10	9	8	7	6	5	4	3	2	1

Many more equivalences occur, other than those shown, but A.R.E. cannot be present between rows (1,13), (3,11) and (5,9), since $(1 + 13) = (3 + 11) = (5 + 9) = 14 = m$.

From rows $a_1 = 5$, $a_2 = 9$,

$$10 \cdot (5) \equiv 8 \bmod (14)$$

$$4 \cdot (9) \equiv 8 \bmod (14)$$

thus $g = 4 < \frac{m}{2} - 2$, but

$f = 10 > \frac{m}{2} - 1$

and $(g + \frac{m}{2}) = 11$, $(f + \frac{m}{2}) = 17$

and column 17 does not exist. The arrays generated by the pairs 1,13 or 3,11 have a similar result and are free from A.R.E.

To summarize the results of this section, we have shown.

i) if m is an odd integer, its relative primes $a_i < m$, generate an array free from A.R.E., if and only if, all differences $(a_j - a_p)$ are relatively prime to m.

ii) if m is an odd integer, the number of rows in an array free from A.R.E., generated by relative primes $a_i < m$, is upper bounded by $(p_1 - 1)$, where p_1 is the smallest prime divisor of m.

iii) the set of $A = \left\{a_1, a_2, \text{---}, a_{p_1 - 1}\right\}$ integers relatively prime to m, m = odd, generate an array free from A.R.E., if and only if, the set A has distinct residues modulo (any divisor of m).

iv) if m is an even integer, an array free from A.R.E., has a maximum number of 2 rows and is generated by any pair of relative primes a_i, a_j, if $(a_i + a_j) = m$.

11.3 Existence of 'Cyclic' Arrays.

11.3.1. Introduction to the theory.[4,5,27]

We introduced in the previous section the concept of reduced residue systems. We then examined the arrays they produced and developed some fundamental properties, which guaranteed the arrays are free from array row equivalence (A.R.E.).

We will now continue our study of reduced residue systems and the following theorems immediately are illuminating. The bulk of the material in this section can be found in references 4 and 5.

Theorem 11.3.1.1. (Eulers Theorem)

If $(a,m) = 1$, then

$$a^{\phi(m)} \equiv 1 \text{ modulo } (m).$$

This does not imply that $\phi(m)$ is the only power of a, which gives a residue of one, it merely states that any relative prime raised to the power $\phi(m)$, has a residue of one, modulo (m).

Definition 11.3.1.1.

If $(a,m) = 1$, the smallest positive integer S, such that,

$$a^S \equiv 1 \text{ modulo } (m),$$

is called "the order of a modulo (m)".

Since every integer a, has $\phi(m)$ as a power with a residue of one, then since S is the smallest integer which satisfies our definition above, then

$$\text{Ord}_m\, a \leqslant \phi(m)$$

where the notation means, the order of a, modulo (m), is less than or equal to $\phi(m)$.

We also have;

Theorem 11.3.1.2.

If the order of a is S mod (m) then S divides $\phi(m)$.

Theorem 11.3.1.3.

If the order of a is S modulo (m), and k is a positive integer, then

$$a^k \equiv 1 \text{ modulo } (m)$$

if and only if, S divides k, written $S|k$.

Thus no power of a relative prime can give a residue of one, uless it is a multiple of the order of the relative prime.

Theorem 11.3.1.4.

If $a \equiv b$ modulo (m), then for any positive integer k,

$$a^k \equiv b^k \text{ modulo } (m).$$

If in the above theorem a has order S modulo (m), then

$$a^S \equiv b^S \equiv 1 \text{ modulo } (m)$$

and we have;

Theorem 11.3.1.5.

If $a \equiv b$ modulo (m), then a and b have the same order modulo (m).

Therefore, for all positive integers that are members of the same residue class, since any two of them say x and y, are related by

$$x \equiv y \text{ modulo } (m),$$

then they all have the same order modulo (m).

Theorem 11.3.1.6.

If $(a,m) = 1$, and $\text{Ord}_m\, a = S$, then the set of elements,

$$\{a, a^2, a^3, \ldots, a^S\}$$

have distinct residues modulo (m) and are said to be "incongruent modulo (m)".

So, amazingly, if one raises a relative prime to all positive powers less than and equal to its order, one immediately obtains a set of integers from distinct residue classes. This leads to the following theorem.

Theorem 11.3.1.7.

If $(a,m) = 1$, and $\text{Ord}_m\ a = \phi(m)$, then the set of elements,

$$\{a, a^2, a^3, \ldots\ldots, a^{\phi(m)}\}$$

is a reduced residue system, modulo (m).

This follows from Theorem 11.3.1.6. and our definition of a reduced residue system. We wish at this point to introduce a definition of our own.

Definition 11.3.1.2.

If $(a,m) = 1$, and $\text{Ord}_m\ a = S$, then the set of elements,

$$\{a, a^2, a^3, \ldots\ldots, a^S\}$$

will be called a 'cyclic' set of order S modulo (m).

This follows from the fact that since

$a^S \equiv 1$ modulo (m), then

$a^{S+1} \equiv a \bmod (m)$

where mod (m) = modulo (m), and

$a^{S+2} \equiv a^2 \bmod (m)$, until

$\vdots$

$a^{2S} \equiv 1 \bmod (m)$

Therefore the set $\{a^{S+1}, a^{S+2}, \ldots., a^{2S}\}$ is also a 'cyclic' set of order S, and

$$\text{Ord}_m\ a = \text{Ord}_m\ a^{S+1} = S.$$

Any relative prime, a, that has order $\phi(m)$ mod (m) is given a special name, defined below.

Definition 11.3.1.3.

If $(a,m) = 1$, and $\mathrm{Ord}_m\, a = \phi(m)$, then a is called a primitive root of m.

It will be perhaps helpful if at this point we illustrate the preceding theorems with an example.

Example 11.3.1.1.

Let $m = 11$, then the set of relative primes, less than m, which represent its residue classes is,

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

We omit the integer zero since it represents the residue of integers that are multiples of m, and are therefore not relatively prime.

We then have the following 'cyclic' sets reduced mod (11).

i) $1^2 \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\, 1 = 1$

ii) $\{2, 4, 8, 5, 10, 9, 7, 3, 6, 1\}$

and $2^{10} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\, 2 = 10$.

iii) $\{3, 9, 5, 4, 1\}$

and $3^5 \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\, 3 = 5$.

iv) $\{4, 5, 9, 3, 1\}$

and $4^5 \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\, 4 = 5$.

v) $\{5, 3, 4, 9, 1\}$

and $5^5 \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\, 5 = 5$.

iv) $\{6, 3, 7, 9, 10, 5, 8, 4, 2, 1\}$.

and $6^{10} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\ 6 = 10$.

vii) $\{7, 5, 2, 3, 10, 4, 6, 9, 8, 1\}$

and $7^{10} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\ 7 = 10$.

viii) $\{8, 9, 6, 4, 10, 3, 2, 5, 7, 1\}$

and $8^{10} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\ 8 = 10$.

ix) $\{9, 4, 3, 5, 1\}$.

and $9^{5} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\ 9 = 5$.

x) $\{10, 1\}$.

and $10^{2} \equiv 1 \bmod (11)$, $\mathrm{Ord}_{11}\ 10 = 2$.

So we have a set of orders,

$\{1, 2, 5, 10\}$.

From definition 11.2.1.3.

$\phi(m) = 10$,

and theorems 11,3.1.1., 11.3.1.2., 11.3.1.6., 11.3.1.7. can be seen to be satisfied.

In particular, the elements 2, 6, 7, and 8 are primitive roots of 11.

We can now clarify Theorem 11.3.1.5.

If

$\mathrm{Ord}_m\ a = \mathrm{Ord}_m\ b$, this does not imply that

$a \equiv b \bmod (m)$,

otherwise any positive integer would only have at most one primitive root, which is not so, as the following theorem shows.

Theorem 11.3.1.8.

If there exists a primitive root modulo (m), then there are precisely

$$\phi(\phi(m))$$

primitive roots mod (m), which are incongruent mod (m).

From the above example there are four primitive roots and $\phi(\phi(m)) = \phi(10) = 4$.

The preceding theorem began "If there exists a primitive root", and this is because, not all positive integers have primitive roots.

Which integers do have primitive roots is well established and given in the following theorem.

Theorem 11.3.1.9.

A positive integer $m > 1$, has a primitive root, if and only if, m is one of the following

$$2,\ 4,\ p^k,\ 2p^k$$

where p is any odd prime and k any positive integer.

From theorem 11.3.1.7. and our example 11.3.1.1. above we can see that there is a direct equivalence between each integer, less than and relatively prime to m, and each integer, which is a power of a primitive root.

The particular power of the primitive root relative to the relative prime, is given a special name defined below.

Definition 11.3.1.4.

If $(a,m) = 1$, and g is a primitive root mod (m), then the least non-negative integer i, such that

$$a \equiv g^i \bmod (m)$$

is called the index of a (relative to the primitive root g) and is written

$$\mathrm{ind}_g(a).$$

But from theorem 11.3.1.8. we know that there are $\phi(\phi(m))$ primitive roots, so which do we use to determine ind(a), since ind(a) will be different for each primitive root? Some text books stipulate that the smallest primitive root is used and others write $\mathrm{ind}_g(a)$, to indicate which primitive root is concerned. We will use the latter convention.

Theorem 11.3.1.10.

Let m be a positive integer with a primitive root g, then,

$$\mathrm{ind}_g(a) = \mathrm{ind}_g(b)$$

if and only if,

$$a \equiv b \bmod (m).$$

Therefore relative to a given primitive root, the index of an integer, which is a member of a reduced residue system, is unique.

There is a connection between index and order, as we now show.

Theorem 11.3.1.11.

If $\mathrm{ord}_m a = S$, then

$$\mathrm{Ord}_m a^n = \frac{S}{(n,S)}$$

where $(n,S) = \mathrm{H.C.F.}\ (n,S)$.

If g is a primitive root mod (m), then

$$\mathrm{Ord}_m g = \phi(m)$$

therefore from Theorem 11.3.1.11.

$$\mathrm{Ord}_m g^n = \frac{\phi(m)}{(n,\phi(m))} \qquad 11.3.1.1.$$

but if $\text{ind}_g(a) = n$, then 11.3.1.2.

$$a \equiv g^n \bmod (m)$$

and from Theorem 11.3.1.5.

$$\text{Ord}_m(a) = \text{Ord}_m g^n \qquad 11.3.1.3.$$

From equations 11.3.1.1,2,3 above, then

$$\text{Ord}_m a = \frac{\phi(m)}{(\text{ind}_g(a),\ \phi(m))} \qquad 11.3.1.4.$$

Since for all $a < m$, $\text{ind}_g(a)$ can take any value between 1 and $\phi(m)$, then $(\text{ind}_g(a), m)$ will assume all values that are divisors of m.

We have proved the following theorem.

<u>Theorem 11.3.1.12</u>.

If m has a primitive root there exists a 'cyclic' set for every order S which divides $\phi(m)$.

Note, from equation 11.3.1.4., that a is a primitive root if and only if $\text{ind}_g(a)$ and $\phi(m)$ are relatively prime, hence from Theorem 11.3.1.8., there are $\phi(\phi(m))$ primitive roots of m.

11.3.2. The 'Cyclic' Array.

We are interested in this section in assessing the results of section 11.2 in the light of the theory just presented in section 11.3.1. That is, is it possible to find 'cyclic' sets of relative primes mod (m), which generate arrays mod (m), free from A.R.E.?

A) m = EVEN.

It was shown in section 11.2 that when m is even an array is limited to two rows in order to be free from A.R.E.

Theorems 11.2.2.6. and 11.2.2.7. showed that one solution was to use two relative primes a_j, a_i such that $(a_j + a_i) = m$. And Theorem 11.2.2.2. stated that,

$$((a_j - a_i), m) = 2$$

was another condition.

To reconcile the two conditions let us assume $a_j = (m-1)$ and $a_i = 1$, then certainly $(a_j + a_i) = m$, but does $(a_j - a_i) = m-2$ have

$$((m-2), m) = 2 \text{ ?}$$

The answer is yes because all even integers m_e can be formed from the equation,

$$m_e = 2 \cdot x \quad , \qquad x = 1,2,3,\ldots..$$

Since x alternates from odd to even, two adjacent even numbers m_{e_1} and m_{e_2} (whose difference is 2) will have x odd and x even. Let x_1 = odd and x_2 = even, then since an odd and an even integer are relatively prime when adjacent (a difference of 1) we have,

$$(m_{e_1}, m_{e_2}) = (2x_1, 2x_2) = 2 \ ,$$

since $\quad (x_1, x_2) = (x_2 - 1, x_2) = 1$

and therefore,

$$(m_{e_1}, m_{e_2}) = (m_{e_2} - 2, m_{e_2}) = 2 \ .$$

It is only necessary now to show that (m-1) and 1 can be arranged in 'cyclic' form. To do this we must show that one of them has order 2 mod (m). The integer 1 certainly hasn't but,

$$\begin{aligned}(m-1)^2 &= m^2 - 2m + 1 \\ &= m(m-1) + 1 \\ &\equiv 1 \bmod (m).\end{aligned}$$

Therefore, $\text{Ord}_m(m-1) = 2$.

Consider the following example.

Example 11.3.2.1.

Let m = 10, then we use $a_1 = 9$ and $a_2 = 1$ and the array is

$$\begin{array}{ccccccccc} 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

Now, $(9-1) = 8$ and as expected

$(8,10) = 2$, also

$9^2 \equiv 1 \bmod (10)$.

Note that all integers left of the duplicated centre column (and right of it) are distinct.

We have proved the following theorem.

Theorem 11.3.2.1.

If m is an even positive integer, there exists a cyclic set of order 2 mod (m), generated by (m-1), which generates an array mod (m), free from A.R.E.

This completes the case when m is even.

B) m = ODD.

In this section we will show that we can use 'cyclic' sets to generate arrays of the form below;

$$(A) = \begin{pmatrix} a & 2a & 3a & \ldots\ldots\ldots & (m-1)a \\ a^2 & 2a^2 & 3a^2 & \ldots\ldots\ldots & (m-1)a^2 \\ a^3 & 2a^3 & 3a^3 & \ldots\ldots\ldots & (m-1)a^2 \\ \vdots & \vdots & \vdots & & \\ a^S & 2a^S & 3a^S & \ldots\ldots\ldots & (m-1)a^S \end{pmatrix} \bmod (m) \quad 11.3.2.1.$$

where, $(a,m) = 1$

$\mathrm{Ord}_m a = S$ and $S \mid \phi(m)$.

But this implies and requires that the positive difference of any two elements from the 'cyclic' set, $(a^i - a^j)$, is relatively prime to m. From Theorem 11.2.2.4. this can only be guaranteed if the 'cyclic' set is incongruent mod (any divisor of m). We know it is incongruent mod (m) from Theorem 11.3.1.6., but to guarantee it is incongruent mod (any divisor of m), it must exist as a 'cyclic' set of order S mod (any divisor of m).

In (A) we must also constrain S to be less than p_1 the smallest prime divisor of m, due to the upper bound of Theorem 11.2.2.4.

Definition 11.3.2.1.

A 'cyclic' array is an array generated by a 'cyclic' set of order S mod (m), and is free from A.R.E., if and only if, the following conditions hold.

i) $S \leqslant (p_1 - 1)$, where p_1 is the smallest prime divisor of m.

ii) $\mathrm{Ord}_x a = S$, where $x > 1$, is any divisor of m, and a is the generator of the 'cyclic' set.

This definition is binding for all odd integers m and defines a 'cyclic' array used in the convolutional codes presented. Conditions i and ii of the above definition must be shown to hold for the different forms of m, that is;

a) $m = p$, an odd prime,

b) $m = p^{\alpha}$, p an odd prime and α an arbitrary positive integer.

c) $m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots\ldots p_r^{\alpha_r}$, p_i $(i = 1,2,\ldots\ldots,r)$ odd primes and α_i $(i = 1,2,\ldots,r)$ arbitrary positive integers.

B.1) $m = p$, an odd prime.

The smallest prime divisor of m is then, p itself, so that any 'cyclic' set of order $S \leq (p-1)$, will satisfy the first condition of definition 11.3.2.1.

From Theorem 11.3.1.9., m has primitive roots, so that there always exists an element, $a_i < m$, such that

$$a_i^{\phi(m)} \equiv 1 \bmod (m)$$

and $\quad \mathrm{Ord}_m a_i = \phi(m).$

Also from Theorem 11.3.1.12., we know that 'cyclic' sets exist, of all orders S, which divide $\phi(m)$. Therefore, if m is an odd prime, any 'cyclic' set satisfies condition (i) of definition 11.3.2.1.

Since p is divisible only by itself and unity, there are no other divisors of m, thus if a_j generates a 'cyclic' set, with $\mathrm{Ord}_m\, a_j = S$, then this satisfies condition (ii) of definition 11.3.2.1., since m is the only divisor of m, greater than one.

The following example illustrates how to construct 'cyclic' arrays for the case, $m = 11$.

<u>Example 11.3.2.1.</u>

Let $m = 11$, an odd prime, then $\phi(m) = 10$, and the set of relative primes, are all non-zero integers less than 11, giving,

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

The integer 2 is a primitive root of 11 and gives the following 'cyclic' set.

$$\begin{array}{l} \text{'cyclic' set} \\ \text{index.} \end{array} \left\{ \begin{array}{cccccccccc} 2, & 4, & 8, & 5, & 10, & 9, & 7, & 3, & 6, & 1 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10 \end{array} \right\}$$

From equation 11.3.14.

$$\text{Ord}_{11}\ a = \frac{10}{(\text{ind}_2 a,\ 10)} \qquad \text{giving,}$$

order	integer
2	10,
5	4, 5, 9, 3,
10	2, 8, 7, 6,
1	1

To generate an array we only require the 'cyclic' set generated by one element of a given order. Using 2 as an element of order 10 gives the following array.

		$1 \cdot a \equiv$	$2 \cdot a \equiv$	$3 \cdot a \equiv$	$4 \cdot a \equiv$	$5 \cdot a \equiv$	$6 \cdot a \equiv$	$7 \cdot a \equiv$	$8 \cdot a \equiv$	$9 \cdot a \equiv$	$10 \cdot a \equiv$	
a	$\equiv$	2	4	6	8	10	1	3	5	7	9	
a^2	$\equiv$	4	8	1	5	9	2	6	10	3	7	
a^3	$\equiv$	8	5	2	10	7	4	1	9	6	3	
a^4	$\equiv$	5	10	4	9	3	8	2	7	1	6	
a^5	$\equiv$	10	9	8	7	6	5	4	3	2	1	
a^6	$\equiv$	9	7	5	3	1	10	8	6	4	2	mod (m)
a^7	$\equiv$	7	3	10	6	2	9	5	1	8	4	
a^8	$\equiv$	3	6	9	1	4	7	10	2	5	8	
a^9	$\equiv$	6	1	7	2	8	3	9	4	10	5	
a^{10}	$\equiv$	1	2	3	4	5	6	7	8	9	10	

It can now be seen that the effect of ordering the rows of the array, according to the 'cyclic' set of a primitive root, is to produce an array whose columns are 'cyclic' shifts of the original 'cyclic' set, which is the left-most column.

Choosing 5 as the element of order 5, we obtain the 'cyclic' set.

$$\begin{array}{ll} \text{'cyclic' set} & \left\{\begin{array}{lllll} 5, & 3, & 4, & 9, & 1 \\ 5^1, & 5^2, & 5^3, & 5^4, & 5^5 \end{array}\right\} \\ \text{index} & \end{array}$$

Which gives the following 'cyclic' array.

$$\begin{array}{cccccccccc} 5 & 10 & 4 & 9 & 3 & 8 & 2 & 7 & 1 & 6 \\ 3 & 6 & 9 & 1 & 4 & 7 & 10 & 2 & 5 & 8 \\ 4 & 8 & 1 & 5 & 9 & 2 & 6 & 10 & 3 & 7 \\ 9 & 7 & 5 & 3 & 1 & 10 & 8 & 6 & 4 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

Using a 'cyclic' set of order $S < \phi(m)$ produces an array whose columns are 'cyclic' shifts of $(m-1)/S$ distinct columns. In this case the distinct columns are;

$$\{5, 3, 4, 9, 1\} \quad \& \quad \{10, 6, 8, 7, 2\}$$

From the element of order 2, we obtain the array;

$$\begin{array}{cccccccccc} 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10. \end{array}$$

And the columns are 'cyclic' shifts of

$$\frac{m-1}{S} = \frac{10}{2} = 5$$

distinct columns. The distinct columns being;

$$\{1, 10\}, \{2, 9\}, \{3, 8\}, \{4, 7\}, \{5, 6\}.$$

We have proved the following theorem.

<u>Theorem 11.3.2.2.</u>

If $m = p$, an odd prime, there exist 'cyclic' sets, for all orders, which divide $(p-1)$. The 'cyclic' arrays generated modulo (m) by all 'cyclic' sets, are free from Array Row Equivalence.

This concludes the results for m an odd prime.

B.2) $m = p^{\alpha}$, p an odd prime and α any positive integer < 1.

Since the smallest prime is p, we are interested in 'cyclic' sets of order $S \leqslant (p-1)$, that divide $\phi(m)$. From definition 11.2.1.3.;

$$\phi(m) = p^{\alpha-1}\,(p-1). \qquad 11.3.2.2.$$

Since m has primitive roots (from Theorem 11.3.1.9.) then 'cyclic' sets exist for all divisors of $\phi(m)$ from Theorem 11.3.1.12. Thus 'cyclic' sets of order $S = (p-1)$ exist and since if $S|(p-1)$, then $S|\phi(m)$, 'cyclic' sets exist of all orders that divide (p-1).

A general divisor of $\phi(m)$ can be expressed as, $d_{\phi(m)}$, where,

$$d_{\phi(m)} = S\,p^{x} \quad , \quad S|(p-1) \text{ and } S \leqslant (p-1), \quad x = 0,1,2,\ldots,(\alpha-1),$$

and 'cyclic' sets exist for all $d_{\phi(m)}$. But if $x > 0$, $d_{\phi(m)} > (p-1)$, therefore condition (i) of definition 11.3.2.1. is only satisfied by 'cyclic' sets whose order S divides (p-1).

To satisfy condition (ii) of definition 11.3.2.1. we require to know if a 'cyclic' set of order $S \leqslant (p-1)$, mod (p^{α}), exists as a 'cyclic' set of order S mod (any divisor of p^{α}). Any divisor of p^{α} must be of the form p^{c}, where $c \leqslant \alpha$. Consider the following theorem from Le Veque[4].

<u>Theorem 11.3.2.3.</u>

If p is an odd prime, and an integer a exists such that, $\mathrm{Ord}_{p}\, a = S$ and p^{z} divides $(a^{S}-1)$ but p^{z+1} does not divide $(a^{S}-1)$ then,

$$\mathrm{Ord}_{p^{n}}\, a = S\,p^{\max(0,n-z)}$$

where,

$$\begin{aligned} \max\,(0,n-z) &= 0 \quad \text{if } (n-z) \leqslant 0 \\ &= n-z \quad \text{if } (n-z) \geqslant 0 \end{aligned}$$

That is, if $a^S \equiv 1 \bmod (p)$, then

$$a^S \equiv 1 \bmod (p^n)$$

for all $n \leqslant z$. From the point of view of 'cyclic' arrays, if we find a 'cyclic' set of order $S \leqslant (p-1)$, mod (p^α), then from Theorem 11.3.2.3., if the 'cyclic' set is generated by a,

$$\text{Ord}_{p^\alpha}\, a = S' \cdot p^{\max(0,\alpha-z)} = S$$

therefore,

$$S'\ p^{\max(0,\alpha-z)} \leqslant (p-1)$$

which is only possible if $p^{\max(0,\alpha-z)} = 1$, therefore $S' = S$ and $\alpha \leqslant z$.

