SYNCHRONIZABLE ERROR-CORRECTING CODES

by

Joseph George Caldwell
University of North Carolina, Chapel Hill
and
Research Triangle Institute, Durham

Institute of Statistics Mimeo Series No. 469
May 1966

A new technique for correcting synchronization errors in the transmission of discrete-symbol information is developed. The technique can be applied to any t-additive-error-correcting Bose-Chaudhuri-Hocquenghem code, to provide protection against synchronization errors involving any desired range of symbol losses or gains. The synchronization error is corrected at the first complete received word after the word containing the synchronization error, even if this following word contains up to t additive errors. An upper bound is derived for the capability of a class of codes that can simultaneously correct synchronization errors and additive errors. This upper bound is an extension of the Hamming upper bound for the capability of additive-error-correcting codes. The capability of the new synchronization technique is compared with the upper bound. It is shown how the redundancy associated with the synchronization-error-correcting ability can be sacrificed to enable additional additive-error correction. Examples are presented illustrating in detail the application of the technique.

This research was supported in part by the National Science Foundation Grant No. GP-5790 and Army Research Office, Durham Grant No. DA-ARO-D-31-124-6670

DEPARTMENT OF STATISTICS
UNIVERSITY OF NORTH CAROLINA
Chapel Hill, N. C.
ACKNOWLEDGEMENTS

I gratefully acknowledge the guidance and help given me by my advisor, Prof. R. C. Bose, throughout the course of this research. For his excellent direction and for the mathematical and statistical theory he has taught me, I shall remain grateful.

I wish to thank the other members of my doctoral committee, Prof. I. M. Chakravarti, Prof. W. J. Hall, Prof. N. L. Johnson, Prof. R. R. Kuebler, Jr., and Prof. D. M. Mesner for their helpful suggestions and comments. I am particularly indebted to Prof. Kuebler for his diligent and thorough review of my work. His many excellent suggestions substantially improved the dissertation.

For financial assistance at various stages of my studies – undergraduate and graduate – I wish to express my appreciation to the General Motors Corporation, the National Aeronautics and Space Administration, the Carnegie Institute of Technology, and the Statistics Department of the University of North Carolina; to my mother and father; and to my parents-in-law, Mr. and Mrs. L. H. Tinsley. Finally, I am grateful to the Research Triangle Institute, for enabling me to complete my formal education while in its employ.

I wish to express my profound gratitude to my wife, Timi, for the steadfast encouragement she has given me throughout my academic studies. For her help in so many ways, and for the innumerable sacrifices that she and