Since,

$$\text{Ord}_{p^\alpha}\, a = S \quad , \qquad \alpha \leqslant z$$

then

$$\text{Ord}_{p^{c}}\, a = S \quad , \qquad \text{for any } s < \alpha$$

and the 'cyclic' set exists as a 'cyclic' set of order S mod (any divisor of p^α). Therefore any 'cyclic' set of order $S \leqslant (p-1)$, satisfies the condition (ii) of definition 11.3.2.1.

Theorem 11.3.2.4.

If $m = p^\alpha$, p an odd prime and α any positive integer, any 'cyclic' set of order $S \leqslant (p-1)$ mod (m) generates a 'cyclic' array free from A.R.E.

The following example is given for $m = (7)^2$.

Example 11.3.2.3.

Let $m = 7^2 = 49$, then

$\phi(7^2) = 7 \cdot 6 = 42$, and the list of all possible divisors of $\phi(m)$ is, simply

{2, 3, 6, 7, 14, 21, 42}.

This gives the orders of possible 'cyclic' sets, but since we are constrained to orders, $S \lesssim (p-1) = 6$, we can only use {2, 3, 6} as orders. To find which integers $a < 49$ have the orders we require, we generate the complete set,

$$a,\ a^2 \bmod (m);\ a^3 \bmod (m),\ \ldots\ldots,\ a^{\phi(m)} \bmod (m)$$

where a is a primitive root. For $m = 7^2$, 3 is a primitive root.

'cyclic' set	3,	9,	27,	32,	47,	43,	31,	44,	34,	4,	12,	36,	10,	30,
indices of 3	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	41,	25,	26,	29,	38,	16,	48,	46,	40,	22,	17,	2,	6,	18,
	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	5,	15,	45,	37,	13,	39,	19,	8,	24,	23,	20,	11,	31,	1
	29	30	31	32	33	34	35	36	37	38	39	40	41	42

Putting equation 11.3.1.4. in the form required we obtain

$$\mathrm{Ord}_{49}\ a = \frac{42}{(\mathrm{ind}_3\ a,\ 42)}\ .$$

Using this equation, each integer a in the 'cyclic' set above along with its index, gives its order S mod (7^2).

e.g. if $a = 48$, $\mathrm{ind}_3\ 48 = 21$

$$\mathrm{Ord}_{49}\ 48 = \frac{42}{(21,\ 42)} = \frac{42}{21} = 2,$$

and $48^2 \equiv 1 \bmod (49)$.

For the complete 'cyclic' set we obtain the following sets of orders and associated elements of that order.

Order S	Integers of order S
2	48
3	30, 18
6	31, 19
7	43, 36, 29, 22, 15, 8
14	27, 34, 41, 6, 13, 20
21	9, 32, 44, 4, 25, 16, 46, 2, 37, 39, 23, 11
42	24, 33, 3, 47, 12, 10, 26, 38, 40, 17, 5, 45.

There are two 'cyclic' sets of order 6 possible, generated by 31 and 19. However we only require one 'cyclic' set and we use that generated by 19.

Let $a = 19$, then $a^6 \equiv 1 \bmod (7^2)$

$$\begin{array}{cccccc} a & a^2 & a^3 & a^4 & a^5 & a^6 \\ \{19 & 18 & 48 & 30 & 31 & 1\} \end{array} \bmod (7^2)$$

To verify Theorem 11.3.2.3., we reduce the 'cyclic' set mod (7), since 7 is the only divisor of 7^2. This gives,

$$\{5, 4, 6, 2, 3, 1\} \bmod (7)$$

therefore $a^6 \equiv 1 \bmod (7)$. We also find that $7^3 | (a^6-1)$ but $7^4 \nmid (a^6-1)$, meaning 7^4 does not divide (a^6-1), therefore from Theorem 11.3.2.3., $z = 3$. Thus $a = 19$ will generate a 'cyclic' set of order 6 mod (7^3), that is $m = 343$, and since $S \leq (p-1)$ and the set is incongruent mod (7^2) and mod (7), it will generate an array free from A.R.E. mod (343).

For $m = 49$, $a = 19$ generates the 'cyclic' array below, of which the first 19 columns are given.

19	38	8	27	46	16	35	5	24	43	13	32	2	21
18	36	5	23	41	10	28	46	15	33	2	20	38	7
48	47	46	45	44	43	42	41	40	39	38	37	36	35
30	11	41	22	3	33	14	44	25	6	36	17	47	28
31	13	44	26	8	39	21	3	34	16	47	29	11	42
1	2	3	4	5	6	7	8	9	10	11	12	13	14

40	10	29	48	18	. . . .	30
25	43	12	30	48	. . . .	31
34	33	32	31	30	. . . .	1
9	39	20	1	31	. . . .	19
24	6	37	19	1	. . . .	18
15	16	17	18	19	. . . .	48

When the order of the 'cyclic' set, S, is less than $\phi(m)$, we obtain codes with rate $R > 1/2$. Since there are 48 distinct integers in the array, $k_o = 48$, and since the order $S = 6$, $n_o - k_o = 6$,

thus $n_o = 6 + 48 = 54$ and the rate is, $R = 48/54 = 0.88888$.

There are 6 orthogonal estimates of each of the 48 message-digits therefore the error-correcting capability, $t = S/2 = 3$.

There are two 'cyclic' sets of order 3 possible, generated by 30 and 18. Only one can be used and we will use that generated by 18.

Let $a = 18$, then $a^3 \equiv 1 \bmod (7^2)$

$$\begin{array}{ccc} a & a^2 & a^3 \\ \{\,18 & 30 & 1\,\} \end{array} \bmod (7^2)$$

also

$$\{\,4 \quad 2 \quad 1\,\} \bmod (7)$$

Again $7^3 | (a^3-1)$ but $7^4 \nmid (a^3-1)$ so that from Theorem 11.3.2.3., $z = 3$ and therefore

$$a^3 \equiv 1 \bmod (7^3)$$

and $a = 18$ generates a 'cyclic' set of order $3 < (p-1)$, mod (7^3)

which is incongruent mod (7^2) and mod (7) and therefore generates a 'cyclic' array free from A.R.E.

For m = 49, a = 18 generates the 'cyclic' array below, of which the first 19 columns are given.

18	36	5	23	41	10	28	46	15	33	2	20	38
30	11	41	22	3	33	14	44	25	6	36	17	47
1	2	3	4	5	6	7	8	9	10	11	12	13

7	25	43	12	30	48	. . . .	31
28	9	39	20	1	31	. . . .	19
14	15	16	17	18	19	. . . .	48 .

This gives a code with,

$k_o = 48, \quad n_o - k_o = 3, \quad n_o = 51, \quad d_m = 4$

Rate = 48/51 = 0.941.

Finally, there is only one 'cyclic' set of order 2 possible generated by 48.

Let a = 48, then $a^2 \equiv 1 \bmod (7^2)$

$$\begin{array}{cc} a & a^2 \\ \{ 48 & 1 \} \end{array} \bmod (7^2)$$

also

$$\{ \; 6 \quad 1 \; \} \bmod (7)$$

But $7^2 | (a^2-1)$ but $7^3 \nmid (a^2-1)$ so that from Theorem 11.3.2.3., z = 2, so that for $m = 7^3$,

$$\begin{aligned} \text{Ord}_{7^3}\, 48 &= 2 \cdot p^{\max(0,3-2)} \quad , \quad p = 7 \\ &= 2 \cdot 7 \end{aligned}$$

and 48 has order 14 mod (7^3). Since 14 > (p-1), this is of no use, as A.R.E. will occur.

For m = 49, a = 48 generates the 'cyclic' array below.

```
48  47  46  45 . . . . . . . . .  3   2   1
 1   2   3   4 . . . . . . . . . 46  47  48
```

This gives a code with,

$k_o = 48, \quad n_o - k_o = 2, \quad n_o = 50, \quad d_m = 3$

Rate = 48/50 = 0.96.

This concludes example 11.3.2.3. and the section on $m = p^{\alpha}$.

B.3) $m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots\ldots p_r^{\alpha_r}$

In this case we specify, $\alpha_i \geq 0$ for all i, but $\alpha_i \neq 0$ for all i. Also we assume that p_1 is the smallest prime factor in m, and

$$p_1 < p_2 < \ldots\ldots\ldots < p_r$$

From Theorem 11.3.1.9. it can be seen that integers of this form do not have primitive roots so that Theorem 11.3.1.12. does not hold and equation 11.3.1.4. cannot be used to find the orders of different integers since index is relative to some primitive root. We are not sure if 'cyclic' sets exist, however we can deduce from Theorem 11.3.1.1. that any integer a, such that (a,m) = 1, must generate a 'cyclic' set, since,

$$a^{\phi(m)} \equiv 1 \bmod (m)$$

and $\phi(m)$ cannot be the smallest integer which satisfies this congruence, or else a would be a primitive root which is not possible. Thus some $S < \phi(m)$ with $S | \phi(m)$ must exist, such that,

$$a^S \equiv 1 \bmod (m).$$

From Definition 11.2.1.3.,

$$\phi(m) = p_1^{\alpha_1 - 1} \cdot p_2^{\alpha_2 - 1} \ldots\ldots p_r^{\alpha_r - 1} \cdot (p_1 - 1) \cdot (p_2 - 1) \ldots . (p_r - 1)$$

so that, devisors of $\phi(m)$ which are $\leqslant (p_1-1)$, are possible, as orders for 'cyclic' sets.

The following theorem, known as the Chinese remainder theorem, will be found useful and is given with its proof, as the proof is useful also.

Throughout the following, unless stated otherwise, we assume $(x,m) = 1$.

Theorem 11.3.2.5.

If the positive integers m_i, $i = 1,2,\ldots,r$ are relatively prime in pairs and if a_i, $i = 1,2,\ldots,r$ are any given integers, then the r congruences

$$x \equiv a_i \bmod (m_i) \qquad i = 1,2,\ldots,r,$$

have a common solution which is unique modulo $(m_1.m_2 \ldots\ldots m_r)$.

Proof.

Let $m = m_1\, m_2 \ldots\ldots m_r$, and write $m = m_i\, M_i$, then $(m_i,M_i) = 1$, for each i, and from Euler's Theorem,

$$M_i^{\phi(m_i)} \equiv 1 \bmod (m_i)$$

but M_j includes m_i as a factor so that

$$M_j \equiv 0 \bmod (m_i).$$

If we write,

$$x = \sum_{i=1}^{r} a_i\, M_i^{\phi(m_i)}$$

$$x \equiv \sum_{i=1}^{r} a_i\, M_i^{\phi(m_i)} \bmod (m_j)$$

but $$M_i^{\phi(m_i)} \equiv 0 \bmod (m_j) \quad \text{for } i \neq j$$

therefore,

$$x \equiv a_j M_j^{\phi(m_j)} \bmod (m_j)$$

$$\equiv a_j \bmod (m_j) \quad , \quad j = 1,2,\ldots,r.$$

(since $M_j^{\phi(m_j)} \equiv 1 \bmod (m_j)$),

so that x is a common solution to all congruences. If there is another solution, say z, then,

$$z \equiv x \equiv a_j \bmod (m_j) \quad \text{for all } j$$

$$z = b_1 m_1 + x$$

$$z = b_2 m_2 + x$$

$$\vdots$$

$$z = b_r m_r + x$$

and from Theorem 11.2.1.2.,

$$z \equiv x \bmod (\text{L.C.M. } \{m_1, m_2, \ldots, m_r\})$$

but since $(m_1, M_1) = 1$, then

$$\text{H.C.F. } (m_1, m_2, m_3, \ldots, m_r) = 1$$

thus $\text{L.C.M. } (m_1, m_2, \ldots, m_r) = m_1 . m_2 . \ldots\ldots . m_r.$

and $z \equiv x \bmod (m)$

and x is unique mod (m).

Q.E.D.

Since $m = p_1^{\alpha_1} . p_2^{\alpha_2} \ldots . p_r^{\alpha_r}$ if we write

$$m = p_i P_i \quad , \quad \text{then} \quad (p_i, P_i) = p_i$$

and a unique solution to the congruences

$$x \equiv a_i \bmod (p_i) \qquad i = 1,2,\ldots,r$$

cannot be guaranteed. However, if we write

$$m = p_i^{\alpha_i} \cdot P_i \quad , \quad \text{then} \quad (p_i^{\alpha_i}, P_i) = 1$$

and we can find a unique common solution to the congruences,

$$x \equiv a_i \bmod (p_i^{\alpha_i}) \qquad i = 1.2,\ldots,r$$

that is, we have

$$\left.\begin{array}{l} x = b_1\, p_i^{\alpha_1} + a_1 \\ x = b_2\, p_2^{\alpha_2} + a_2 \\ \vdots \\ x = b_r\, p_r^{\alpha_r} + a_r \end{array}\right\} \qquad 11.3.2.3.$$

From Theorems 11.3.1.4. and 11.3.1.5., we also have,

$$\mathrm{Ord}_{p_i^{\alpha_i}}\, x = \mathrm{Ord}_{p_i^{\alpha_i}}\, a_i \qquad i = 1,2,\ldots,r. \qquad 11.3.2.4.$$

But what is the order of x mod (m)?

The following theorem shows that the only solution useful to the construction of arrays for codes, is that

$$\mathrm{Ord}_{p_i^{\alpha_i}}\, x = \mathrm{Ord}_{p_i^{\alpha_i}}\, a_i = S \qquad 11.3.2.5.$$

for all i = 1,2,...,r.

Theorem 11.3.2.6.

If x is a common solution to the congruences,

$$x \equiv a_i \bmod (p_i^{\alpha_i}) \qquad i = 1,2,\ldots,r$$

then if $\quad \mathrm{Ord}_{p_i^{\alpha_i}}\, a_i = S_i \qquad i = 1,2,\ldots,r \qquad$ 11.3.2.6.

and $\quad m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots\cdots p_r^{\alpha_r} \quad$, then

$$\mathrm{Ord}_m\, x = c \cdot d \quad , \qquad c \geqslant 1$$

where

$$d = \text{L.C.M.}\ (S_1, S_2, S_3, \ldots, S_r)$$

Proof.

Since

$$a_i^{S_i} \equiv 1 \bmod (p_i^{\alpha_i}), \qquad i = 1,2,\ldots,r$$

we have

$$x^{S_1} \equiv 1 \bmod (p_1^{\alpha_1})$$

$$x^{S_2} \equiv 1 \bmod (p_2^{\alpha_2})$$

$$\vdots$$

$$x^{S_r} \equiv 1 \bmod (p_r^{\alpha_r})$$

Let us assume $x^b \equiv 1 \bmod (m)$ then for any divisor $p_i^{\alpha_i}$ of m we have,

$$x^b \equiv 1 \bmod (p_1^{\alpha_1})$$

$$x^b \equiv 1 \bmod (p_2^{\alpha_2})$$

$$\vdots$$

$$x^b \equiv 1 \bmod (p_r^{\alpha_r})$$

So, from Theorem 11.3.1.3.,

$$S_i | b \quad \text{for all } i = 1,2,\ldots,r.$$

Since b must be divisible by all orders S_i, then

$$b \geqslant \text{L.C.M.}\ (S_1, S_2, \ldots, S_r)$$

$$\geqslant d.$$

$$= c \cdot d \ , \quad c \geqslant 1.$$

Q.E.D.

Thus if x has order S and the a_i have different orders, then the 'cyclic' set generated by x will not have the same order mod (any divisor of m) and A.R.E. will occur. We therefore require the orders of the a_i to be the same, as specified by equation 11.3.2.5. Equation 11.3.2.6. requires,

$$S_i | \phi(p_i^{\alpha_i}) \qquad i = 1,2,\ldots.,r$$

but equation 11.3.2.5., requires,

$$S | \phi(p_i^{\alpha_i})$$

for all i = 1,2,...,r.

So that one requirement is that the $\phi(p_i^{\alpha_i})$ must have common divisors. Since, for each $p_i^{\alpha_i}$, 'cyclic' sets exist for all orders that divide $\phi(p_i^{\alpha_i})$, if the $\phi(p_i^{\alpha_i})$ have common divisors then there will exist a set of a_i with order equal to the common divisor.

Of course the common divisors can only be used if they are less than or equal to (p_1-1); that is, from definition 11.3.2.1. we must constrain $S \leqslant (p_1-1)$.

If we assume a set of a_i's have been obtained with same order S for all $p_i^{\alpha_i}$ then in Theorem 11.3.2.6., we have,

$$S_1 = S_2 = \ldots\ldots = S_r = S$$

and

$$b \geqslant S.$$

The following theorem shows that in fact

$$b = S.$$

Theorem 11.3.2.7.

If x is a common solution to the congruences,

$$x \equiv a_i \bmod (p_i^{i}), \qquad i = 1,2,\ldots,r$$

then if, $$\text{Ord}_{p_i^{\alpha_i}} a_i = S, \qquad i = 1,2,\ldots,r$$

and $$m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot\cdot\cdot\cdot\cdot p_r^{\alpha_r}$$, then

$$\text{Ord}_m x = S.$$

<u>Proof.</u>

Since,

$$x^S \equiv 1 \bmod (p_i^{\alpha_i}) , \qquad 11.3.2.7.$$

for all $i = 1,2,\ldots,r$, then

$$p_i^{\alpha_i} \mid (x^S-1) \text{ for all } i.$$

But, $(p_i^{\alpha_i}, p_j^{\alpha_j}) = 1$ for $i \neq j$

so that if $m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot\cdot\cdot\cdot\cdot p_r^{\alpha_r}$

$$m \mid (x^S-1) \text{ and}$$

$$x^S \equiv 1 \bmod (m).$$

If $c < S$ exists such that

$$x^c \equiv 1 \bmod (m)$$

then since $p_i^{\alpha_i}$ divides m for all i

$$x^c \equiv 1 \bmod (p_i^{\alpha_i})$$

which is not possible since S is the smallest integer for which equation 11.3.2.7. holds.

Therefore, $\text{Ord}_m x = S.$

Q.E.D.

So far the procedure is as follows;

i) find a set of integers a_i, such that

$$\text{Ord}_{p_i^{\alpha_i}}\, a_i = S \quad \text{for all } i$$

where $\qquad S \mid \phi(p_i^{\alpha_i}) \qquad$ for all i,

and $\qquad S \leqslant (p_1-1)$.

ii) Use the Chinese remainder theorem to obtain a unique solution x, to the congruences,

$$x \equiv a_i \bmod (p_i^{\alpha_i})$$

for all i. Then $\text{Ord}_m\, x = S$ and x generates a 'cyclic' set of order S mod (m) and mod (any divisor of the form $p_i^{\alpha_i}$), for all i.

However, to fully satisfy the second condition of definition 11.3.2.1., the 'cyclic' set generated by x must have order S mod (any divisor of m).

Since,

$$\text{Ord}_{p_i^{\alpha_i}}\, x = \text{Ord}_{p_i^{\alpha_i}}\, a_i, \qquad \text{for all } i$$

then $\qquad \text{Ord}_{p_i^{\alpha_i}}\, x = S, \qquad$ for all i,

therefore, from Theorem 11.3.2.3.,

$$\text{Ord}_{p_i^{s}}\, x = S$$

for any $s \leqslant \alpha_i$, and all i.

In particular, then

$$\text{Ord}_{p_i}\, x = S, \qquad \text{for all } i. \qquad 11.3.2.8.$$

Consequently, if there existed some integer $c \leqslant S$, such that,

$$x^c \equiv 1 \bmod (\text{some divisor of } m)$$

then this implies that

$$x^c \equiv 1 \bmod (\text{some divisor } p_i)$$

since some prime p_i must always be a divisor of m. But this contradicts equation 11.3.2.8., therefore c = S, and x will generate a 'cyclic' set of order S mod (any divisor of m).

These results are put into the theorem form for brevity as follows.

Theorem 11.3.2.8.

If $m = p_1^{\alpha_1}\cdot p_2^{\alpha_2}\cdot\ldots\cdot p_r^{\alpha_r}$, where each p is an odd prime, and $p_1 < p_2 < \ldots < p_r$. Then if a set of integers a_i, $i = 1,2,\ldots,r$, exist such that

$$\operatorname{Ord}_{p_i^{\alpha_i}} a_i = S \quad \text{for all } i$$

where $S \mid \phi(p_i^{\alpha_i})$ for all i and

$$S \leqslant (p_1-1)$$

then an integer, x, can be found as a unique solution to the congruences,

$$x \equiv a_i \bmod (p_i^{\alpha_i})$$

and x will generate a 'cyclic' set of order S mod (any divisor of m).

Two examples follow, one with

$$m = p_1\cdot p_2\cdot p_3 = 5\cdot 13\cdot 37,$$

and one with

$$m = p_1^2\cdot p_2^2 = 5^2\cdot 13^2.$$

Example 11.3.2.4.

Let $m = 5\cdot 13\cdot 37 = 2405$, then

$$\phi(5) = 4 = 2\cdot 2$$

$$\phi(13) = 12 = 2\cdot 2\cdot 3$$

$$\phi(37) = 36 = 2\cdot 2\cdot 3\cdot 3$$

The set of common divisors is limited to two values, $S = 2$ and $S = 4$ and $S \leq (p_1-1)$ for both.

(a) Let, $S = 4$ then by procedures described in section B.1. we find elements of order 4 mod (each p_i), since $\alpha_i = 1$, for all i. We obtain,

$$2^4 \equiv 1 \bmod (5)$$

$$5^4 \equiv 1 \bmod (13)$$

$$31^4 \equiv 1 \bmod (37)$$

and therefore, we require a solution to the congruences,

$$x \equiv 2 \bmod (5)$$

$$x \equiv 5 \bmod (13)$$

$$x \equiv 31 \bmod (37)$$

One can use the solution indicated in the proof of Theorem 11.3.2.5., that is,

$$x = \sum_{r=1}^{3} a_i M^{\phi(p_i)} \qquad 11.3.2.9.$$

$$= 2(13\cdot 37)^4 + 5(5\cdot 37)^{12} + 31(5\cdot 13)^{36}$$

of course this is reduced mod $(5\cdot 13\cdot 37)$ to find x, but it still involves a lot of work. The author found a solution by trial and error, using a short-cut, in the following way. We saw in part (A) that

$$(m-1)^2 \equiv 1 \bmod (m)$$

for any m. Therefore if x has order 4, then,

$$x^2 \equiv (m-1) \bmod (m).$$

This produced the solution,

$$x \equiv 512 \bmod (2405)$$

and it is seen that,

$$512 \equiv 2 \bmod (5)$$

$$512 \equiv 5 \bmod (13)$$

$$512 \equiv 31 \bmod (37)$$

Generating the 'cyclic' set of 512,

let $a = 512, \quad (512)^4 \equiv 1 \bmod (5\cdot13\cdot37)$

a	a^2	a^3	a^4	
{512	2404	1893	1}	mod (2405).

To check that this exists as a 'cyclic' set of order 4 mod (any divisor of m), we can reduce the set as below.

{ 2	4	3	1 }	mod (5)
{ 5	12	8	1 }	mod (13)
{ 31	36	6	1 }	mod (37)
{ 57	64	8	1 }	mod (5·13)
{142	184	43	1 }	mod (5·37)
{ 31	480	450	1 }	mod (13·37)

Thus the array generated mod (2405), by the 'cyclic' set generated by $x \equiv 512 \bmod (2405)$, will specify a code with,

$$k_o = 2404, \quad n_o - k_o = 4, \quad n_o = 2408, \quad d_m = 5$$

$$\text{Rate} = 2404/2408 = 0{\cdot}9983.$$

(b) Let S = 2, then we find integers of order 2 mod (each p_i), which results in the following,

$$4^2 \equiv 1 \bmod (5)$$

$$12^2 \equiv 1 \bmod (13)$$

$$36^2 \equiv 1 \bmod (37)$$

and therefore we require a solution to the congruences,

$$x \equiv 4 \bmod (5)$$

$$x \equiv 12 \bmod (13)$$

$$x \equiv 36 \bmod (37).$$

Using equation 11.3.2.9. above,

$$x = 4(13\cdot37)^{4} + 12(5\cdot37)^{12} + 36(5\cdot13)^{36}$$

But of course, we know,

$$(m-1)^{2} \equiv 1 \bmod (m)$$

or $$(2404)^{2} \equiv 1 \bmod (2405)$$

and in fact

$$2404 \equiv 4 \bmod (5)$$

$$2404 \equiv 12 \bmod (13)$$

$$2404 \equiv 36 \bmod (37)$$

therefore $x \equiv 2404 \bmod (2405)$ and we have the 'cyclic' set below.

Let $a = 2404$, $a^{2} \equiv 1 \bmod (m)$

$$\begin{array}{cc} a & a^{2} \\ \{2404 & 1\} \end{array} \bmod (2405).$$

Modulo (any divisor of m), we have,

$$\{\ 4 \quad 1\ \} \bmod (5)$$

$$\{\ 12 \quad 1\ \} \bmod (13)$$

$$\{\ 36 \quad 1\ \} \bmod (37)$$

$$\{\ 64 \quad 1\ \} \bmod (5\cdot13)$$

$$\{184 \quad 1\ \} \bmod (5\cdot37)$$

$$\{480 \quad 1\ \} \bmod (13\cdot37)$$

Thus the array generated mod (2405) by the 'cyclic' set generated by $x \equiv 2404 \bmod (2405)$, will specify a code,

$k_o = 2404, \quad n_o - k_o = 2, \quad n_o = 2406, \quad d_m = 3$

Rate = 2404/2406 = 0·99916.

In the following example a slightly different approach is used, which may be quicker.

Example 11.3.2.5.

Let $m = 5^2 \cdot 13^2 = 4225$,

then,

$$\phi(5^2) = 20 = 2\cdot2\cdot5$$

$$\phi(13^2) = 156 = 2\cdot2\cdot3\cdot13$$

Common divisors = {2, 4}.

(a) Let S = 4, then we find, using trail and error,

$$(268)^4 \equiv 1 \bmod (4225)$$

but we must check that the residues of 268 mod (any divisor of 4225) generate 'cyclic' sets of order 4.

In particular,

$268 \equiv 18 \bmod (5^2)$

$268 \equiv 99 \bmod (13^2)$ and we obtain the 'cyclic' sets

$\{18, \quad 24, \quad 7, \quad 1\} \bmod (5^2)$

$\{99, \quad 168, \quad 70, \quad 1\} \bmod (13^2)$

Since 18 & 99 have order 4, we know that 268 has order 4 also. Checking all divisors we have

$268 \equiv 3 \bmod (5)$ and

$\{3, 4, 2, 1\} \bmod (5)$, 3 has order 4 mod (5)

$268 \equiv 8 \bmod (13)$ and

$\{8, 12, 5, 1\} \bmod (13)$

and 8 has order 4 mod (13)

$268 \equiv 8 \mod (5\cdot 13)$ and

$\{8, 64, 57, 1\} \mod (5\cdot 13)$

and 8 has order 4 mod $(5\cdot 13)$.

$268 \equiv 268 \mod (5^2\cdot 13)$ and

$\{268, 324, 57, 1\} \mod (5^2\cdot 13)$

and 268 has order 4 mod $(5^2\cdot 13)$.

$268 \equiv 268 \mod (5\cdot 13^2)$ and

$\{268, 844, 577, 1\} \mod (5\cdot 13^2$

and 268 has order 4 mod $(5\cdot 13^2)$

Therefore the 'cyclic' set,

$\{268, 4224, 3957, 1\} \mod (4225)$

exists as a 'cyclic' set of order 4, modulo any divisor of 4225.

(b) Let S = 2. then we find,

$(4224)^2 \equiv 1 \mod (4225)$

again we must check that the residues of 4224 mod (any divisor of 4225) generate 'cyclic' sets of order 4.

In particular,

$4224 \equiv 24 \mod (5^2)$ and

$\{24, 1\} \mod (5^2)$

and 24 has order 2 mod (5^2).

$4224 \equiv 168 \mod (13^2)$ and

$\{168, 1\} \mod (13^2)$

and 168 has order 2 mod (13^2). Therefore we know that 4224 has order 2 mod (4225).

Checking all divisors,

$4224 \equiv 4 \mod (5)$

$\{4, 1\}$

and 4 has order 2.

$$4224 \equiv 12 \mod (13)$$

$$\{12, 1\}$$

and 12 has order 2.

$$4224 \equiv 64 \mod (5 \cdot 13)$$

$$\{64, 1\}$$

and 64 has order 2.

$$4224 \equiv 324 \mod (5^2 \cdot 13)$$

$$\{324, 1\}$$

and 324 has order 2.

$$4224 \equiv 844 \mod (13^2 \cdot 5)$$

$$\{844, 1\}$$

and 844 has order 2.

Therefore the cyclic set,

$$\{4224, 1\} \mod (4225)$$

exists as a 'cyclic' set of order 2, modulo any divisor of 4225.

There are no other common divisors $\leq (p_1-1)$ and the example is concluded.

Before leaving this section, dealing with composite integers, it may be worth mentioning that it is possible to choose composite integers so that a 'cyclic' set of maximum order exists.

Let $\quad p_i = b_i(p_1-1) + 1, \qquad i = 1,2,\ldots,r,$

and we choose the b_i so that p_i is an odd prime, with of course p_1 an odd prime also. When $b_i = 1$, $p_i = p_1$.

Then since,

$$\phi(p_i) = p_i-1 = b_i(p_1-1)$$

and

$$\phi(p_i^{\alpha_i}) = p_i^{\alpha_i-1}(p_i-1)$$

$$= p_i^{\alpha_i-1}\, b_i(p_1-1)$$

we have (p_1-1) always as a common divisor of the $\phi(p_i^{\alpha_i})$ for all i.

This way, we can guarantee a set of a_i's exist for order (p_1-1) mod $(p_i^{\alpha_i})$, from which to find x.

The following example demonstrates the idea.

Example 11.3.2.6.

Let $p_1 = 5$, then an associated set of primes which can be used to form composite integers, from which 'cyclic' sets of order $(p_1-1) = 4$ mod (the composite) are obtainable, is given by;

$$p_i = b_i(p_1-1) + 1, \qquad p_i \text{ an odd prime}$$

$b_1 = 1$	$p_1 = 1\,(5-1) + 1 = 5$
$b_2 = 3$	$p_2 = 3\,(5-1) + 1 = 13$
$b_3 = 4$	$p_3 = 4\,(5-1) + 1 = 17$
$b_4 = 7$	$p_4 = 7\,(5-1) + 1 = 29$
$b_5 = 9$	$p_5 = 9\,(5-1) + 1 = 37$
$b_6 = 10$	$p_6 = 10\,(5-1) + 1 = 41$
$b_7 = 13$	$p_7 = 13\,(5-1) + 1 = 53$
$b_8 = 15$	$p_8 = 15\,(5-1) + 1 = 61$
$b_9 = 18$	$p_9 = 18\,(5-1) + 1 = 73$

Any integer formed by a composite product of these primes or powers of these primes, with $p_1 = 5$, can be used to find 'cyclic' sets of order 4 mod (the composite), which exist as 'cyclic' sets mod (any divisor of the composite).

This idea can be extended in the following way. Suppose one wishes to construct a code with error-correcting capability 10. Then this can be accomplished with a code whose array is generated by a 'cyclic' set of order S = 20.

Let $m = p_1 = 41$, an odd prime.

Then from section B.1.

$$\phi(41) = 40$$

which has divisors, $\{2,4,5,10,20,40\}$.

Using a 'cyclic' set of order 20, gives a code with rate,

$$R = 40/60 = 2/3.$$

What if we require a rate greater than this?

Let $m = (41)^2$, then from section B.2.

$$\phi(41^2) = 41 \cdot 40$$

which has divisors, less than or equal to (p_1-1) of,

$$\{2,4,5,10,20,40\}.$$

Using a 'cyclic' set of order 20 gives a code of rate,

$$R = 1680/1700 = 84/85.$$

However we can slightly improve the rate without using $m = (41)^3$.

Let $m = 41 \cdot 61$,

then from example 11.3.26,

$$41 = 2(20) + 1$$

$$61 = 3(20) + 1$$

here we are implying $20 = p_1-1$, but this means $p_1 = 21$ which is not prime.

By choosing p_1 as a non-prime we can construct composities from which 'cyclic' sets can be obtained of orders less than the actual (p_1-1) used, since here $p_1 = 41$, but we cannot obtain a cyclic set of order 40 mod $(41 \cdot 61)$.

$$\phi(41) = 2 \cdot 2 \cdot 2 \cdot 5$$

$$\phi(61) = 2 \cdot 2 \cdot 3 \cdot 5$$

the set of common divisors is,

$$\{2,4,5,10,20\}$$

Using a set of order 20 mod $(41\cdot 61)$ gives a code of rate,

$$R = 2500/2520 = 125/126.$$

We can generalize our results in the following way.

If one wishes to form a code whose 'cyclic' array is generated by a 'cyclic' set of order S mod (m), then if,

(i) $m = p_i$

(ii) $m' = p_i^{\alpha}$

(iii) $m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots . p_r^{\alpha_r}$

the primes p_i used must be of the form,

$$p_i = b_i(S) + 1$$

$$b_i = 1,2,\ldots\ldots \quad , \quad S \leq (p_1 - 1).$$

Let $m = p$ and $S = 2t$, then any t - error-correcting code can be constructed from any p, where

$$p = b_i(2t) + 1 = \text{a prime}.$$

Since the code will have $k_o = m - 1 = b_i(2t)$ message-digits and $n_o - k_o = 2t$ check-digits the codes will have rate,

$$R = \frac{b_i\cdot(2t)}{b_i(2t) + (2t)} = \frac{b_i}{b_i + 1} \quad .$$

11.4 Non-cyclic Arrays.

The array generated by a 'cyclic' set, has in effect, each of its rows generated by an element of the 'cyclic' set. The distinct leftwise sequences of the array will not be disrupted if the rows of the array are permutated. This corresponds to permutating the elements of the 'cyclic' set. For example the array generated by the set {1,2,3,4} mod (5), in example 11.1.1. is a permutation on the 'cyclic' array generated by the 'cyclic' set {2,4,3,1} mod (5). One can also use sub-arrays generated by subsets of the elements in the cyclic set. For example we could use the set {1,2,3} with safety since we know the leftwise sequences are distinct.

The sub-arrays described (as distinct from sub-'cyclic' arrays, using sub-'cyclic' sets) and the permutated arrays are examples of non-'cyclic' arrays. They suffer from the disadvantage that they require k_o majority gates to decode. Otherwise we can devise non-'cyclic' arrays with an error-correcting capability not achiev able with 'cyclic' array codes. Thus non-'cyclic' arrays must be limited to codes with small k_o, to keep decoder complexity down.

Example 11.4.1.

Consider the array below, generated by the 'cyclic' set of 3 mod (7)

$$\begin{array}{cccccc} 3 & 6 & 2 & 5 & 1 & 4 \\ 2 & 4 & 6 & 1 & 3 & 5 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 4 & 1 & 5 & 2 & 6 & 3 \\ 5 & 4 & 1 & 6 & 4 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array} \quad \text{mod } (7)$$

We can obtain sub-'cyclic' arrays by using the sub-'cyclic' sets {2 4 1} and {6 1}. But we can also use the non-'cyclic' sets,

{1 2 3}, {1 2 3 4}, {1 2 3 4 5} mod (7)

which generate non-'cyclic' sub-arrays. The non-'cyclic' set {1 2 3 4} gives a code,

$$k_o = 6, \quad n_o - k_o = 4, \quad n_o = 10, \quad d_m = 5$$

$$\text{Rate} = 6/10 = 3/5$$

which is not attainable with a 'cyclic' array having $k_o = 6$ distinct integers.

We now obtain some non-'cyclic' results.

Theorem 11.4.1.

If an integer m, is expressed as a product of its primes,

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots\cdots p_r^{\alpha_r}$$

with $p_1 < p_2 < \cdots\cdots < p_r$

then the integers below, with $b \lesssim (p_1 - 1)$

$$(1,2,3,\ldots,b) = X$$

a) are relatively prime to m, and

b) are the largest set of consecutive integers relatively prime to m, when $b = (p_1 - 1)$.

Proof.

Since p_1 is the smallest prime factor, no $x < p_1$ is divisible by any p_i, prime factor of m and part (a) is trivial.

Part (b) is satisfied by proving that any set of p_i consecutive integers contains at least one integer divisible by p_i.

Let a general sequence of p_i integers be,

$$x,\ x+1,\ x+2, \ldots\ldots\ldots,\ x+p_i-1$$

i) if x is divisible by p_i, then it is trivial.

ii) if x is not divisible by p_i, then x can be expressed as,

$$x = kp_i + r, \qquad r < p_i$$

then since $r < p_i$, $x + p_i - r$ must be contained in the sequence and

$$x + p_i - r = kp_i + p_i = (k+1)p_i\ .$$

Since p_1 is the smallest prime factor in m, any sequence of p_1 or more consecutive integers contains at least one divisible by p_1.

Since X is a relatively prime set with $b \leqslant (p_1-1)$ elements it must be a maximum set, when $b = (p_1-1)$.

Q.E.D.

Note that this theorem is true for m even or odd, but when m is even $p_1-1 = 1$, thus it is effectively confined to m being odd.

The following theorem shows that the set defined in Theorem 11.4.1. generates an array free from A.R.E.

Theorem 11.4.2.

Any array generated modulo (m) by a set of consecutive relative primes, with $b \leqslant (p_1-1)$

$$\{1,2,3,\ldots\ldots,b\} = X$$

where p_1 is the smallest prime factor of m, is free from array row equivalence.

Proof.

We require to show that any difference,

$$(a_i-a_j), \qquad a_i a_j \in X ,$$

is also relatively prime to m.

From Theorem 11.4.1. we know every $a_i \in X$ is relatively prime to m, and since any difference,

$(a_i-a_j) \in X$, also, then every difference gives,

$((a_i-a_j),m) = 1.$

Q.E.D.

The array generated in example 11.1.1. is an example of the array specified in Theorem 11.4.2. but the following two theorems give another set of a_i's which generate non-'cyclic' arrays.

Theorem 11.4.3.

If an integer m, is expressed as a product of its primes,

$$m = p_1^{\alpha_1} \; p_2^{\alpha_2} \; \ldots\ldots p_r^{\alpha_r}$$

with $p_1 < p_2 < \ldots\ldots < p_r$

then the set of integers, with $b \leqslant (p_1-1)$

$$a_i,\ 2a_i,\ 3a_i,\ \ldots\ldots\ ,\ ba_i$$

where $a_i < m$ and $(a_i,m) = 1$, is a set of b distinct integers, relatively prime to m, when reduced modulo (m).

Proof.

Theorem 11.4.1. shows that any member of the set, $1,2,\ldots,p_1-1$ is relatively prime to m, and since $(a_i,m) = 1$, and the product of two relative primes, is also relatively prime, any member of the set a_i, $2a_i,\ldots,ba_i$ is relatively prime to m.

It remains to be shown that the residues are distinct, and from Theorem 11.2.2.1. it can be seen that the set is a subset of the set,

$$a_i,\ 2a_i,\ 3a_i,\ldots\ldots,(m-1)a_i$$

which has incongruent residues mod (m) if $(a_i,m) = 1$.

Q.E.D.

We must now show that the array generated by the above set of relative primes, is free from A.R.E.

Theorem 11.4.4.

The array generated modulo (m) by a set of relative primes, with $b \leqslant (p_1-1)$,

$$a_i,\ 2a_i,\ldots\ldots,\ ba_i$$

where p_1 is the smallest prime factor of m, is free from A.R.E.

Proof.

We must show that any difference is also relatively prime.

Let, $(ha_i - da_i) = ca_i$, $h > d$

$= (h-d)a_i$

since $h,d \in \{1,2,\ldots, (p_1-1)\} = X$

then $(h-d) \in X$.

From Theorem 11.4.1., any $x \in X$ is relatively prime to m, therefore,

$$((h-d),m) = 1$$

Q.E.D.

Therefore either of the two sets

$\{1,2,\ldots,b\}$, $b \leqslant (p_1-1)$

$\{a_i, 2a_i,\ldots, ba_i\}$, $b \leqslant (p_1-1)$, $(a_i,m) = 1$.

can be used to construct non-'cyclic' arrays. Provided k_o is small, the resulting codes may be practical to implement.

11.5 Encoding and Decoding.

11.5.1. Introduction.

If one considers the first column of a cyclic array, it can be arranged in the following form.

$$\{a, a^2, a^3,\ldots, a^S\} \mod (m).$$

If a is a primitive root of m, every integer, $1 \leqslant x \leqslant (m-1)$, can be expressed as a power of a, such that if

$$x \equiv a^r \mod (m)$$

then we say $r = \text{ind}_a(x)$.

Therefore when we wish to form column x, of our array, we obtain the set below,

$$\{ax, a^2x, a^3x, \ldots, a^Sx\} \bmod (m).$$

But since $x \equiv a^r \bmod (m)$, the set becomes,

$$\{a^{r+1}, a^{r+2}, a^{r+3}, \ldots, a^{r+S}\} \bmod m,$$

and since,

$$a^{S+1} \equiv a \bmod (m)$$

the x'th column is a cyclic shift of the first column by $\text{ind}_a(x)$ positions.

However if,

$$\text{Ord}_m a = S < \phi(m)$$

then a is not a primitive root of m and every integer, $1 \leqslant x \leqslant (m-1)$, cannot be expressed as a power of a. The integer a now organizes a subset of the set of all relative primes to m, into a subgroup under the operation of multiplication mod (m).

Let $A = \{a^1, a^2, a^3, \ldots, a^S\}$, $S < \phi(m)$, be the 'cyclic' set generated by a, mod (m) then if $x \in A$, column x is a 'cyclic' shift of A, written $(A)^i$ where

$$x \equiv a^i \bmod (m) \quad \text{or} \quad i = \text{ind}_a(x).$$

If, $x \notin A$ then column x is a coset of the subgroup generated by a, mod (m), and is written xA, where

$$xA = \{xa, xa^2, xa^3, \ldots, xa^S\} \bmod (m).$$

If, $y \notin A$, but $y \in xA$, then let $y \equiv xa^i \bmod (m)$, $a^i \in A$ so that column y is given by the coset,

$$yA = xa^iA = x(A)^i$$

and column y is a 'cyclic' shift of the coset column generated by xA.

Therefore in an array of order S, there will be (m-1)/S basic cosets as columns (including the 'cyclic' set A considered as the coset 1·A) and all other columns are 'cyclic' shifts of these.

However before we consider methods of encoding and decoding, we must mention that the form of the arrays, as discussed so far, is not the final form used explicity for the implementation of the codes. The arrays are modified by the following mappings,

(a) if $\mathrm{Ord}_m a = S = \phi(m)$,

then for each element x of the array we apply the mapping,

$$x \longrightarrow \mathrm{ind}_a(x) \qquad 11.5.1.1.$$

(b) if $\mathrm{Ord}_m a = S < \phi(m)$,

then for each element x of the 'cyclic' generating set and its 'cyclic' shifts, we apply the mapping,

$$x \longrightarrow \mathrm{ind}_a(x) \quad . \qquad 11.5.1.2.$$

For the cosets x_2A, $x_3A, \ldots, x_{(m-1)/S}A$, we apply the mappings,

$$\begin{aligned} x_2A &\longrightarrow \{S+1,\ S+2, \ldots., 2S\} = B_2 \\ x_3A &\longrightarrow \{2S+1,\ 2S+2, \ldots., 3S\} = B_3 \\ &\vdots \\ x_{(m-1)/S}A &\longrightarrow \{m-1-S,\ m-S, \ldots, m-1\} = B_{(m-1)/S} \end{aligned} \qquad 11.5.1.3$$

For the columns, $x_2A^i, \ldots, x_{(m-1)/S}A^i$ we use the mappings,

$$\begin{aligned} x_2A^i &\longrightarrow B_2{}^i \\ x_3A^i &\longrightarrow B_3{}^i \\ &\vdots \\ x_{(m-1)/S}A^i &\longrightarrow B^i_{(m-1)/S} \end{aligned} \qquad 11.5.1.4.$$

This results in an array whose rows retain their leftwise sequence properties but whose columns are now 'cyclic' shifts of sets of consecutive integers.

The examples below demonstrate both cases for $S = \phi(m)$ and $S < \phi(m)$.

Example 11.5.1.1.

Consider the array generated by the cyclic set generated by 3, mod (7).

3	6	2	5	1	4
2	4	6	1	3	5
6	5	4	3	2	1
4	1	5	2	6	3
5	3	1	6	4	2
1	2	3	4	5	6

Since $\mathrm{Ord}_7 3 = 6 = \phi(m)$ we apply the mappings from 11.5.1.2.

$x \longrightarrow \mathrm{ind}_3(x)$, that is

$3 \equiv 3^1$ mod (7)	$3 \longrightarrow 1$
$2 \equiv 3^2$ mod (7)	$2 \longrightarrow 2$
$6 \equiv 3^3$ mod (7)	$6 \longrightarrow 3$
$4 \equiv 3^4$ mod (7)	$4 \longrightarrow 4$
$5 \equiv 3^5$ mod (7)	$5 \longrightarrow 5$
$1 \equiv 3^6$ mod (7)	$1 \longrightarrow 6$

Applying these mappings gives the array,

1	3	2	5	6	4
2	4	3	6	1	5
3	5	4	1	2	6
4	6	5	2	3	1
5	1	6	3	4	2
6	2	1	4	5	3

11.5.1.5.

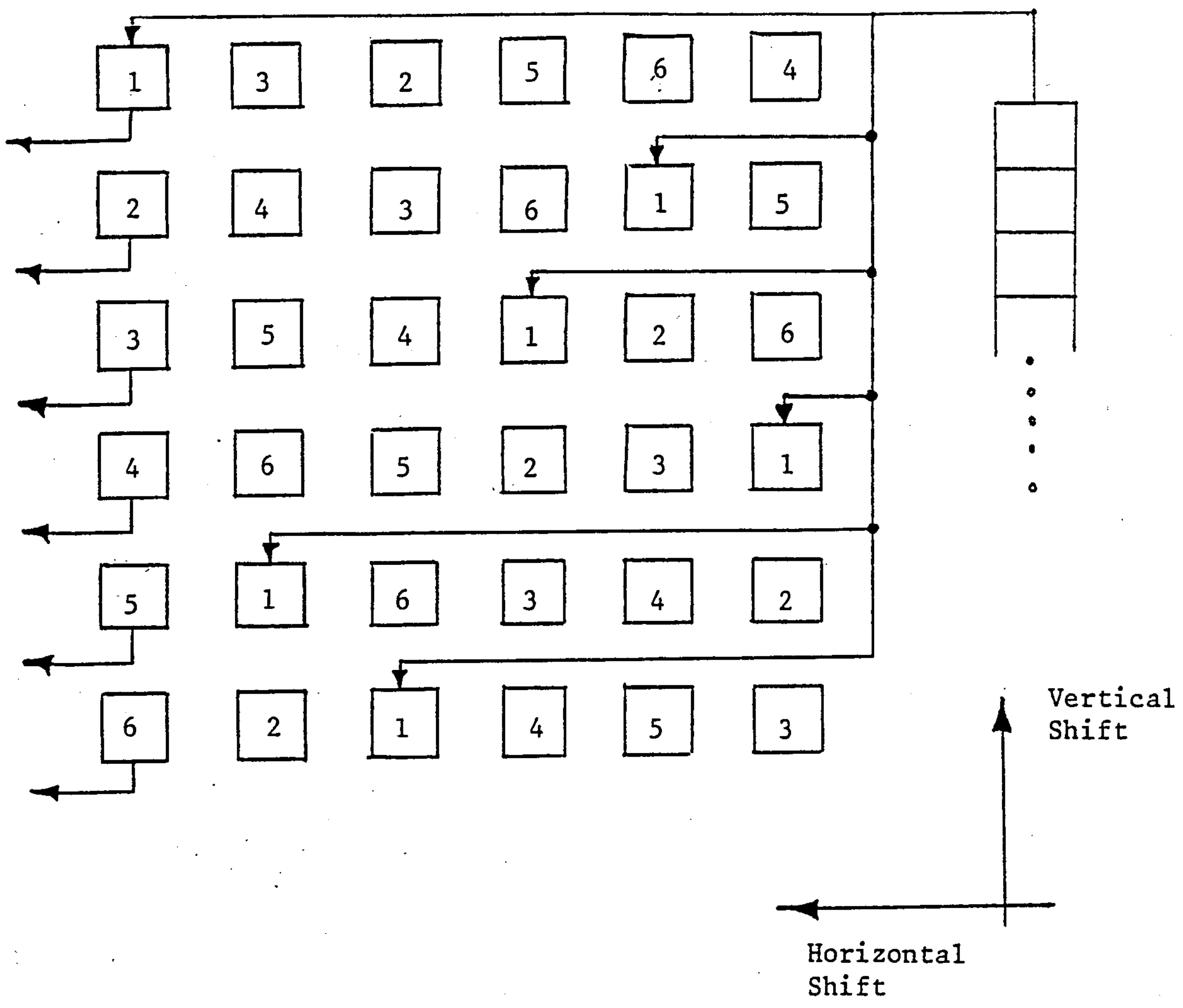


FIG. 11.5.1.1.

ENCODING ARRAY FOR (72,36) CODE.

And this form is used to encode and decode.

Consider the array generated by the 'cyclic' set generated by 2, mod (7)

2	4	6	1	3	5
4	1	5	2	6	3
1	2	3	4	5	6

Since $\text{Ord}_7 2 = 3 < \phi(m)$, the array will have $(m-1)/S = 6/3 = 2$ distinct cosets and all columns are 'cyclic' shifts of these basic columns. For the 'cyclic' set we use the mapping 11.5.1.2,

$2 \equiv 2^1 \mod (7)$ $\quad\quad 2 \longrightarrow 1$

$4 \equiv 2^2 \mod (7)$ $\quad\quad 4 \longrightarrow 2$

$1 \equiv 2^3 \mod (7)$ $\quad\quad 1 \longrightarrow 3$

Since $x_2A = \{6\ 5\ 3\}$, from 11.5.1.3.

$6 \longrightarrow S+1 = 4$

$5 \longrightarrow S+2 = 5$

$3 \longrightarrow S+3 = 6$

This gives the modified array,

1	2	4	3	6	5
2	3	5	1	4	6
3	1	6	2	5	4

which is used for encoding and decoding.

11.5.2. Encoding.

The encoding scheme presented in this section is only one method by which the encoding can be achieved. Nevertheless it illustrates the idea and demonstrates a serial 'cyclic' or parallel encoding method.

Consider the array 11.5.1.5. in example 11.5.1.1. and the 6 × 6 array of storage elements in figure 11.5.1.1. If we associate one storage element with one integer of the array then the encoding is performed, using a serial input stream, in the following way.

(a) The first message-digit is put into the storage elements corresponding to integer 1. (Connections as shown in figure 11.5.1.1.)

(b) The array of storage elements is clocked once vertically and the second message-digit is put into the storage elements, which now can be thought to correspond to the integer 2.
The elements in the top rows being returned to the bottom row.

(c) The process is continued until the array of storage elements has been clocked 6 times, when it will contain message-digits 1 to 6 in the distribution shown by the array of integers.

(d) The array of storage elements is then clocked once to the left, horizontally.

(e) The process is continued by repeating (a) to (d) for the six message-digits of the second sub-block to be encoded and all subsequent sub-blocks of message-digits.

The array of storage elements is used to form the encoded check-digits which can be obtained in the following ways.

i) Serially, from the output of the storage element corresponding to integer 1 in the first column, when it is clocked vertically, or

ii) In parallel, from the horizontal outputs of all storage elements in the first column when (d) is performed in the operations.

Encoding using a parallel message-digit input is performed simply as follows.

iii) All six message-digits from the first sub-block are loaded simultaneously into their positions in the storage element array corresponding to the integers in array 11.5.1.5.

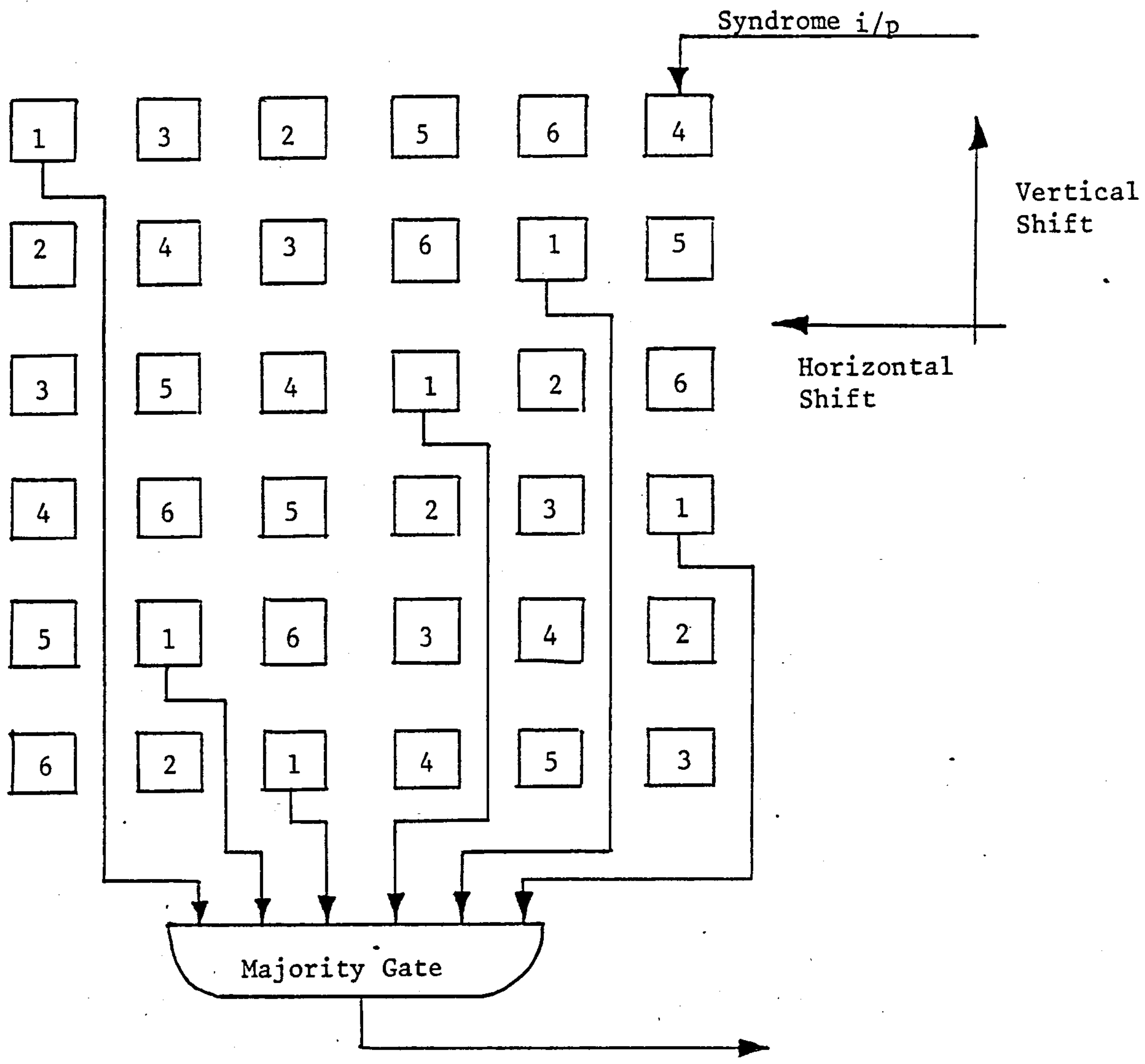


FIG. 11.5.3.1.

DECODING ARRAY FOR (72,36) CODE.

iv) The storage array is clocked once leftwise horizontally for parallel output or vertically 6 times for serial output from the storage element in column one corresponding to integer 1.

v) Repeat (iii) and (iv) for each sub-block of message-digits.

If the integer array has (m-1)/S coset columns then the message-digit sub-block must be sub-divided into (m-1)/S sub-sub-blocks and each sub-sub-block is simultaneously serially loaded into the encoding storage array. Otherwise the process is the same and can be done serially or in parallel.

11.5.3. Decoding.

The following decoding method is one way in which 'cyclic' decoding may be performed and is shown for the code specified by array 11.5.1.5.

Figure 11.5.3.1. shows the form of the decoding storage array and associated majority-logic unit only. Figure 11.5.3.2. shows a block diagram of the complete decoder.

As each sub-block of received message-digits arrives, it is re-encoded and added to its corresponding received check-digits to form the syndrome for that block. All the convolutional relationships of the code are inherent in the syndrome digits so that the storage array in figure 11.5.3.1. is really a modified buffer. To fill the buffer serially we input the syndrome digits of a sub-block at the point shown and clock vertically 6 times. Then clock horizontally leftwards and serially input the next set of syndrome digits from the following sub-block. Alternatively when the syndrome digits are obtained in parallel, they are put into the storage elements of the righthand column of figure 11.5.3.1. and the array of storage elements

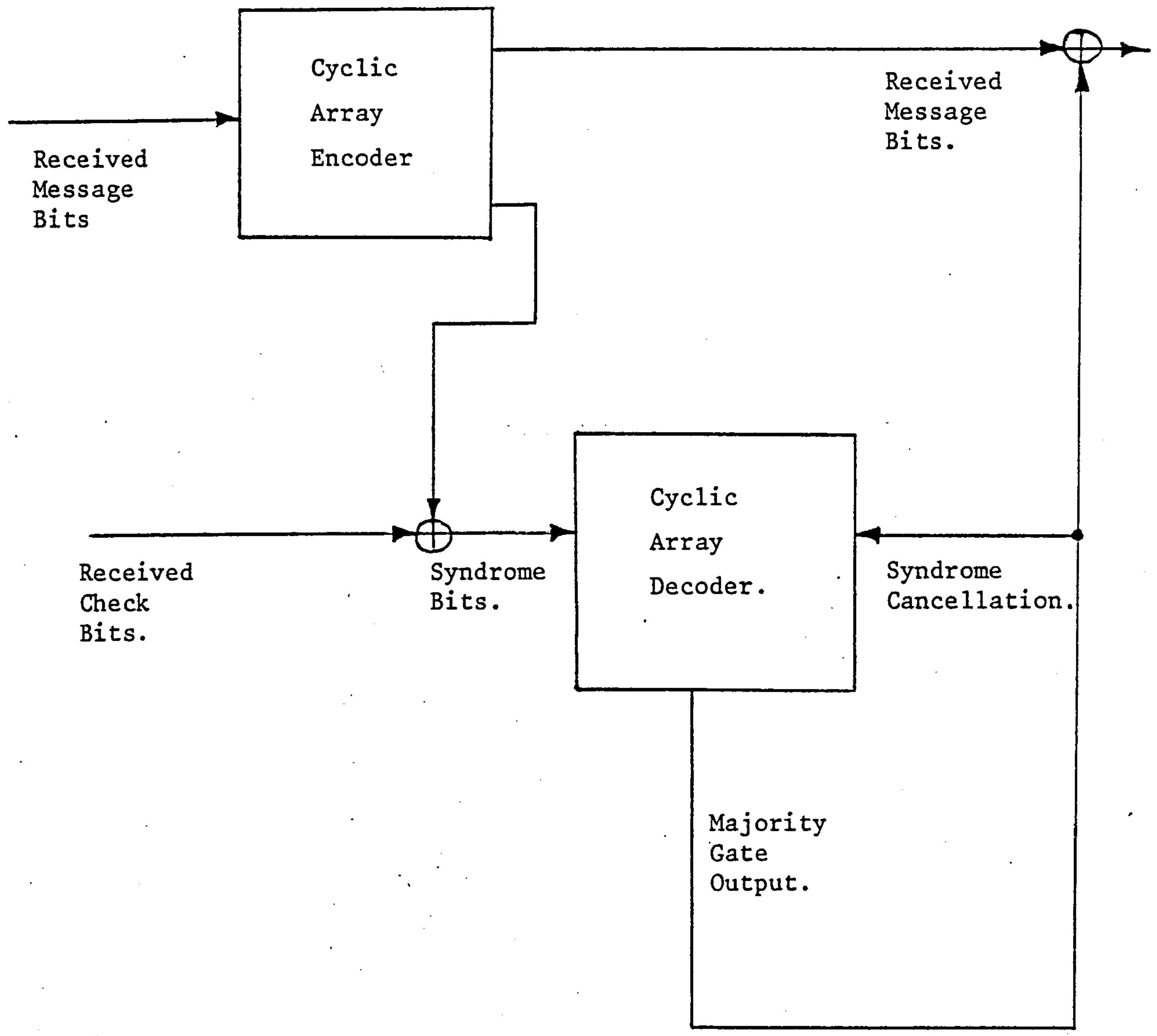


FIG. 11.5.3.2.

A GENERAL 'CYCLIC' DECODER.

is clocked leftwise horizontally once before accepting the next set of syndrome digits.

Nevertheless by the time the syndrome digits of some general block e reach the leftmost column, the estimates of the errors in the message-digits of block e are located at those points indicated by superimposing the integer array on the storage array, as shown in figure 11.5.3.1. We obtain the 6 self-orthogonal estimates of the first error as shown and clock the array vertically 6 times to sequentially decode all 6 errors. Thus the necessity for having columns of consecutive integers using the mappings indicated in section 11.5.1. Also we can now see that if the array contains (m-1)/S cosets as columns, we will have (m-1)/S sets of distinct integers and therefore we require (m-1)/S majority-logic gates to decode cyclically.

11.6 The Code Parameters.

The parameters of interest in a random error-correcting convolutional code are;

(i) d_m, minimum distance.

(ii) n_o, sub-block length.

(iii) k_o, number of sub-block information digits.

(iv) N, block constraint length.

(v) R, information rate of transmission.

Since we are defining the parameters of codes that can be decoded cyclically, we restrict the parameters to codes developed from odd integers only.

i) Minimum distance, d_m.

From section 10.4. we know that the minimum distance of a random error-correcting code is equal to the minimum weight of an initial n_oN-digit code sequence whose 0'th block is non-zero. Because no two message-digits from block 0 are ever in the same check-digit, the minimum weight code sequence whose 0'th block is non-zero, is that whose 0'th message block has weight one.

Thus the minimum distance of the codes is simply equal to the number of times a single message-digit appears in a constraint length of n_oN digits.

Since each row of an array contains a permutation on the set of integers 1,2,....,(m-1), if there are S rows then the code contains S+1 appearances of all message-digits from a given block over a constraint length of n_oN digits.

Thus the minimum distance is given by;

$$d_m = S + 1 = n_o - k_o + 1 .$$

ii) Sub-block length n_o.

The number of check-digits in a sub-block is equal to S the number of rows in an array. Therefore,

$$n_o = k_o + S.$$

iii) Sub-block information digits, k_o.

This is equal to the number of distinct integers in the array and is determined by the chosen integer m, for the array reduced mod (m), so that,

$$k_o = m - 1.$$

iv) Block constraint length, N.

This is given by the number of columns in an array and is equal to the number of integers in the rows, which is m-1. Therefore,

$$N = m - 1.$$

v) Transmission rate, R.

This is given by the ratio,

$$R = \frac{k_o}{n_o} = \frac{k_o}{k_o + S} = \frac{m - 1}{m - 1 + S}$$

The rate therefore is a minimum when $S = k_o$ and a maximum when $S = 2$. Therefore

$$R_{max} = \frac{m - 1}{m + 1} .$$

We noted in earlier sections that given an array with S rows we could obtain $J = S$, self-orthogonal estimates of every message-digit in a current block. Therefore $d_m = J + 1$ and the codes are self-orthogonal up to their minimum distance and are one-step cyclically majority-logic decodable, using $\frac{m-1}{S}$ majority gates.

We can now formally define the codes as follows.

Definition 11.6.1.

Let m be any odd positive integer which can be factored into its primes,

$$m = p_1^{\alpha_1} \, p_2^{\alpha_2} \, \ldots\ldots p_r^{\alpha_r}$$

where,

$$p_1 < p_2 < \ldots\ldots < p_r$$

$$0 \leq \alpha_1, \alpha_2, \ldots\ldots, \alpha_r \leq \infty , \quad \text{but } m \neq 1$$

and let S be any positive integer such that,

$$S \leq (p_1 - 1)$$

and

$$S \mid \phi(p_i^{\alpha_i})$$

for all $1 \leq i \leq r$.

Then there exists a self-orthogonal one-step majority-logic decodable convolutional code with the parameters,

$$n_o = k_o + S$$

$$d_m = S + 1$$

$$N = m - 1$$

$$k_o = m - 1$$

which can be cyclically decoded with (m-1)/S majority-logic dates.

CHAPTER 12

12. PERFORMANCE OF THE CODES.

12.1 Introduction.

In this section we examine some aspects of the codes which give insight into their performance and enable comparison with other codes in ways other than minimum distance, rate and constraint length $n_o N$.

A parameter of interest which will be found useful in other sections is effective constraint length, n_e. In section 12.2, we develop expressions for n_e which enable its calculation quite simply from the 'cyclic' set which generates the array.

Peterson and Weldon[19] pp.104-105 showed that on the binary symmetric channel, B.S.C., if we assume that unlimited or catastrophic error propagation never occurs then the probability of incorrectly decoding some e'th block can be approximated quite closely by the probability of incorrectly decoding the first block transmitted. For the 'cyclic' codes we assume in section 12.3 that an error pattern of weight $(t+x) > t$ has occurred and calculate the probability of incorrectly decoding one digit from the sub-block. Assuming a B.S.C. means each error pattern of weight $(t+x)$ is equally likely and we only need to know how many patterns, from all possible, cause incorrect decoding to find the probability of incorrectly decoding that digit.

Being self-orthogonal the codes are naturally free from unlimited or catastrophic error propagation so that a close approximation can be made by considering the probability of erronously decoding the first block transmitted. Considering the first block transmitted is another way of saying we can consider any block provided error free decoding occurred in all previous N-1 blocks; that is, there is no error-propagation. Assuming that all previous blocks have been correctly decoded means that the only digits involved in the decoding of a digit are those in the effective constraint length n_e. It will be seen that n_e is strongly related to the probability of incorrectly decoding a digit, under these assumptions.

In section 12.3 we calculate the probabilities for a 'cyclic' code and compare these calculations with others for two self-orthogonal codes from the class of Robinson and Bernstein[7].

In general, where the 'cyclic' codes can be compared with those of Robinson and Bernstein, for a given t, the 'cyclic' codes constraint length n_oN is longer, although its effective constraint length n_e is shorter or equal. In section 12.4 we ask the question, is the 'cyclic' codes' increased length n_oN justified in terms of the reduction in the probability of an incorrect decoding?

Although the 'cyclic' codes are longer than those of Robinson and Berstein[7] they are shorter than many of the very high rate codes presented by Wu.[22,23,24]

Finally in section 12. we consider the problem of error propagation.

We define the autonomous case, assumed in our definition of propagation length L, and introduce the concept of propagation efficiency. We then calculate L for the 'cyclic' codes and compare these values with values of L calculated for Robinson and Bernstein's codes, under the same assumptions.

12.2 Effective Constraint Length.

Effective constraint length was first considered by Massey[1] who defined it as;

"The maximum number of bits which can influence the threshold decoding of any one message-bit in the first k_o message-bits of a sub-block".

In the following we will show that any code specified by an array, generated by a 'cyclic' set

$$A = (a, a^2, a^3, \ldots, a^S) \bmod (m),$$

with cosets,

$$x_2 A = (x_2 a, x_2 a^2, \ldots, x_2 a^S) \bmod (m)$$
$$\vdots$$
$$x_{m-1/S} A = (x_{m-1/S} a,\ x_{m-1/S} a^2, \ldots, x_{m-1/S} a^S) \bmod (m)$$

has effective constraint length, n_e, where

$$n_e = \max (n_{e_1}, n_{e_2}, \ldots, n_{e_{m-1/S}}) \qquad 12.2.1.$$

and

$$n_{e_i} = 1 + \sum_{j=1}^{S} x_i\ a^j \bmod (m) \qquad 12.2.2.$$

where;

x_i = any integer contained in the i'th coset, $x_i A$, though usually the smallest.

$$i = 1, 2, \ldots, \frac{m-1}{S}\ .$$

The specification of x_i arises from the closure properties of the 'cyclic' set and its cosets, developed in section 11.4.1. That is we take an array, before performing the mappings in section 11.4, and add up the elements in each of the m-1/S distinct cosets, then add one to give $\frac{m-1}{S}$ distinct coset sums. Then n_e is equal to the maximum coset sum.

To show this let us assume we wish to discover the effective constraint length involved in the decoding of some integer (message-digit error) b which is present in some coset x_iA. The coset x_iA and its cyclic shifts will form columns y of the array, where $y \in x_iA$, so that the integer b will be in those columns $y \in x_iA$.

If b is in column y then there are (y-1) integers to the left of b, which are message-digit errors from other blocks, in that check sum on b.

Since b occurs in columns x_ia mod (m), x_ia^2 mod (m) etc., the total number of message-digit errors from other blocks, which can influence all S check sums on b is

$$\sum_{j=1}^{S} (x_ia^j \bmod(m) - 1)$$

If we consider the S check-digits containing the check sums on b, plus of course b itself, then

$$n_{e_i} = 1 + S + \sum_{j=1}^{S} (x_ia^j \bmod(m) - 1) \qquad 12.2.3.$$

And equations 12.2.2. and 12.2.3. are identical.

If $S = \phi(m)$, then

$$x_1 = x_2 = x_3 = \ldots\ldots = x_{m-1/S} = 1$$

$$n_{e_1} = n_{e_2} = n_{e_3} = \ldots\ldots = n_{e_{m-1/S}} = n_e$$

and

$$n_e = 1 + \sum_{j=1}^{S} a^j \bmod (m) \qquad 12.2.4.$$

The equations above assume, with feedback decoding, that all previous decodings were correct or, equivalently, no error propagation has occurred.

The following example illustrates these results.

<u>Example 12.2.1.</u>

Let m = 7 and consider the code specified by the array generated by the 'cyclic' set,

$$A = (3, 3^2, 3^3, 3^4, 3^5, 3^6) \bmod (7)$$

reduced mod (7), this becomes the set,

$$(3, 2, 6, 4, 5, 1).$$

For this code, $S = 6 = \phi(m)$ so that from equation 12.2.4.,

$$n_e = 1 + \sum_{j=1}^{6} 3^j \bmod (7)$$

$$= 1 + (3 + 2 + 6 + 4 + 5 + 1)$$

$$= \underline{\underline{22}}$$

Consider the code specified by the array generated by the cyclic set,

$$A = (2, 2^2, 2^3) \bmod (7).$$

Since $S < \phi(m)$, we require all x_i up to $i = m\text{-}1/S = 2$. From equation 12.2.2., let $x_1 = 1$, then

$$n_{e_1} = 1 + \sum_{j=1}^{3} 2^j \bmod (7)$$

$$= 1 + (2 + 4 + 1) = \underline{\underline{8}}$$

Since there are only two cosets, we know that the other coset contains the integers 3,5 and 6, and since $3 \in x_2A$, is the smallest integer in x_2A, from equation 12.2.2.,

$$n_{e_2} = 1 + \sum_{j=1}^{3} 3 \cdot 2^j \bmod (7)$$

$$= 1 + (6 + 5 + 3) = \underline{\underline{15}}$$

Therefore,

$$n_e = \max (n_{e_1}, n_{e_2})$$

$$= \max (8, 15)$$

$$= \underline{\underline{15}}$$

Finally, if

$$A = (6, 6^2) \bmod (7)$$

then we have,

$$n_{e_1} = 1 + \sum_{j=1}^{2} 6^j \bmod (7)$$

$$= 1 + (6 + 1) = \underline{\underline{8}}$$

since in this case $x_2 = 2$,

$$n_{e_2} = 1 + \sum_{j=1}^{2} 2 \cdot 6^j \bmod (7)$$

$$= 1 + (5 + 2) = \underline{\underline{8}}$$

and $x_3 = 3$,

$$n_{e_3} = 1 + \sum_{j=1}^{2} 3 \cdot 6^j \bmod (7)$$

$$= 1 + (4 + 3) = \underline{\underline{8}}$$

And therefore, $n_e = \underline{\underline{8}}$.

Table 12.2.1., shows the effective constraint lengths of cyclic codes of various rates and error-correcting capability, compared with other codes.

TABLE 12.2.1.

Effective constraint lengths n_e.

J	n_e 'Cyclic'	n_e C.S.O.C.$^{7}_{1}$	n_e C.S.O.C.$^{15}_{2}$	Rate
2	4	4	4	1/2
4	11	11	11	1/2
6	22	22	22	.
8		37	37	.
10	56	56	56	.
12	79	79		.
14		106		.
16	137	137		.
18	172	172		.
20		211		.
22	254	254		.
24		301		.
26				.
28	406			1/2
2	6	7		2/3
4		20	20	.
6	40	41	41	.
8	69	70	72	.
10		111		.
12		152		.
14	204	218		.
16				.
18	334			2/3
2	8	10		3/4
4	27	31	32	.
6	58	66	63	.
8		105		.
10	156	181		.
12	223			3/4

J	n_e 'Cyclic'	n_e C.S.O.C.$_1^7$	n_e C.S.O.C.$_2^{15}$	Rate
2	10	13		4/5
4	35	39	40	.
6		83		.
8		161		.
10	206			4/5

J	n_e 'Cyclic'	n_e C.S.O.C.$_3^{22,23}$	Rate
26	690	743	2/3
3	75	82	12/13
5	184	209	12/13
6	238	339	13/14

12.3 Unbounded Probabilities of a Decoding Failure.

We wish to be able to calculate the probability that the decoder will fail to correctly decode a message-digit assuming an error pattern of weight (t+x) has occurred. We do this by calculating the number of error patterns which cause $\geqslant J/2$ check sums to be in error, with equality when the digit to be decoded is in error also. If we assume each error pattern is equally likely then we can express the probability of error in the decoding of some digit from the coset $x_i A$, as

$$p_{x_i A}(t+x) = \frac{N(e_i)_{t+x}}{{}^{n_o N}N_{t+x}} \qquad 12.3.1.$$

where;

$${}^{n_o N}N_{t+x} = \frac{n_o N!}{(t+x)!\ (n_o N - t-x)!}$$

is the number of possible patterns of weight t+x, and

$N(e_i)_{t+x}$ = the number of error patterns of weight t+x which cause a decoding failure on the digit from the coset $x_i A$ under consideration.

The maximum number of digits from the constraint length $n_o N$ which affect the decoding of a digit is the effective constraint length n_e. However the number of digits involved in the decoding of some digit from the coset $x_i A$ is n_{e_i} given in equation 12.2.2. When t+1 errors occur they must affect t+1 check sums, to cause a decoding failure, and must therefore occur within the n_{e_i} digits which compose the check sums. However if t+x errors occur, for every t+x-i, $i < x$, errors within the n_{e_i} digits there are i errors within the $(n_oN - n_{e_i})$ digits, not concerned in the check sums, to consider as erronous patterns.

Let $N'(e_i)_{t+x-i}$ be the number of error patterns of weight t+x-i which affect the check sums, then,

$$N(e_i)_{t+x} = N'(e_i)_{t+x} + \sum_{j=1}^{x-1} \frac{(n_oN - n_{e_i})!}{j!(n_oN - n_{e_i} - j)!} N'(e_i)_{t+x-j} \qquad 12.3.2.$$

When $x = 1$,

$$N(e_i)_{t+1} = N'(e_i)_{t+1} \ .$$

When $x = 2$,

$$N(e_i)_{t+2} = N'(e_i)_{t+2} + \frac{(n_oN - n_{e_i})!}{1!(n_oN - n_{e_i} - 1)!} N'(e_i)_{t+1}$$

Obviously n_{e_i} is an important parameter as it also affects the value of $N'(e_i)_{t+x}$. However it is the distribution of the n_{e_i} digits among the check sums, rather than the value of n_{e_i}, which determines the value of $N'(e_i)_{t+x}$. Therefore we must resort to specific examples to examine how the probability varies with x. Nevertheless two general statements can be made.

i) If $n_{e_i} >> n_{e_j}$

$$N'(e_i)_{t+x} > N'(e_j)_{t+x}$$

$$12.3.3.$$

ii) $$N'(e_i)_{t+x} < \frac{(n_{e_i})!}{(t+x)!\ (n_{e_i} - t-x)!}$$

The probability of erronous decoding is different then for digits from different cosets. However from the development of equation 12.2.3. we know that, not only is n_{e_i} the same for every digit in a coset but the form of the check sums (number of digits in each sum) is the same for every digit in a coset. Thus $N'(e_i)_{t+x}$ is the same for every digit and therefore so is $N(e_i)_{t+x}$ and the probability $p_{x_iA}(t+x)$.

However in example 12.3.1. which follows, $S = \phi(m)$ so that n_{e_i}, the distribution of the n_{e_i} digits in the check sums, and therefore the probability of an erronous decoding, are the same for every i and therefore every digit in a sub-block. Two specimen calculations are given for (t+1) and (t+2) errors and the other results are presented up to (t+6) errors.

Example 12.3.1.

The code chosen is;

$$t = 3, \quad n_oN = 72, \quad \text{Rate} = 1/2.$$

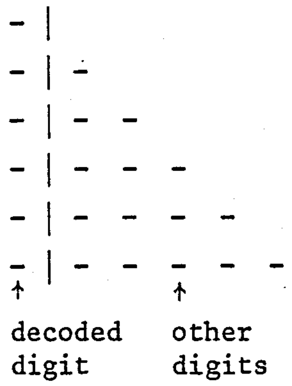
The 'cyclic' array has the following form;

$$\begin{array}{cccccc} 3 & 6 & 2 & 5 & 1 & 4 \\ 2 & 4 & 6 & 1 & 3 & 5 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 4 & 1 & 5 & 2 & 6 & 3 \\ 5 & 3 & 1 & 6 & 4 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

The leftwise sequences or check sums for digit one are;

```
1
1  4
1  3  5
1  6  4  2
1  5  2  6  3
1  2  3  4  5  6
```

Equivalent form.



To find the number of error patterns which cause decoding failure we only need the distribution of the check sums and this equivalent form is given here.

Since for this code $S = \phi(m)$ the equivalent form is the same for the check sums of every message-digit in a sub-block. The number of digits involved in the check sums is given by the effective constraint length n_e.

The value of $N(e_i)_{t+x}$ is of course influenced by the value of n_e, but it is also influenced by the equivalent form of the check sums or their distribution.

To calculate $N(e_i)_{t+x}$ we proceed in three stages,

i) calculate the number of erronous patterns occurring in the message-digit errors only. We include the decoded digit and denote this calculation by the symbol, (M_{t+x}).

ii) calculate the number of erronous patterns in the check-digit errors only, denoting this by the symbol (C_{t+x}).

iii) calculate the number of erronous patterns occurring in combinations of message-digit errors and check-digit errors. This is denoted by the symbol, $(C_i \; M_{t+x-i})$ for $i = 1,2,\ldots,(t+x-1)$, where this implies that i and t+x-i errors are present in the respective digits.

<u>A) (t+1) errors.</u>

a) (t+1) errors in the message-digit errors only (M_4).

a.1) the decoded digit is not in error.

In this case the 4 errors must be among the interfering message-digit errors and constrained to one error per check-sum or row of the equivalent form. If there are N(x) other digits in row x of the equivalent form, then with four errors in say rows t,u,v and w, the number of erronous patterns is given by,

$$= N(t)\cdot N(u)\cdot N(v)\cdot N(w).$$

All combinations of four products from the 5 rows will give the total for a.1.

$$\begin{array}{rcr}
1\cdot 2\cdot 3\cdot 4 & = & 24 \\
1\cdot 2\cdot 3\cdot 5 & = & 30 \\
1\cdot 2\cdot 4\cdot 5 & = & 40 \\
1\cdot 3\cdot 4\cdot 5 & = & 60 \\
2\cdot 3\cdot 4\cdot 5 & = & \underline{120} \\
 & & \underline{\underline{274}}
\end{array}$$

a.2) the decoded digit is in error.

In this case one of the four errors is the decoded digit itself and the other three errors, constrained to one per row, cause it to be decoded as zero. The number of patterns is given by all possible products of the three N(x)'s.

$$\begin{array}{rcr@{\qquad\qquad}rcr}
1\cdot 2\cdot 3 & = & 6 & 1\cdot 3\cdot 4 & = & 12 \\
1\cdot 2\cdot 4 & = & 8 & 1\cdot 3\cdot 5 & = & 15 \\
1\cdot 2\cdot 5 & = & 10 & 2\cdot 4\cdot 5 & = & 40 \\
2\cdot 3\cdot 4 & = & 24 & 1\cdot 4\cdot 5 & = & \underline{20} \\
2\cdot 3\cdot 5 & = & 30 & & & \underline{87} \\
3\cdot 4\cdot 5 & = & \underline{60} & & & \\
 & & \underline{138} & & &
\end{array}$$

$$= 138 + 87 = \underline{\underline{225}}$$

b) (t+1) errors in the check-digit errors only. (C_4)

There are only 6 check-digits used so that the number of erronous patterns possible are,

$$= \frac{6!}{4!\ 2!} = \underline{\underline{15}}$$

c) (t+1) errors in the message-digits and check-digits.

c.1) One check-digit error and 3 message-digit errors. ($C_1\ M_3$)

c.1.1.) ($C_1\ M_3$, decoded digit not in error)

The message-digit errors will assume all patterns found in a.2. Each individual pattern occupies three rows so that the single check-digit error can be in any of the remaining 3 rows. Thus the number of erronous patterns

$$= 3 \times 225 = \underline{\underline{775}}$$

c.1.2.) ($C_1\ M_3$, decoded digit in error)

Since one error is fixed in the decoded digit this leaves 2 errors to distribute among the interfering message-digit errors. The total number of patterns due to message errors is given by all products of two N(x)'s.

$1\cdot 2 = 2$	$2\cdot 3 = 6$	$3\cdot 4 = 12$	$4\cdot 5 = 20$
$1\cdot 3 = 3$	$2\cdot 4 = 8$	$3\cdot 5 = 15$	
$1\cdot 4 = 4$	$2\cdot 5 = 10$	27	
$1\cdot 5 = 5$	24		
14			

$$= 14 + 24 + 27 + 20 = 85$$

And for each of these patterns the check-digit error can be in any of the four unused rows, giving a total of

$$= 4 \times 85 = \underline{\underline{340}}$$

c.2) Combinations of C_2 M_2.

c.2.1.) (C_2 M_2, decoded digit not in error.)

There are 85 patterns of two errors in the message-digit errors and for each pattern there are all combinations of 2 check-digit errors from the 4 unused check sums giving

$$= \frac{4!}{2!\ 2!} \cdot 85 = \underline{\underline{510}}$$

c.2.2.) (C_2 M_2, decoded digit in error.)

This leaves one message-digit error to be present in the other message-digit errors and for each such pattern, 2 check errors from 5 unused check sums giving,

$$= \frac{5!}{2!\ 3!} \cdot 15 = \underline{\underline{150}}$$

c.3) Combinations of C_3 M_1.

c.3.1.) (C_3 M_1, decoded digit not in error.)

The approach should now be clear!

$$= \frac{5!}{3!\ 2!} \cdot 15 = \underline{\underline{150}}$$

3.3.2.) (C_3 M_1, decoded digit in error.)

$$= \frac{6!}{3!\ 3!} = \underline{\underline{20}}$$

There are no other patterns possible, and we have the total as;

$$N'(e_i)_{t+1} = 274 + 225 + 15 + 775 + 340 + 510 + 150 + 150 + 20$$

$$= \underline{\underline{2459}}$$

Since x = 1,

$$N(e_i)_{t+1} = N'(e_i)_{t+1} = 2459$$

and,

$$N^{72}_{t+1} = N^{72}_{4} = \frac{72!}{4!\ 68!}$$

$$= \underline{\underline{1,028,790}}$$

and therefore,

$$p(4) = \frac{2459}{1,028,790}$$

$$= \underline{\underline{23\cdot 9 \cdot 10^{-4}}}$$

<u>B) (t+2) errors.</u>

Comment; If two errors occur in one row of the equivalent form they cancel leaving t errors which can be correctly decoded. We therefore only consider patterns which can affect 4 or 5 distinct rows.

a) (t+2) errors in message-digit errors only (M_5).

a.1.) decoded digit is not in error.

There are only 5 distinct rows of interfering digits, thus,

$$= 1\cdot 2\cdot 3\cdot 4\cdot 5 = \underline{\underline{120}}$$

a.2.) decoded digit in error.

This leaves 4 errors in other digits so that from section (M_4) a.1.

$$= \underline{\underline{274}}$$

b) 5 errors in 6 check-digits (C_5)

$$= \frac{6!}{5!\ 1!} = 6$$

c) (t+2) errors ($C_i\ M_{t+x-i}$)

c.1.) Combinations ($C_1\ M_4$)

c.1.1.) ($C_1\ M_4$, decoded digit not in error.)

$$= \frac{2!}{1!\ 1!} \cdot 274 = \underline{\underline{548}}$$

c.1.2.) ($C_1\ M_4$, decoded digit in error) from (M_4) a.2.

$$= \frac{3!}{1!\ 2!} \cdot 225 = \underline{\underline{775}}$$

c.2.) Combinations ($C_2\ M_3$)

c.2.1.) ($C_2\ M_3$, decoded digit not in error)

$$= \frac{3!}{2!\ 1!} \cdot 225 = \underline{\underline{775}}$$

c.2.2.) ($C_2\ M_3$, decoded digit in error) from ($C_1\ M_3$) c.1.2., there are 85 patterns of two errors in all interfering message-digit errors.

$$= \frac{4!}{2!\ 2!} \cdot 85 = \underline{\underline{510}}$$

c.3.) Combinations ($C_3\ M_2$)

c.3.1.) ($C_3\ M_2$, decoded digit not in error)

$$= \frac{4!}{3!\ 1!} \cdot 85 = \underline{\underline{340}}$$

c.3.2.) ($C_3\ M_2$, decoded digit in error)

$$= \frac{5!}{3!\ 2!} \cdot 15 = \underline{\underline{150}}$$

c.4.) Combinations ($C_4\ M_1$)

c.4.1.) ($C_4\ M_1$, decoded digit not in error)

$$= \frac{5!}{4!\ 1!} \cdot 15 = \underline{\underline{75}}$$

c.4.2.) ($C_4\ M_1$, decoded digit in error)

$$= \frac{6!}{4!\ 2!} = \underline{\underline{15}}$$

There are no other patterns of weight 5 possible within the n_e digits and

$$N'(e_i)_{t+2} = 120 + 274 + 6 + 548 + 775 + 775 + 510 + 340 + 150 + 75 + 15$$

$$= \underline{\underline{3588}}$$

From equation 12.3.2. if x = 2,

$$N(e_i)_{t+2} = N'(e_i)_{t+2} + \frac{(n_oN - n_e)!}{1!\ (n_oN-n_e-1)!} \cdot N'(e_i)_{t+1}$$

$$= 3588 + \frac{(72-22)!}{1!(72-22-1)!} \cdot 2459$$

$$= \underline{\underline{126,538}}$$

and therefore, since

$$N_5^{72} = \frac{72!}{5!\ 67!} = 13,991,544$$

then

$$p(5) = \frac{126,538}{13,991,544}$$

$$= \underline{\underline{90{\cdot}44\ 10^{-4}}}$$

It is obvious that when x > 1,

$$\sum_{j=1}^{x-1} \frac{(n_oN-n_{e_i})!}{j!\ (n_oN-n_{e_i}-j)!} \cdot N'(e_i)_{t+x-j} >> N'(e_i)_{t+x} \qquad 12.3.4.$$

and this is used to calculate estimates of the probability for $x > 4$.

The calculations were performed on the following codes;

a) $R = 1/2$, $t = 3$, $n_oN = 72$, 'cyclic'

b) $R = 1/2$, $t = 3$, $n_oN = 36$, C.S.O.C.

c) $R = 1/2$, $t = 4$, $n_oN = 72$, C.S.O.C.

d) $R = 2/3$, $t = 4$, $n_oN = 384$, 'cyclic'

e) $R = 2/3$, $t = 4$, $n_oN = 261$, C.S.O.C.

f) $R = 2/3$, $t = 5$, $n_oN = 393$, C.S.O.C.

The results are given in Table 12.3.1.

TABLE 12.3.1.

Probability of a decoding failure when t+x errors occur.

Rate = 1/2.

	'Cyclic'	C.S.O.C.	C.S.O.C.
t+x	$t = 3$, $n_oN = 72$	$t = 3$, $n_oN = 36$	$t = 4$, $n_oN = 72$
4	$2{\cdot}39{\cdot}10^{-3}$	$41{\cdot}7{\cdot}10^{-3}$	0
5	$9{\cdot}10^{-3}$	$100{\cdot}8{\cdot}10^{-3}$	$6{\cdot}4{\cdot}10^{-3}$
6	$20{\cdot}6{\cdot}10^{-3}$	$159{\cdot}10^{-3}$	$21{\cdot}3{\cdot}10^{-3}$
7	$36{\cdot}9{\cdot}10^{-3}$	$210{\cdot}7{\cdot}10^{-3}$	$42{\cdot}9{\cdot}10^{-3}$
8	$56{\cdot}9{\cdot}10^{-3}$	$249{\cdot}5{\cdot}10^{-3}$	$68{\cdot}7{\cdot}10^{-3}$
9	$79{\cdot}6{\cdot}10^{-3}$	$265{\cdot}10^{-3}$	$95{\cdot}7{\cdot}10^{-3}$

Rate = 2/3

	'Cyclic'	C.S.O.C.	C.S.O.C.
	$t = 4$, $n_oN = 384$	$t = 4$, $n_oN = 261$	$t = 5$, $n_oN = 393$
5	$2{\cdot}10^{-5}$	$19{\cdot}3{\cdot}10^{-5}$	0
6	$10{\cdot}2{\cdot}10^{-5}$	$88{\cdot}3{\cdot}10^{-5}$	$2{\cdot}43{\cdot}10^{-5}$
7	$30{\cdot}10^{-5}$	$237{\cdot}9{\cdot}10^{-5}$	$12{\cdot}9{\cdot}10^{-5}$
8	$67{\cdot}8{\cdot}10^{-5}$	$491{\cdot}8{\cdot}10^{-5}$	$39{\cdot}5{\cdot}10^{-5}$
9	$131{\cdot}6{\cdot}10^{-5}$	$863{\cdot}10^{-5}$	$90{\cdot}2{\cdot}10^{-5}$

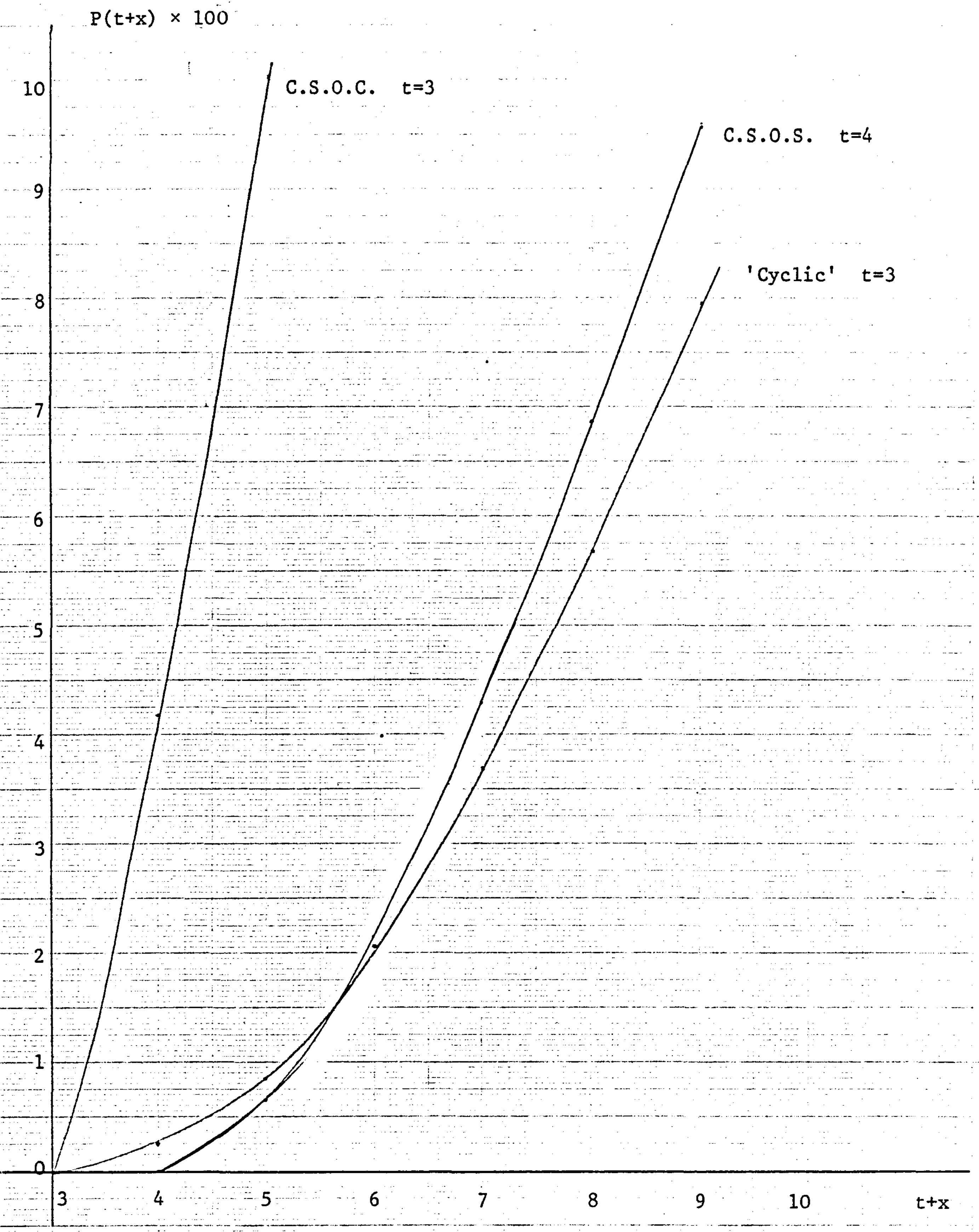


FIG. 12.3.1.

COMPARISON OF THE UNBOUNDED ERROR CORRECTION CAPABILITY OF THE CODES:

'CYCLIC' $R = 1/2$, $n_o N = 72$, $t = 3$.

C.S.O.C. $R = 1/2$, $n_o N = 72$, $t = 4$.

C.S.O.C. $R = 1/2$, $n_o N = 36$, $t = 3$.

The rate half results are drawn in graph form in figure 12.3.1.

For the 'cyclic' rate 2/3 code, the figures given are for the digits in the coset containing digit 1. The C.S.O.C. rate 2/3 figures are on digits 1 and 2 of a sub-block, for the $t = 5$ and $t = 4$ codes respectively.

12.4 Equal Rate and Error-correcting Capability.

In section 12.3. we examined how $p_{x_i.A}(t+x)$ changed with increasing $(t+x)$. We also saw in example 12.3.1. that the equivalent form of the check sums on some digit was sufficient to calculate the quantities $N'(e)_{t+x}$ and therefore the probabilities too.

However what was not shown is that the equivalent forms for the digits in the 'cyclic' $t = 3$ code and the C.S.O.C. $t = 3$ code are identical, so that,

$$N'(e)_{t+1} = 2459$$

for both codes. Of course all $N'(e)_{t+x}$ are equal in value for these two codes, so that the differing probabilities in Table 12.3.1. arise purely due to the different constraint lengths of the two codes. Since the C.S.O.C. has, $n_oN = 36$, and the 'cyclic', $n_oN = 72$, we could ask, is the increase in constraint length n_oN, justified in terms of the reduction in the probability of a decoding failure? Also, is there some optimum constraint length n_oN beyond which the reduction in probability is not justified? Both questions are answered by considering a "flexible" code whose rate, error pattern weight $t+x$, error-correcting capability t, check sum equivalent form and therefore effective constraint length, are held constant while n_oN is increased.

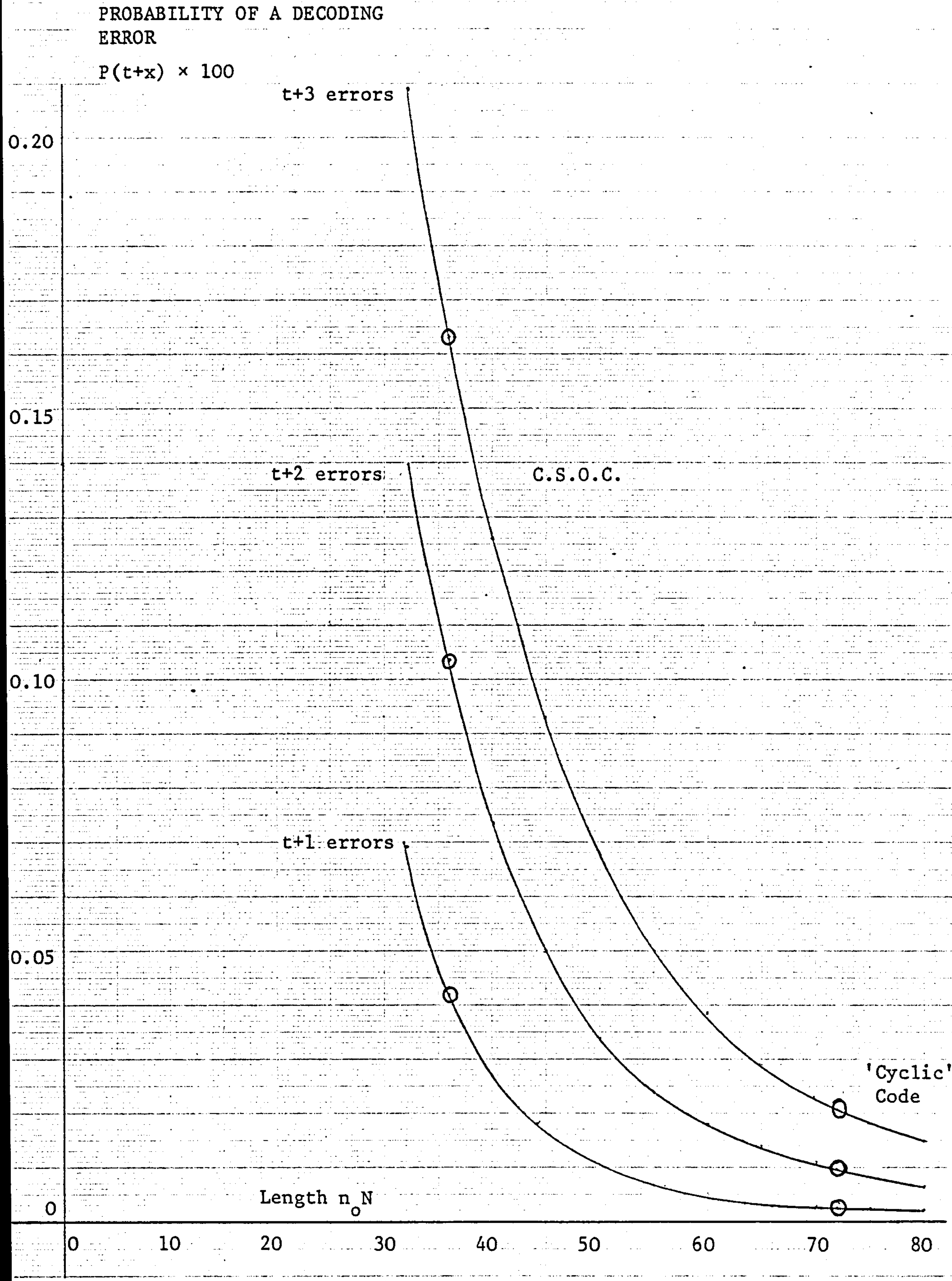
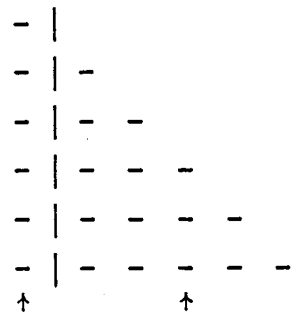


FIG. 12.4.1.

COMPARISON OF THE C.S.O.C. $n_oN = 36$ AND 'CYCLIC' $n_oN = 72$ CODES FOR EQUAL $R = 1/2$ AND $t = 3$.

In figure 12.4.1. we show three curves of probability against n_oN, for (t+1), (t+2) and (t+3) errors, for the digits of a code whose check sum equivalent form is given by;



The t = 3, C.S.O.C. and t = 3 'cyclic' codes can be compared by locating them on each curve, using the values from Table 12.3.1.

The curves were plotted using the equations below;

$$p(t+1) = \frac{2459 \cdot 4!\ (n_oN-4)!}{n_oN!}$$

$$p(t+2) = \frac{N(e)_{t+2} \cdot 5!\ (n_o-5)!}{n_oN!}$$

$$p(t+3) = \frac{N(e)_{t+3} \cdot 6!\ (n_oN-6)!}{n_oN!}$$

where $N(e)_{t+x}$ is given by equation 12.3.2. In figure 12.4.2. we also plot probabilities for the rate 2/3 C.S.O.C. t = 4, and rate 2/3, 'cyclic' t = 4 codes. In this case the check sum equivalent forms are not the same for both codes so that two curves are shown for each (t+x).

That is for the 'cyclic' code,

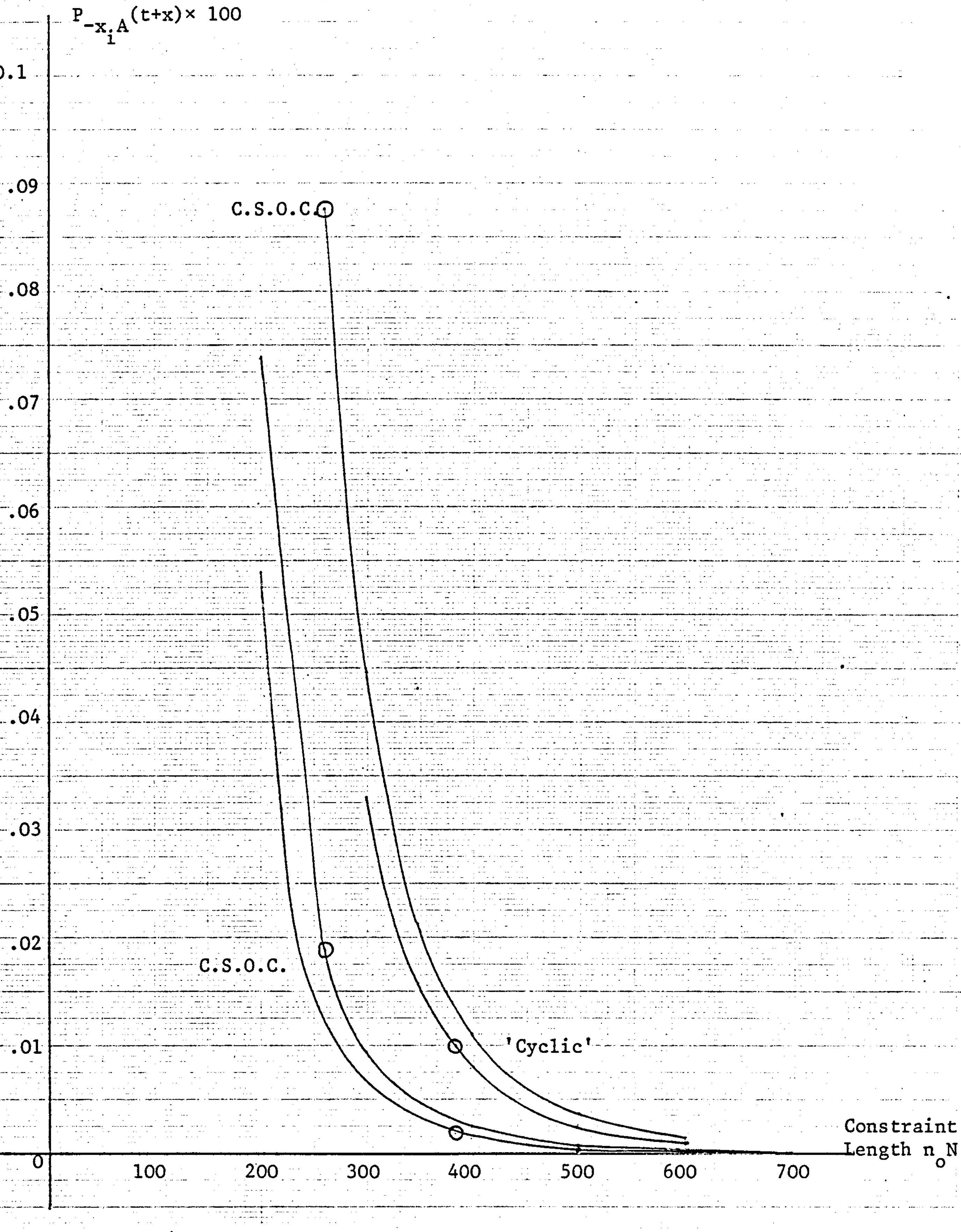


FIG. 12.4.2.

COMPARISON OF THE C.S.O.C. $n_o N = 261$ AND 'CYCLIC' $n_o N = 384$ CODES FOR EQUAL $R = 2/3$ AND $t = 4$.

$$p(t+1) = \frac{1,378,161 \cdot 5!\ (n_o N-5)!}{n_o N!}$$

but for the C.S.O.C.,

$$p(t+1) = \frac{1,869,925 \cdot 5!\ (n_o N-5)!}{n_o N!}$$

12.5 Error Propagation.

12.5.1. Introduction.

Error propagation occurs only in convolutional codes which are decoded in the feedback mode. It can occur in two ways,

(a) an error in some message-digit is decoded as binary zero, so that using feedback syndrome cancellation, (F.S.C.) does not eliminate its effects from the check-digits of following blocks.

(b) no error in some message-digit is decoded as binary one, so that F.S.C. adds an error in the check-digits of the following blocks where the message-digit error resided.

This can have the effect of causing a further decoding error in a subsequent block which would normally have been correctly decoded. That is some digit from a subsequent block may have t check sums in error, from t errors over its constraint length $n_o N$, but a (t+1)'th check sum can be put in error by a propagated error from a previous block.

It has been shown[7,12] that if a convolutional code has the property that F.S.C. reduces the binary weight of the $N(n_o-k_o)$-digit syndrome then the code can recover from error propagation automatically providing a sufficiently long error free period occurs. It was also shown that self-orthogonal codes have this property, so that the 'cyclic' codes have this property too. Having stated that convolutional self-orthogonal codes, C.S.O.C.'s, can recover automatically, we may ask, how soon can a code recover. Having said this we must first dicuss what is meant by recover. If we assume the decoding failure occurred due to an error pattern of weight t+1, then it is possible that this error pattern plus the propagated error, will cause a further decoding failure in one of the (N-1) blocks following the block in which the original decoding failure occurred. If this situation occurs then the code has not recovered from the original propagated error and we would then concern ourselves with how soon it could recover from the second propagated error. For this reason we make our first assumption, that is, we assume the N-1 blocks following a decoding failure are correctly decodable in the presence of the error pattern and propagated error. This is referred to as the autonomous case. We then require to know how soon the code can be subjected to t or less errors again, without upsetting the autonomous case. In other words without a further decoding failure occurring due to the propagated error, which must be in one of the N-1 blocks following the original decoding failure.

If we can develop conditions which guarantee the autonomous case we have developed conditions under which the code is guaranteed to recover from error propagation. This is because, since the code has constraint length N blocks long a propagated error cannot affect the decoding of the (N+1)'th block. Therefore if the N-1 blocks following the decoding failure can be successfully decoded the N'th block, (N+1)'th block overall, and all subsequent blocks must be free from the propagated error.

The number of digits the code requires to recover from a propagated error is called the propagation length and is given the symbol, L.

A more precise definition of L assuming the autonomous case is given below.

<u>Definition 12.5.1.1.</u>

The propagation length, L, is a multiple of the sub-block length n_o, and is defined as the least number of bits, including the last bit incorrectly decoded, which must follow a decoding error and beyond which the code can be subjected to t or less errors again without further decoding errors occurring due to the propagated error.

Robinson[12] developed an upper bound for L as,

$$L \leqslant 2n_oN - n_o \qquad 12.5.1.1.$$

and to enable comparison of L for different codes we introduce the concept of propagation efficiency,

$$E = 1 - \frac{L}{2n_oN - n_o} \quad . \qquad 12.5.1.2.$$

The smaller L, the more efficient is the code.

12.5.2. Propagation in the 'cyclic' codes.

The definition 12.5.1. assumes a single propagated error and this will be assumed throughout unless stated otherwise. From the discussion in the introduction to this section it is apparent that the actual value of L is not too easy to calculate and in fact it will be many different values depending upon the different error patterns which caused the propagated error. In the following results it is assumed that a propagated error has occurred due to an error pattern of weight t+1 and we consider rate R = 1/2 codes only. After F.S.C. we develop conditions which indicate a maximum value for L and we proceed in the following manner.

(a) We assume a particular digit has been decoded in error causing propagation. Due to the cyclic nature of the code, any conditions developed on one digit will also hold for other digits of the same sub-block. The sub-block from which the error is propagated is considered as block 1.

(b) We then assume the N-2 blocks following block 1 are successfully decoded and the N'th block is about to be decoded in the presence of the propagated error.

(c) Only one digit in block N can be interfered with by the propagated error, so that provided the next t+1 check sums on that digit are free from error (this assumes the worst case, that is that the digit in block N, interfered by the propagated error, is in error itself) it can be successfully decoded. This sets an initial value for L at $(N+t+1)n_o$, since this assumes no further errors can occur in the t+1 blocks that follow block N.

Having set this initial value for L, any blocks whose 2t check sums are within the $(N+t+1)n_o$ digits will be decoded successfully, assuming the conditions for the autonomous case. That is, successful decoding of block x is assured, providing

$$n_o(x+n-1) \leqslant n_o(N+t+1)$$

$$x \leqslant t+2. \qquad 12.5.2.1.$$

THE 66-BIT SYNDROME FOR THE nN=72, t=3, R=1/2 CODE.

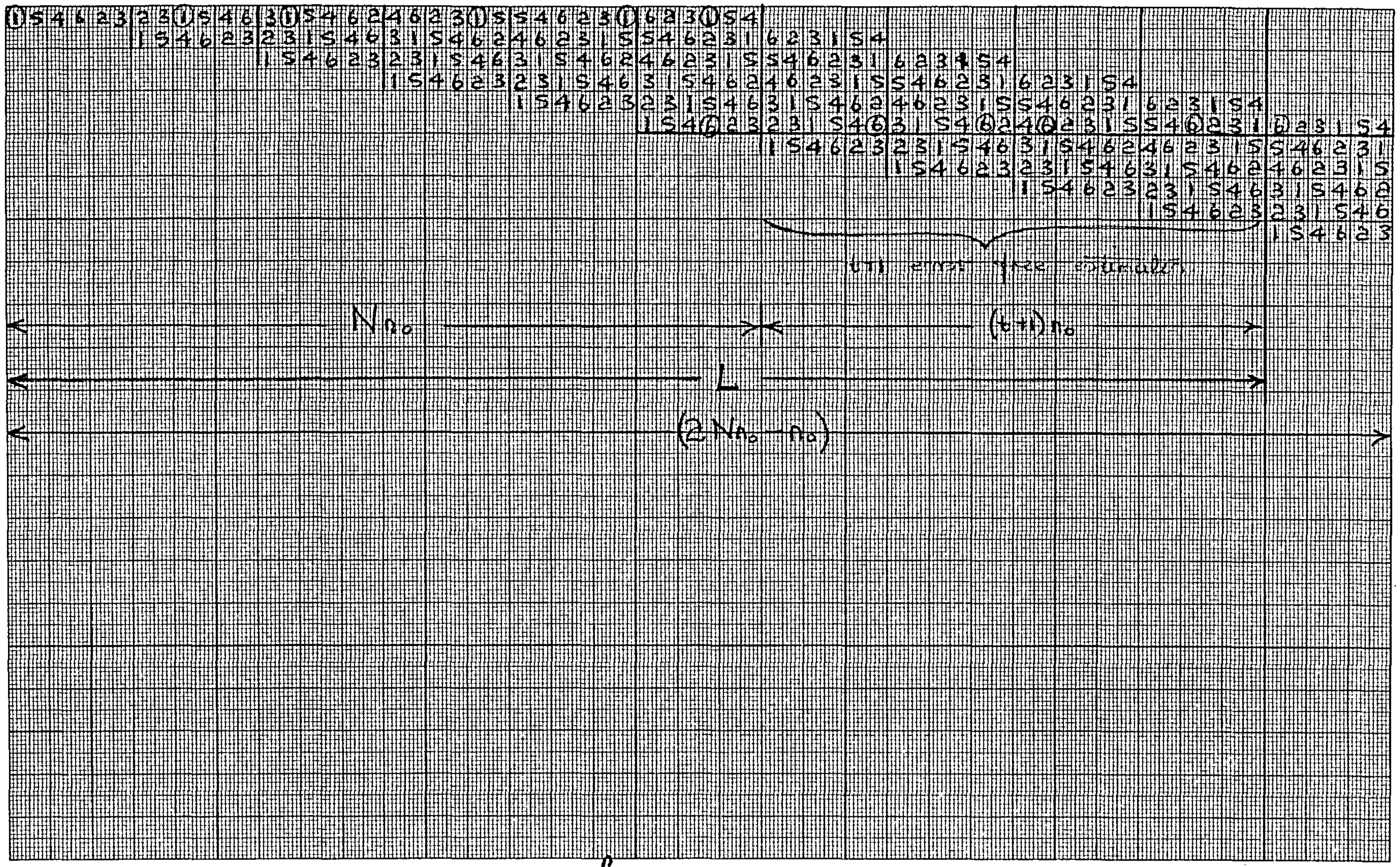


fig. 12.5.2.1.

(d) We then move backwards and consider the decoding of block N-1, only if $N-1 > t+2$. We try to determine conditions which would require that L be greater than $(N+t+1)n_o$ in order that all digits from block N-1 can be decoded successfully under our assumptions.

(e) We repeat (d) until block $y \leqslant t+2$, when we stop. The value of L is the maximum obtained for all blocks $y > t+2$, but $y \leqslant N$.

As an example consider figure 12.5.2.1., which shows the $2Nn_o - n_o$ syndrome digits of the $n_oN = 72$, $t = 3$, $R = 1/2$ code, assuming an error is propagated by digit 1 from some block 1.

From (c) our initial value for L is $(N+t+1)n_o = (6+3+1)\cdot 12 = 120$. However $N-1 = 5 = t+2$ so that from (e) we can stop and this initial value of L is the final value.

12.5.3 Propagation in the C.S.O.C.'s[7].

The search procedure applied to the 'cyclic' codes was also applied to the C.S.O.C.'s[7], with rate, $R = 1/2$.

The results can be shown numerically and a specimen is given for the $n_oN = 172$, $t = 6$, $R = 1/2$ code, below.

<u>Example 12.5.3.1.</u>

The generator sequence for the $n_oN = 172$, $t = 6$, $R = 1/2$, C.S.O.C. is[17];

(1, 3, 7, 25, 30, 41, 44, 56, 69, 76, 77, 86).

We assume that error propagation has occurred for some block 1 and was in part due to errors in the message-digits of blocks 74, 77 and 86. These errors affected check sums 76, 77 and 86 of block 1. However when we wish to decode block 74, it has its check sums in blocks;

(74, 76, 80, 98, 103, 114, 117, 129, 142, 149, 150, 159)

The check sum in block 76 is in error from the propagated error, but if one examines those blocks containing the errors in blocks 77 and 86 we have;

(77, 79, 83, 101, 106, 117, 120, 132, 145, 152, 153, 162)

(86, 88, 92, 110, 115, 126, 129, 141, 154, 161, 162, 171)

We can see that although F.S.C. will cancel these errors in blocks 77 and 86, they will be present in blocks 117 and 129 which both contain check sums for the message-digit in block 74. Therefore of those twelve check sums 74, 117 and 129 are in error, from propagation and existing errors. To decode this digit successfully we require at least 7 error-free check sums, since 74 is in error itself. This required then that no further errors must occur in any block $y \leqslant 149$, giving

$$L = n_o \cdot 149 = 298.$$

Table 12.5.1. compared the figures calculated for the 'cyclic' and C.S.O.C. codes.

The values given are the maximum that could be found. The table also compares the propagation efficiency of the two codes.

TABLE 12.5.1.

Error propagation length L, in the rate 1/2 C.S.O.C's and 'cyclic' C.S.O.C's.

t	n_oN	$2n_oN-n_o$	L 'cyclic'	L C.S.O.C.	E
3	72	132	120		0.091
4	72	142		132	0.0705
5	200	380	320		0.1579
6	172	342		298	0.128
6	288	552	456		0.174
7	256	510		452	0.1138
8	512	992	800		0.1936
9	434	866		702	0.1894
9	648	1260	1008		0.2
10	568	1134		932	0.1782
11	968	1892	1496		0.2094
12	852	1702		1358	0.2022

CHAPTER 13

Chapter 13 is a reprint of an article, "Pseudostep Orthogonalization: An Algorithm for improving Reed-Massey threshold codes" published in Electronics Letters 8th June 1978, Vol. 14., No.12. pp.355-357.

readily be added by means of a second full-wave rectifier biased to operate at the voltage corresponding to the outer significant condition.

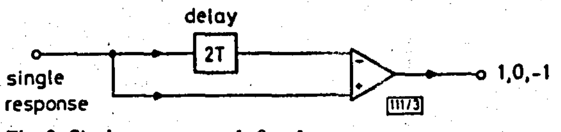


Fig. 3 *Single response to 1, 0, −1 processor*

The block diagram of the complete timing recovery circuit is shown in Fig. 1 and a stylised illustration of the waveforms at several points in the circuit when it is presented with an undistorted (1, 0, −1) eye pattern is given in Fig. 2. The timing signal component present in a 1 V peak-to-peak processed signal (Fig. 2*d*) was measured to be 0·06 V r.m.s. at optimum demodulating carrier phase. It was only about 4 dB less when transmitting via a channel with such severe slope and sag group-delay distortion that the (1, 0, −1) eye pattern was barely recognisable. The level of the timing signal component varied less than 5 dB with demodulating carrier phase under any of the conditions tested. With pseudorandom data modulation at 60 kbaud and no added noise, the peak-to-peak jitter at the output of the second-order phase-locked loop, which had a double-sided bandwidth of 100 Hz, varied between 2% and 4% under the same test conditions. No complete theoretical explanation or analysis of the operation of the circuit can be given at present, but it is reasonable to deduce that the strong timing-signal component obtained arises from the constraints imposed by the (1, 0, −1) processing on the occurrence and polarity of the signal transitions. As a corollary to this, it may be inferred that the phase of the recovered timing signal is determined mainly by the predominant (1, 0, −1) signal components, i.e. those at half the Nyquist frequency.

The circuit is being used successfully in an experimental modem for transmission over group-band channels at rates in the range 48-72 kbit/s according to CCITT Recommendation V36. It has also been established that the same principle can be applied to extract a timing signal with the same valuable properties from the corresponding segments of a multilevel (1, 0, −1) signal.

Finally it is worth noting that the circuit can be used for timing-signal recovery in single-response data modems by driving it from the bandlimited received baseband signal via a linear (1, 0, −1) processor such as that shown in Fig. 3 (T is the data unit interval). If a sample-and-hold delay element is used for this purpose, additional band limiting will be required between the processor and the timing recovery circuit.

Acknowledgment: The permission of the Director of Research of the British Post Office to publish this letter is acknowledged.

P. N. RIDOUT *4th May 1978*

Post Office Research Department
Martlesham Heath
Ipswich, IP5 7RE
England

References

1 RIDOUT, P. N., and RIDOUT, I. B.: 'Adaptive multiple response reception and waveform correction for data transmission via channels in the fdm network', paper presented at IEEE International seminar on digital communications, Zurich, 12-15 March 1974
2 KRETZMER, E. R.: 'Generalization of a technique for binary data communication', *IEEE Trans.*, 1966, COM-14, pp. 67-68

0013-5194/78/1111-0354 $1.50/0

PSEUDOSTEP ORTHOGONALISATION: AN ALGORITHM FOR IMPROVING REED-MASSEY THRESHOLD CODES

Indexing term: Error correction codes

An algorithm is presented which can be applied to Reed-Massey algorithm codes and utilises orthogonal and nonorthogonal check sums with a resulting improved performance. The algorithm is applied to a well-known class of convolutional threshold codes with a subsequent improvement in the nonbounded error-correcting capability.

Introduction: In 1954, Reed[1] first proposed a threshold decoding scheme for a class of codes developed by Muller, but it was not until 1963 that a unified theory emerged, developed by Massey.[2] The Reed-Massey algorithm basically proposed obtaining $J = (d - 1)$ orthogonal check sums, where d is the minimum distance of the codes. Each check sum is assigned equal priority in the decoding scheme and providing $t \leqslant J/2$ errors occur, they can be corrected. Many useful classes of codes have been based upon this algorithm.

In recent years attention has been focused on Rudolph's algorithm[3] which proposes obtaining $2t\lambda$ nonorthogonal check sums, where λ is the number of check sums in which each digit, except the decoded digit, appears. Ng[4] later showed that this algorithm could be improved by assigning the zero parity check more than one vote. In a more recent correspondence Duc[5] proposed a new algorithm, called pseudostep orthogonalisation, where the decoder utilises a combination of orthogonal and nonorthogonal check sums.

The main application of the above algorithms[3-5] has been the decoding of codes previously known not to be decodable with the Reed-Massey algorithm. The pseudostep algorithm presented in the next section can also be applied to Reed-Massey algorithm codes, with a resulting improvement in performance.

The algorithm: The algorithm combines the ideas of Reed-Massey's algorithm, Duc's algorithm and Ng and can be outlined in the following manner:

(*a*) S check sums are obtained

(*b*) $(S - 2)$ check sums are orthogonal and are assigned 3 votes each

(*c*) 2 check sums are nonorthogonal and are assigned 2 votes each

(*d*) the zero parity check is assigned 3 votes

The total vote becomes

$$S_T = 3(S - 2) + 4 + 3 = 3S + 1$$

Let J be the number of orthogonal check sums of a code which is threshold decodable using the Reed-Massey algorithm.

(i) If d is odd, $J = (d - 1) = 2t$, and we set $S = J + 1$, giving $S_T = 6t + 4$.

(ii) If d is even, $J = (d - 1) = 2t + 1$, and we set $S = J + 1$, giving $S_T = 6t + 7$.

That is, we find an extra check sum nonorthogonal on one of the original orthogonal check sums.

When $S_T = 6t + 4$ we decode all error patterns of weight t plus many patterns of weight $(t + 1)$ which would have caused a decoding failure with $J = 2t$.

When $S_T = 6t + 7$ we decode all error patterns of weight t and all patterns of weight $(t + 1)$ except those which include the digit common to the nonorthogonal check sums. However, the majority of these patterns are also correctly decoded.

Application of the algorithm: The algorithm has been applied to Massey's trial and error (t.e.c.) convolutional codes[2] and, in addition, a small set of rate $\frac{1}{4}$ codes were developed. The resulting constructions are given in Table 1, where the following points apply:

(*a*) The notation for generator sequences and check sum rules is the same as used by Massey.[2]

(*b*) While developing the codes some Reed-Massey algorithm codes were found which are shorter than Massey's t.e.c.s and are given for completeness in Table 2.

(*c*) Sub-block length = *n*, constraint length = *N*, rate = 1/*n*.

Comments: The code, $n = 4$, $S = 10$, $N = 5$, was examined to discover the number of patterns of errors of weight 5 which caused a decoding failure. Of the 15,504 error patterns of weight five, 3,876 include the common digit in the non-

Table 1 IMPROVED MASSEY T.E.C. CODES

n	*S*	*N*	Generator sequences	Rules for forming check sums
3	5	3	$(0, 1, 2)^1$ $(0, 1)^2$	$(0^1)(0^2)(1^1)*(1^2)*(2^1 2^2)$
3	7	5	$(0, 1, 2, 3)^1$ $(0, 1, 3)^2$	$(0^1)(0^2)(1^1)*(1^2)*(2^1 2^2)$ $(3^2)(3^1 4^1)$
3	9	8	$(0, 1, 2, 3, 6, 7)^1$ $(0, 2, 3)^2$	$(0^1)(0^2)(1^1)*(2^2)(1^2 3^2)$ $(3^1 4^1)(6^1 6^2)(7^1 7^2)*$ $(2^1 4^2 5^2 5^1)$
3	11	12	$(0, 1, 9, 11)^1$ $(0, 1, 2, 3, 5, 8, 9)^2$	$(0^1)(0^2)(1^1)(2^1 2^2)(9^1)$ $(3^2 4^2)(3^1 5^1 5^2)(8^1 8^2)(11^1)*$ $(1^2 4^1 6^1 6^2)(7^2 9^2 10^2)*$
3	13	18	$(0, 1, 15, 17)^1$ $(0, 4, 5, 6, 7, 9, 12, 13, 16)^2$	$(0^1)(0^2)(1^1 1^2)(4^2)(5^2)(15^1)$ $(2^1 6^2)(7^2 10^2 11^2 11^1)(9^2 3^1 5^1)$ $(4^1 10^1 12^1 16^2)*(6^1 8^1 12^2)(17^1)*$ $(13^2 7^1 9^1 8^2 2^2 3^2)$
4	5	2	$(0, 1)^1$ $(0, 1)^2$ $(0)^3$	$(0^1)(0^2)(0^3)(1^1)*(1^2)*$
4	7	3	$(0, 1, 2)^1$ $(0, 1)^2$ $(0, 2)^3$	$(0^1)(0^2)(0^3)(1^1)*(1^2)*$ $(2^1 2^2)(2^3)$
4	10	5	$(0, 1, 2, 3)^1$ $(0, 1, 4)^2$ $(0, 2, 4)^3$	$(0^1)(0^2)(0^3)(1^1)*(1^2 1^3)(2^1 2^2)$ $(2^3)(3^1 3^2)*(4^2)$ $(4^3 3^3 4^1)$
4	11	6	$(0, 1, 2, 3)^1$ $(0, 1, 4, 5)^2$ $(0, 2, 5)^3$	$(0^1)(0^2)(0^3)(1^1)(1^2 1^3)(2^1 2^2)$ $(2^3)*(3^1 4^1)(5^1)(4^2 3^2 4^3)$ $(5^1 5^2 3^4)*$
4	13	8	$(0, 1, 2, 3, 7)^1$ $(0, 1, 4, 5)^2$ $(0, 2, 5, 6, 7)^3$	$(0^1)(0^2)(0^3)(1^1)(1^2 1^3)(2^1 2^2)$ $(2^3)(3^1 4^1)(2^3 2^4 3^4)(5^3)$ $(5^2 6^1 3^3)*(6^2 7^3)*$ $(4^1 7^1 7^2)$
4	15	11	$(0, 1, 2, 3, 7, 9)^1$ $(0, 1, 4, 5, 9, 10)^2$ $(0, 2, 5, 7, 8)^3$	$(0^1)(0^2)(0^3)(1^1)(1^2 1^3)(2^1 2^2)$ $(2^3)(3^1 4^1)(3^2 4^2 4^3)(5^3)*$ $(7^3 6^2 8^1)*(5^2 3^2 6^1)$ $(8^2 6^3 8^3)(5^1 7^1 7^2)$ $(9^1 9^2 9^3 10^2)$
4	14	9	$(0, 1, 2, 3, 7)^1$ $(0, 1, 4, 5, 8)^2$ $(0, 2, 5, 7)^3$	$(0^1)(0^2)(0^3)(1^1)(2^1 2^3)(2^3)$ $(3^1 4^1)(3^2 4^2 4^3)(5^3)*$ $(7^3 6^2 8^1)*(7^1 5^1 7^2)(1^2 1^3)$ $(5^2 3^3 6^1)(8^2 6^3 8^3)$
5	7	2	$(0, 1)^1$ $(0, 1)^2$ $(0, 1)^3$ $(0)^4$	$(0^1)(0^2)(0^3)(0^4)(1^1)*(1^2)*$ $(1^3 1^4)$
5	9	3	$(0, 1, 2)^1$ $(0, 1)^2$ $(0, 1)^3$ $(0, 2)^4$	$(0^1)(0^2)(0^3)(0^4)(1^1)*(1^2)*$ $(1^3 1^4)(2^1 2^2)(2^4)$
5	11	4	$(0, 1, 2, 3)^1$ $(0, 1, 3)^2$ $(0, 1)^3$ $(0, 2)^4$	$(0^1)(0^2)(0^3)(0^4)(1^1)*(1^2)*(1^3 1^4)$ $(2^4)(2^1 2^2)(3^1 2^3)(3^2 3^3)$

Table 1 IMPROVED MASSEY T.E.C. CODES–*continued*

n	S	N	Generator sequences	Rules for forming check sums
5	13	6	$(0, 1, 2)^1$ $(0, 1, 3)^2$ $(0, 3, 5)^3$ $(0, 2, 3, 4)^4$	$(0^1)(0^2)(0^3)(0^4)(4^3 4^4)$ * $(3^1 3^2)(3^3)(4^2 3^4)(2^1 2^2)$ $(2^3 2^4)(1^1 1^3)(1^2 1^4)(5^3)$ *
5	15	7	$(0, 1, 2)^1$ $(0, 1, 3)^2$ $(0, 3, 5, 6)^3$ $(0, 2, 3, 4, 6)^4$	$(0^1)(0^2)(0^3)(0^4)(1^1 1^3)(1^2 1^4)$ $(2^1 2^2)(2^3 2^4)(3^1 3^2)(4^2 3^4)$ $(3^3)(4^3 4^4)$ * (5^3) * $(5^2 6^2 6^4)(4^1 5^1 6^3 5^4)$
5	17	9	$(0, 1, 2, 4, 5, 7)^1$ $(0, 1, 3, 8)^2$ $(0, 3)^3$ $(0, 2, 3, 4, 7)^4$	$(0^1)(0^2)(0^3)(0^4)(4^3 4^4)(3^1 3^2)$ $(3^3)(4^2 3^4)(2^1 2^2)(2^3 2^4)(1^3 1^1)$ $(1^2 1^4)(4^1 6^4)(5^1 5^4 5^2)$ $(7^1 6^3 5^3)$ * $(7^4 6^2 7^2)$ $(8^2 8^3)$ *
5	19	11	$(0, 1, 2, 4, 4, 5, 7)^1$ $(0, 1, 3, 9)^2$ $(0, 3, 10)^3$ $(0, 2, 3, 4, 7, 8, 9)^4$	$(0^1)(0^2)(0^3)(0^4)(4^3 4^4)(3^1 3^2)$ $(3^3)(4^2 3^4)(2^1 2^2)(2^3 2^4)(1^1 1^3)$ $(1^2 1^4)(4^1 6^4)(5^1 5^4 5^2)$ $(7^1 6^2 5^3)$ * $(7^4 6^2 7^2)(9^2 9^3)$ $(9^4 7^3 6^1)(10^3)$ *

The asterisks in the above table indicate the nonorthogonal check sums

Table 2 SHORTENED REED-MASSEY ALGORITHM CODES

n	t	N	Generator sequences	Rules for forming check sums
4	4	4	$(0, 1, 2, 3)^1$ $(0, 1)^2$ $(0, 2)^3$	$(0^1)(0^2)(0^3)(1^1)(1^2 1^3)$ $(2^1 2^2)(2^3)(3^1 3^2 3^3)$
5	6	5	$(0, 1, 2)^1$ $(0, 1, 3)^2$ $(0, 3)^3$ $(0, 2, 3, 4)^4$	$(0^1)(0^2)(0^3)(0^4)(4^4 4^3)$ $(3^1 3^2)(3^3)(4^2 3^4)(2^1 2^2)$ $(2^3 2^4)(1^1 1^3)(1^2 1^4)$
5	8	8	$(0, 1, 2, 4, 5, 7)^1$ $(0, 1, 3)^2$ $(0, 3)^3$ $(0, 2, 3, 4, 7)^4$	above rules plus $(4^1 6^4)(5^1 5^2 5^4)$ $(7^1 6^3 5^3)(7^4 6^2 7^2)$
5	9	10	$(0, 1, 2, 4, 5, 7)^1$ $(0, 1, 3, 9)^2$ $(0, 3)^3$ $(0, 2, 3, 4, 7, 8, 9)^4$	above rules plus $(9^4 7^3 6^1)(9^2 9^3)$
3	4	7	$(0, 1, 2, 3, 6)^1$ $(0, 2, 3)^2$	$(0^1)(0^2)(1^1)(2^2)(1^2 3^2)$ $(6^1 6^2)(3^1 4^1)(2^1 4^2 5^1 5^2)$

orthogonal check sums. Of these 3,876, the code fails to decode in 1,083 cases.

Conclusions: A pseudostep orthogonalisation algorithm has been presented which can be applied to Reed-Massey algorithm codes to improve their performance.

The algorithm was applied to Massey's t.e.c. convolutional codes resulting in an improved performance, with no increase in length in most cases.

One criticism of t.e.c. convolutional codes which are decoded in the feedback mode is that they do not have the automatic recovery properties of self-orthogonal codes in the presence of error propagation.[6] However, the improved t.e.c. codes presented, having a better unbounded error correction capability, are less likely to have errors propagated than the standard t.e.c. codes.

The advantage of the t.e.c. codes is that, for a given error-correcting capability, they are much shorter than self-orthogonal convolutional codes.[2, 7]

D. McQUILTON
M. E. WOODWARD

8th May 1978

*Department of Electronic and Electrical Engineering
University of Technology
Loughborough, Leics. LE11 3TU
England*

References

1 REED, I. S.: 'A class of multiple-error correcting codes and the decoding scheme', *IRE Trans.*, 1954, IT-4, pp. 38–49
2 MASSEY, J. L.: 'Threshold decoding' (MIT Press, 1963)
3 RUDOLPH, L. D.: 'A class of majority-logic decodable codes', *IEEE Trans.*, 1967, IT-13, pp. 305–307
4 NG, S. N.: 'On Rudolph's majority-logic decoding algorithm', *ibid.*, 1970, IT-16, pp. 651–652
5 DUC, N. Q.: 'Pseudostep orthogonalization: A new threshold-decoding algorithm', *ibid.*, 1971, IT-17, pp. 766–768
6 ROBINSON, J. P., and BERNSTEIN, A. J.: 'A class of binary recurrent codes with limited error propagation', *ibid.*, 1967, IT-13, pp. 106–113
7 REDDY, S. M., and ROBINSON, J. P.: 'Hybrid block-self-orthogonal convolutional codes', *ibid.*, 1972, IT-18, pp. 185–191

0013-5194/78/1116-0355 $1.50/0

14. CONCLUSIONS AND COMMENTS.

14.1 Comparison with other C.S.O.C.'s.

The usefulness and application of convolutional self-orthogonal codes has been shown by Wu[22] who listed their advantages as:

a) simple implementation.

b) freedom from error propagation.

c) guaranteed correction capability beyond the minimum distance, d_m.

d) capability of operating at very high speed.

e) a large number of codes.

In a later paper Wu's[24] results showed that C.S.O.C.'s exhibited the further property that they did not produce additional and bursty errors at the decoder output when the code's capability was exceeded. This he concluded makes these codes superior to Reed-Solomon codes, B.C.H. and Viterbi decoders when used as the inner code of a concatenated code. Also, one of the examples Wu[22] gave for the DITEC digital television system, a C.S.O.C. was chosen in preference to B.C.H. codes, difference-set cyclic codes, Viterbi and sequential decoding after an extensive evaluation.

However Wu's[22] statement that one of the C.S.O.C.'s advantages is that there are a large number of codes throws light on the state of the art at that time since combining all C.S.O.C.'s from Robinson[7] et al., Klieber[14] and Wu[22,23] there were only 143 codes with rate greater than one half. Also the C.S.O.C.'s did not exist at certain rates and, as Wu[23] stated, tend to be single error-correcting for very high rates. These results are in sharp constrast to the class of 'cyclic' C.S.O.C.'s presented in Chapter 11 where, if we consider only primes p, then if there exists some integer x such that $x|(p-1)$ there exists a code with;

$$J = \frac{(p-1)}{x} \quad ; \qquad nN = (x)(x+1)(J)^2$$

$$\text{Rate} = \frac{x}{x+1} .$$

Since the number of primes is infinite, for ANY x, there are theoretically an infinite number of codes of error-correcting capability greater than one. Nevertheless where possible the 'cyclic' codes have been compared with the C.S.O.C.'s and these results are presented in Table 14.1. For a given rate and J we compare actual constraint lengths n_A. The actual constraint length n_A is important from the point of view of decoder complexity and economics, the larger n_A the more expensive the decoder. As can be seen from the Table 14.1. the 'cyclic' codes tend to be shorter for rate $\geqslant 5/6$ and in fact for all codes of rate $\geqslant 17/18$ there are no C.S.O.C.'s shorter than the 'cyclic' codes.

Wu[23] also mentioned that a computer was needed to specify the C.S.O.C.'s at high rates. However in example 11.3.24 it was shown, with a moderate amount of hand calculation, how to construct two codes; (a) $d_m = 5$, Rate = 601/602.

(b) $d_m = 3$, Rate = 1202/1203

both having extremely high rates. However within the range of values that the C.S.O.C.'s and 'cyclic' C.S.O.C.'s can be compared there are many points (values of J and Rate) where the C.S.O.C.'s exist and the 'cyclic' C.S.O.C.'s don't, and vice-versa, so that within this range the two classes of codes can be considered as complementing each other.

We can consider decoder complexity by considering two codes of the same J, n_A and Rate ;

(a) 'Cyclic' $J = 6$, $n_A = 6552$, $R = 13/14$

(b) C.S.O.C. $J = 6$, $n_A = 6552$, $R = 13/14$.

There is one other code in Table 14.1 where $J = 3$, $n_A = 11,988$, $R = 36/37$ for both 'cyclic' and C.S.O.C.

For the C.S.O.C., from Wu[22], the rate 13/14 C.S.O.C. has $k_o = 13$, $n_o = 14$ and requires buffer storage of 6552 and 13 majority-logic gates, plus of course a re-encoder.

The 'cyclic' code has $k_o = 78$, $n_o = 84$ and since we have $\frac{(m-1)}{s} = \frac{78}{6} = 13$ cosets we also require 13 majority-logic gates, buffer storage 6552 plus re-encoder. Thus in terms of equipment required there is perhaps little to choose between these two codes. In general both classes of codes require x majority-logic gates to decode a rate $x/(x+1)$ code. Wu[22] presented a system for replacing the x majority gates by a single gate plus a combinational gating circuit and this idea carries over to the 'cyclic' codes also. The gating circuit changes connections once for each coset to decode all k_o digits of a sub-block. Of course this would require the 'cyclic' array being clocked vertically k_o times and the decoding time is increased as it would also be with the C.S.O.C.'s.

TABLE 14.1.

Comparison of 'cyclic' and C.S.O.C. codes.

J	n_A 'Cyclic'	n_A C.S.O.C.	R.	
2	8	6	.	R
4	32	14	.	R
6	72	36	.	R
10	200	112	.	R
12	288	172	.	R
16	512	360	.	R
18	648	434	1/2	R
22	968	718	.	R
28	1568	1460	.	
30	1800	1682	.	
42	3698	3414	.	
60	7200	6962	.	
72	10368	10082	.	
82	13448	13124		

J	n_A 'Cyclic'	n_A C.S.O.C.	R.	
2	24	9	.	R
3	54	24	.	R
5	150	69	.	R
6	216	120	.	R
8	384	237	.	R
9	486	306	.	R
11	726	507	2/3	R
14	1176	867	.	R
15	1350	1011	.	R
20	2400	2217	.	
23	3174	2499	.	
26	4056	3696	.	
35	7350	6933	.	
50	15000	14271	.	

J	n_A 'Cyclic'	n_AC.S.O.C.	R.	
2	48	16	.	R
4	192	80	.	R
6	432	248	3/4	R
10	1200	812	.	R
12	1728	1620	.	
3	180	65	4/5	R
4	320	135	4/5	R
7	980	615	4/5	R
9	1620	1445	.	
13	3380	3165	.	
18	6480	6310	.	
22	9680	9445	.	
2	120	36	.	K
6	1080	630	5/6	K
14	5880	5934	.	*
2	168	49	.	K
3	378	140	.	K
5	1050	539	.	K
6	1512	882	6/7	K
10	4200	4550	6/7	*
11	5082	5152	.	*
13	7098	7546	.	*
16	10752	11032	.	*
4	896	384	7/8	K
6	2016	1176	7/8	K
5	1800	1368	8/9	
11	8712	9531	.	*

J	n_A 'Cyclic'	n_A C.S.O.C.	R.	
3	990	715	.	
4	1760	1529	.	
6	3960	4235	10/11	*
7	5390	5797	.	*
10	11000	12078	.	*
3	1404	1664	.	*
5	3900	4290	12/13	*
6	5616	5512	12/13	
8	9984	13158	.	*
6	6552	6552	13/14	E
3	1890	1425	14/15	
8	13440	16335	14/15	*
3	2448	2006	16/17	
6	9792	11594	16/17	*
6	11016	14832	17/18	*
4	5472	5947	18/19	*
5	10500	13335	20/21	*
3	4554	5750	22/23	*
3	5400	6750	24/25	*
4	9600	9925	24/25	*
4	12992	16462	28/29	*

J	n_A 'Cyclic'	n_A C.S.O.C.	R.	
3	8370	9727	30/31	*
3	10710	13860	34/35	*
3	11988	11988	36/37	E
3	15750	19823	42/43	*

*	Indicates 'cyclic' code is shorter.
E	Indicates 'cyclic' code is equal.
K	From the class of codes by Klieber[14].
R	From the class of codes by Robinson[7] et al.
	All other codes from Wu[22,23,24].

14.2 Performance of the Codes.

In section 12.2. we presented equations for calculating effective constraint length n_e. Comparative figures were presented in Table 12.2.1. where the 'cyclic' codes n_e are compared with n_e for other C.S.O.C.'s. It can be seen that for rates > 1/2 the 'cyclic' codes appear to have shorter n_e, the difference increasing as rate and J increases. Some insight into this can be obtained from the lower bounds developed in Appendix A. The lower bound on n_e for C.S.O.C.'s is given by:

$$\underset{\text{C.S.O.C.}}{n_e} \geqslant \frac{1}{2}\left(xJ^2 + J + x + 1\right) \qquad 14.2.1.$$

and for the cyclic C.S.O.C.'s:

$$\underset{\text{'cyclic'}}{n_e} \geqslant \frac{1}{2}\left(xJ^2 + J + 2\right) \qquad 14.2.2.$$

for codes of rate x/(x+1), with J check sums per digit of a sub-block. The values of n_e in Table 12.2.1. are compared with their bounds in Table 14.2.1.

The following results, from Table 14.2.1. are apparent;

(a) the 'cyclic' codes n_e is optimum with the 'cyclic' lower bound for the majority of the codes shown.

(b) the C.S.O.C. codes n_e moves away from the C.S.O.C. lower bound as rate and J increase.

(c) from inequalities 14.2.1. and 14.2.2. it is seen that the 'cyclic' bound is tighter than the C.S.O.C., the C.S.O.C. being greater by $\left(\frac{x-1}{2}\right)$ and therefore increasing with rate.

The property (c) accounts for the 'cyclic' codes effective constraint length n_e becoming increasingly smaller as rate increases.

The property (b) shows that as a construction technique the C.S.O.C. codes ratio d_m/n_e deteriorates more than it should and implies tighter constructions are possible.

The fact that the 'cyclic' C.S.O.C.'s appear to hold to the lower bound on n_e is, the author believes, due to the highly symmetric cyclic structure of the codes. This results in all n_{n_i}, for the cosets, being close to the mean, $\overline{n}_n$, and in those results presented in Table 14.2.1. only two codes have n_{n_i} not exactly on the mean $\overline{n}_n$.

In the C.S.O.C.'s however the x(J-1) check sums in the check-digits of blocks other than block (O) are not symmetric and appear to have n_{n_i} which deviate about the C.S.O.C. mean. This deviation appears to be more pronounced for high rate, high error-correcting C.S.O.C.'s.

For both inequalities 14.2.1. and 14.2.2. for rate 1/2 codes, $x = 1$ and

$$n_e \geqslant \frac{1}{2} J^2 + \frac{1}{2} J + 1$$

and equality gives the optimum n_e for rate 1/2 C.S.O.C.'s as shown by Massey[1]. Massey[1] also showed that the effectiveness of the threshold decoding of C.S.O.C.'s was related to the ratio $J/2n_e$ compared to the ratio $(d-1)/2\, n_o N$ guaranteed by the Gilbert Bound. Since for a given J, n_e 'cyclic' < n_e C.S.O.C. at high rates the 'cyclic' codes are closer to the Gilbert Bound than the C.S.O.C.'s.

TABLE 14.2.1.

Table of n_e compared to lower bounds in Appendix A, for C.S.O.C.'s and 'cyclic' C.S.O.C.'s.

J	n_e 'Cyclic'	$n_e \geq$	n_e C.S.O.C.	$n_e \geq$	Rate.
2	4	4	4	4	1/2
4	11	11	11	11	.
6	22	22	22	22	.
8			37	37	.
10	56	56	56	56	.
12	79	79	79	79	.
14			106	106	.
16	137	137	137	137	.
18	172	172	172	172	.
20			211	211	.
22	254	254	254	254	.
24			301	301	.
26					
28	407	407			1/2
2	6	6	7	7	2/3
4			20	20	.
6	40	40	41	41	.
8	69	69	70	70	.
10			111	107	.
12			152	152	.
14	204	204	218	205	.
16					
18	334	334			2/3
2	8	8	10	9	3/4
4	27	27	31	28	.
6	58	58	66	59	.
8			105	102	.
10	156	156	181	157	.
12	223	223			3/4

J	n_e 'Cyclic'	$n_e \geqslant$	n_e C.S.O.C.	$n_e \geqslant$	Rate.
2	10	10	13	12	4/5
4	35	35	39	37	.
6			83	78	.
8			161	135	.
10	206	206			4/5
26	690	690	743	691	2/3
3	75	56	82	62	12/13
5	184	154	209	159	12/13
6	238	238	339	244	13/14

The upper bound on P(t+1) developed in Appendix B. is not a tight bound if:

$$N'(e)_{t+x} \ll \binom{n_e}{t+x}$$

although B.6. is tighter than B.7. However it does indicate the importance of the ratio n_e/n_oN for (t+1) errors.

When (t+x) > (t+1) the second term in equation 12.3.2. becomes more dominant, however tightening of inequality 12.3.3.(ii) is necessary to make an extended bound tighter.

A comparison of the figures in Table 12.3.1, with upper bound B.6., is given in Table 14.2.2.

The underlying result which emerges from this examination of the unbounded performance of the codes is that, for a given rate and length n_oN, if one increases the bounded error-correcting capability, and thus the ratio n_e/n_oN, the unbounded performance deteriorates.

TABLE 14.2.2.

Comparison of calculated P(t+1) with,

$$P(t+1) < \left(\frac{n_e - t}{n_o N - t}\right)^{t+1}$$

	calculated	upper bound
'Cyclic' $n_e = 22$, $t=3$, $n_o N = 72$	$2{\cdot}39{\cdot}10^{-3}$	$5{\cdot}74{\cdot}10^{-3}$
C.S.O.C. $n_e = 22$, $t=3$, $n_o N = 36$	$41{\cdot}7{\cdot}10^{-3}$	$109{\cdot}89{\cdot}10^{-3}$
C.S.O.C. $n_e = 37$, $t=4$, $n_o N = 72$	$6{\cdot}4{\cdot}10^{-3}$	$26{\cdot}91{\cdot}10^{-3}$
'Cyclic' $n_e = 69$, $t=4$, $n_o N = 384$	$2{\cdot}10^{-5}$	$14{\cdot}6{\cdot}10^{-5}$
C.S.O.C. $n_e = 70$, $t=4$, $n_o N = 261$	$19{\cdot}3{\cdot}10^{-5}$	$111{\cdot}7{\cdot}10^{-5}$
C.S.O.C. $n_e = 111$, $t=5$, $n_o N = 393$	$2{\cdot}43{\cdot}10^{-5}$	$42{\cdot}2{\cdot}10^{-5}$

This result is reflected in figures 12.4.1. and 12.4.2. where by increasing $n_o N$, with n_e constant, we see the probability decrease. It is the author's conjecture that this result extends to block codes also and underlies the price one pays for increased bounded error-correcting capability.

Shannon showed that probability of error was intimately connected with a code's length n, such that[26],

$$P(e) \leqslant e^{-nE(R)}$$

where E(R) is a function of rate R. The curves in figures 12.4.2. and 12.4.1. illustrate this inequality and also highlight the fact that the length of a code is a significant factor in determining the probability of error at the decoder output. Since this is so one could expect a reasonable performance from a 'code' whose n_o-k_o redundant digits are all zero's, provided n_oN is long enough. However this is wasting space and can be simulated by time multiplexing many information independent blocks of digits.

For example if we time multiplex 5 blocks

($\longleftarrow$ n_o-k_o for block B.1. $\longrightarrow$)

B.5.	B.4.	B.3.	B.2.	B.1.

of k_o-digits per block, each block being unrelated to any other, from an information content view, then considering only one block, say B.1., the $5 \cdot k_o$ digits can be thought of as a $(n = 5k_o, k_o)$ code with bounded error-correcting capability $t = 0$. Certainly a good unbounded performance would be expected and points out the power of time multiplexing in the context of error-correction.

If we express equation 12.3.1. as a percentage, then from Table 12.3.1., we see that the $t = 3$, $n_oN = 72$ 'cyclic' C.S.O.C. is still correctly decoding more than 92% of all error patterns of weight 9. By putting this figure into context we can see further evidence of the importance of code length n_oN.

Since we can permit up to 3 errors affecting the n_e digits in the check sums, then the number of patterns of errors of weight 9 which cannot possibly cause error is given by:

$$= \sum_{i=0}^{3} \frac{(n_oN - n_e)!}{(9-i)!\,(n_oN-n_e-9+i)!} \cdot \frac{n_e!}{i!(n_e - i)!}$$

And for the $t = 3$, $n_oN = 72$ cyclic code this gives the figure:

$$= \underline{\underline{6 \cdot 1785 \cdot 10^{10}}}$$

This is in fact 72.59% of all patterns of weight 9 and is due to the fact that $n_oN >> n_e$ or that n_e/n_oN is small. Thus·the actual code structure is only correctly decoding approximately 20% of the patterns, the codes length does the rest of the work.

Some interesting results were presented by Mr. K. Tsigiroglou, in a report entitled "Performance evaluation of a new class of Convolutional codes", for his M.Sc. Thesis at Loughborough University of Technology in September 1978. Three codes were simulated on the University's computer namely,

Rate = 1/2,	$n_oN = 72$,	$t = 5$	Massey.
Rate = 1/2,	$n_oN = 72$,	$t = 4$	Robinson/Bernstein.
Rate = 1/2,	$n_oN = 72$,	$t = 3$	'Cyclic'.

X From the curves developed it can be seen that the computer simulations bear out the calculated results obtained in section 12.3.

Also, the Massey code, though of the largest bounded error-correcting capability, has by far the worst overall unbounded performance. There are two main reasons for this, (a) the high ratio n_e/n_oN increases the probability of a decoding failure when $t+x > t$ errors occur. Being an orthogonalizable code n_e/n_oN is close to unity, (b) being orthogonalizable the code suffers from error propagation and tends to generate more errors when a decoding failure occurs.

Although it is known that self-orthogonal convolutional codes automatically limit error propagation[7,12], to actually calculate L is not a simple matter. The most interesting result from the figures in Table 12.5.1. are that both C.S.O.C.'s appear to become more efficient as t increases. The figures shown are the largest that could be found, but are in no way claimed as being the maximum possible since L is different for practically every error pattern which causes propagation. The procedure given in section 12.6.2. was used for both 'cyclic' and C.S.O.C. and can be considered a general procedure for any type of C.S.O.C.

For both the high rate C.S.O.C.'s, decoded with k_o majority gates, and the high rate 'cyclic' C.S.O.C.'s, decoded with $\frac{m-1}{J}$ majority gates, simultaneous feedback syndrome cancellation of these digits before the next decoding operation can cause multiple error propagation if more than one digit is decoded in error. That is, for both codes, with rate $\frac{x}{x+1}$, x digits are decoded simultaneously in one time instant.

For the rate 1/2 'cyclic' C.S.O.C.'s $J = k_o$ and a set of check sums on digit b, from say block 0, cannot contain digit b from any other block as this would imply that a row of the 'cyclic' array contains two identical integers which is not possible by definition of the array. Thus if t+1 errors occur in the k_o message-digits of block 1, they can only occur simultaneously in the check sums of t-1 digits from block 0. This causes t-1 errors to propagate simultaneously. However t-1 propagated errors will not cause any digit from block 1 or any other block to decode in error providing no more errors occur until beyond block N+t. Therefore, the codes can recover from the multiple propagation of t-1 errors, caused by $k_o!/(t+1)!\,(k_o-t-1)!$ error patterns of weight t+1, with propagation length:

$$L = n_o(N+t).$$

The ability of the 'cyclic' rate 1/2 codes to do this, again, is primarily due to the symmetry of the codes' structure. Note that the value of L given above is less than the figures given in Table 12.5.1. for propagation of a single error, this being due to the choice of a particular pattern of errors.

14.3 Further Work.

The rich algebraic structure of this class of cyclically decodable C.S.O.C.'s indicates that a deeper mathematical examination of their structure could bear further fruit. There may be some relationship between the C.S.O.C.'s and 'cyclic' C.S.O.C.'s, particularly those of Wu,[22,23,24] which would shed further light on C.S.O.C.'s in general. The sharing of different values of J, for a given rate, by Wu's C.S.O.C.'s and the 'cyclic' C.S.O.C.'s seems significant.

The pseudostep algorithm presented in Chapter 13 can also be applied to the 'cyclic' C.S.O.C.'s. The problem revolves around the imbedding, into the array, of a set of m-1/s columns which satisfy the algorithm, without disturbing the cyclic property of the code. Specific results have been obtained on a few low rate codes, $\leqslant 5/6$, in which an extra non-orthogonal check sum on every message-digit has been imbedded without altering the code length or 'cyclic' property. Generalizing this idea to one of imbedding further orthogonal check sums has occurred to the author but remains a subject for further study. The specific results mentioned above produced codes that are Self-Pseudostep Orthogonal and will therefore recover from error propagation.

The author feels sure that the work begun in the Appendices can be extended upon, particularly B. From the results presented in **Mr. T. Sigiróglou's report** it is conjectured by the author that the great difference in performance between the C.S.O.C.'s and Massey's orthogonalizable code is primarily due to the differences in their n_e/n_A ratio's and the fact that Massey's code cannot recover from error propagation is secondary. This is a conjecture that certainly needs examination and far more work is required in this direction to obtain meaningful conclusions. If it can be shown that error propagation has insignificant effects providing a code's n_e/n_A ratio is below some threshold, regardless of whether or not the code has automatic recovery properties, then this would be a **useful** contribution to convolutional coding theory.

Another avenue perhaps worth pursuing is that of discovering if the distribution of all n_{n_i} about the mean $\overline{n}_n$ has a bearing on code performance. Since a digit from a sub-block has probability of error closely related to its own effective constraint length n_{e_i} and therefore n_{n_i}, then the distribution of the probabilities of error of the k_o digits from a sub-block will presumably be connected with the distribution of n_{n_i} about the mean $\overline{n}_n$.

As stated previously the values $N'(e)_{t+x}$ are usually much less than $\left(\frac{n_e}{t+x}\right)$ and further work is necessary to find a tighter approximation.

In conclusion, there still remains a great deal of work to be done in assessing the impact and general performance of this class of 'cyclic' C.S.O.C.'s, however the results so far obtained indicate that as a class they exhibit many favourable qualities.

Appendix A

A lower bound on n_e for C.S.O.C.'s and 'cyclic' C.S.O.C.'s.

(a) Lower bound on C.S.O.C.'s.

For C.S.O.C.'s the check-digit of the current block to be decoded, as a syndrome digit, has the equation:

$$s_o = e_o(1) \oplus e_o(2) \oplus \ldots . \oplus e_o(k_o) \oplus e_o(k_o+1). \qquad \text{A.1.}$$

This single syndrome digit provides one check sum for every message-digit error from block 0. The remaining J-1 check sums, for each digit, are in the check-digits of the other N-1 blocks. Considering one digit from block 0 let its effective constraint length be given by:

$$n_e = \left(n_n + (J-1)\right) + 1 + (x-1) + 1 \qquad \text{A.2.}$$

Since $k_o = x$ for a rate x/x+1 C.S.O.C., from equation A.1. we obtain the

$$1 + (x-1) + 1$$

digits of n_e. That is, the digit itself, the (x-1) digits from its own message block and its own blocks' check-digit. Therefore n_n is the total number of digits from other message blocks contained in n_e.

Let $n_{n_1}, n_{n_2}, \ldots, n_{n_x}$ be the values of n_n for the x digits in the current block to be decoded, then it is quite simple to show that:

$$\sum_{i=1}^{x} n_{n_i} = \sum_{x}^{xJ-1} i \qquad \text{A.3.}$$

and we can propose a mean value,

$$\bar{n}_n = \frac{1}{x}\left(\sum_{x}^{xJ-1} i\right) \qquad \text{A.4.}$$

From A.3., if any $n_{n_i} < \bar{n}_n$, then there must be some $n_{n_j} > \bar{n}_n$. Thus, since n_e is a maximum figure,

$$\begin{aligned} n_e &\geq \bar{n}_n + J + 1 + (x-1) \\ &\geq \frac{1}{x}\left(\sum_{x}^{xJ-1} i\right) + J + 1 + (x-1) \\ &\geq \frac{1}{2}\,(xJ^2 + J + x + 1). \qquad \text{A.5.} \end{aligned}$$

Equality is satisfied if $n_{n_i} = \bar{n}_n$ for all $i = 1,2,\ldots,x$.

<u>Example 2.1.</u>

For the code $t = 2$, $R = 2/3$, $nN = 42$ the generator sequences are:

(0, 8, 9, 12) on digit 1.

(0, 6, 11, 13) on digit 2.

Analysis shows that the number of digits in the check sum, contained in the check-digit of block x (represented by the integer x in the generator sequences), excluding the digit itself, is equal to the number of integers less than x in the generator sequences.

Therefore,

$$n_{e_1} = (3 + 4 + 6) + 4 + 1 + 1 = n_{n_1} + 6 = \underline{\underline{19}}$$

$$n_{e_2} = (2 + 5 + 7) + 4 + 1 + 1 = n_{n_2} + 6 = \underline{\underline{20}}$$

so that

$$n_e = \max(n_{e_1}, n_{e_2}) = \underline{\underline{20}}\ .$$

Note that,

$$n_{n_1} + n_{n_2} = 13 + 14 = 27 = \sum_{2}^{7} i \; .$$

From the lower bound A.5.,

$$n_e \geqslant 19{\cdot}5 \geqslant 20 \; .$$

So that this code is optimum in the sense that n_e cannot be lower.

(b) Lower bound on 'cyclic' C.S.O.C.'s.

From equation 12.2.2. the effective constraint length is:

$$n_{e_i} = 1 + \sum_{j=1}^{S} x_i \, a^j \bmod (m) \qquad \text{A.6.}$$

Let,

$$n_{n_i} = \sum_{j=1}^{S} x_i \, a^j \bmod (m) \qquad \text{A.7.}$$

since the $\frac{m-1}{S} = x$ cosets contain all positive integers from 1 to $m-1 = x\,S$

$$\sum_{i=1}^{m-1/S} n_{n_i} = \sum_{i=1}^{xS} i$$

Since S = J,

$$\overline{n}_n = \frac{1}{x}\left(\sum_{1}^{xJ} i\right) \qquad \text{A.8.}$$

Therefore, as before and from A.6. and 7

$$\begin{aligned} n_e &\geqslant 1 + \overline{n}_n \\ &\geqslant 1 + \frac{1}{x}\left(\sum_{1}^{xJ} i\right) \\ &\geqslant 1 + \frac{1}{x}\left(\frac{xJ(xJ+1)}{2}\right) \\ &\geqslant \frac{1}{2}\,(xJ^2 + J + 2) \end{aligned} \qquad \text{A.9.}$$

with equality when $n_{n_i} = \overline{n}_n$ for all $i = 1,2,\ldots,x$.

APPENDIX B

An upper bound on $P_{x_i A}(t+x)$.

From inequality 12.3.3.(ii),

$$N'(e_i)_{t+x} < \frac{(n_{e_i})!}{(t+x)!\ (n_{e_i} - t-x)!} \qquad \text{B.1.}$$

and therefore from equation 12.3.2.,

$$N(e_i)_{t+x} < \binom{n_{e_i}}{t+x} + \sum_{j=1}^{x-1} \binom{n_oN - n_{e_i}}{j}\binom{n_{e_i}}{t+x-j} \qquad \text{B.2.}$$

where $\binom{a}{b} = \frac{a!}{b!\ (a-b)!}$

So that from equation 12.3.1.,

$$P_{x_i A}(t+x) = \frac{N(e_i)_{t+x}}{\binom{n_oN}{t+x}} \quad , \quad \text{or from B.2.}$$

$$P_{x_i A}(t+x) < \frac{\binom{n_{e_i}}{t+x} + \sum_{j=1}^{x-1} \binom{n_oN - n_{e_i}}{j}\binom{n_{e_i}}{t+x-j}}{\binom{n_oN}{t+x}} \qquad \text{B.3.}$$

Since n_e is the maximum n_{e_i},

$$P(t+x) < \frac{\binom{n_e}{t+x} + \sum_{j=1}^{x-1} \binom{n_oN - n_e}{j}\binom{n_e}{t+x-j}}{\binom{n_oN}{t+x}} \qquad \text{B.4.}$$

applies to all digits in a sub-block.

Let x = 1, then

$$P(t+1) < \frac{\binom{n_e}{t+1}}{\binom{n_oN}{t+1}}$$

$$< \frac{(n_e - t) \ldots\ldots\ (n_e)}{(n_oN - t) \ldots\ldots\ (n_oN)} \qquad \text{B.5.}$$

although $\left(\frac{n_e - t}{n_oN - t}\right)^{t+1} < \left(\frac{n_e}{n_oN}\right)^{t+1}$

usually

$$N'(e)_{t+x} << \binom{n_e}{t+x}$$

therefore we can say,

$$P(t+1) < \left(\frac{n_e - t}{n_oN - t}\right)^{t+1} \qquad \text{B.6.}$$

and if $n_e >> t$, $n_oN >> t$

$$< \left(\frac{n_e}{n_oN}\right)^{t+1} \qquad \text{B.7.}$$

To develop a tighter bound it is necessary to tighten inequality B.1.

REFERENCES

1. MASSEY, J.L. "Threshold Decoding", 1963, The M.I.T. Press, Cambridge, Massachusetts.

2. WOZENCRAFT, J.M. and REIFFEN, B. "Sequential Decoding", 1961, John Wiley & Sons Inc., N.Y.

3. ELIAS, P. "Coding for Noisy Channels", 1955, I.R.E. Convention Record, Part 4, pp.37-47.

4. LeVEQUE, W.J. "Topics in Number Theory", 1956, Addison-Wesley Pub. Co. Inc., Massachusetts.

5. McCOY, N.H. "The Theory of Numbers", 1965, The Macmillan Co., N.Y.

6. BIELER, A.H. "Recreations in the Theory of Numbers: The Queen of Mathematics Entertains", 1966, Dover Publications Inc., N.Y.

7. ROBINSON, J.P. and BERNSTEIN, A.J. "A Class of Recurrent Codes with Limited Error Propagation", 1967, I.E.E.E. Trans. Inf. Thy., Vol. IT-13, pp.106-113.

8. VITERBI, A.J. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", 1967, I.E.E.E. Trans. Inf. Thy., Vol. IT-13, pp.260-269.

9. REDDY, S.M. and ROBINSON, J.P. "A Construction for Convolutional Codes using Block Codes", 1968, Inf. and Cont., Vol. 12, pp.55-70.

10. REDDY, S.M. "Further Results on Convolutional Codes Derived from Block Codes", 1968, Inf. and Cont., Vol. 13, pp.357-362.

11. REDDY, S.M. and ROBINSON, J.P. "A Decoding Algorithm for some Convolutional Codes Constructed from Block Codes", 1968, Inf. and Cont., Vol. 13, pp.492-507.

12. ROBINSON, J.P. "Error Propagation and Definite Decoding of Convolutional Codes", 1968, I.E.E.E. Trans. Inf. Thy., Vol. IT-14, pp.121-128.

13. RUDOLPH, L.D. "Generalised Threshold Decoding of Convolutional Codes", 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.739-745.

14. KLEIBER, E. "Some Difference Triangles for Constructing Self-orthogonal Codes", 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.237-238.

15. TONG, S.Y. "Systematic Construction of Self-orthogonal Diffuse Codes", 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.594-604.

16. MORRISSEY, T.N. Jnr. "Analysis of Decoders for Convolutional Codes by Stochastic Sequential Machine Methods", 1970, I.E.E.E. Trans. Inf. Thy., Vol. IT-16, pp.460-469.

17. LIN, S. "An Introduction to Error-Correcting Codes", 1970, Prentice-Hall Inc., Englewood Cliffs, N.J. Chapters 10 and 11.

18. FERGUSON, M.J. "Diffuse Threshold Decodable Rate 1/2 Convolutional Codes", 1971, I.E.E.E. Trans. Inf. Thy., Vol. IT-17, pp.171-180.

19. PETERSON, W.W. and WELDON, E.J. Jnr. "Error-Correcting Codes", 1972, The M.I.T. Press, Cambridge, Mass., Chapter 13.

20. REDDY, S.M. and ROBINSON, J.P. "Hybrid Block-Self-Orthogonal Convolutional Codes", 1972, I.E.E.E. Trans. Inf. Thy., Vol. IT-18, pp.185-191.

21. RUDOLPH, L.D. and ROBBINS, W.E. "One-step Weighted-Majority Decoding", 1972, I.E.E.E. Trans. Inf. Thy., Vol. IT-18, pp.446-448.

22. WU, W.W. "New Convolutional Codes - Part I", 1975, I.E.E.E. Trans. Communic., Vol. Com-23, pp.942-955.

23. WU, W.W. "New Convolutional Codes - Part II", 1976, I.E.E.E. Trans. Communic., Vol. Com-24, pp.19-33.

24. WU, W.W. "New Convolutional Codes - Part III", 1976, I.E.E.E. Trans. Communic., Vol. Com-24, pp.946-955.

25. GOODMAN, R.M.F. "Soft-Decision Threshold Decoding of Convolutional Codes", 1977, I.E.R.E. Conf. on Digital Process., Univ. of Loughborough, Leics., England.

26. LIN, S. "An Introduction to Error-Correcting Codes", 1970, Prentice-Hall Inc., Englewood Cliffs, N.J. Chapter 1.

27. PALEY, H. and WEICHSEL, P.M. "Element of Abstract and Linear Algebra", 1972, Holt, Rinehart and Winston, Inc., N.Y.

28. HALL, H.S. and KNIGHT, S.R. "Higher Algebra. A Sequel to Elementary Algebra for Schools", 1964, page 350, Macmillan and Co. Ltd., London.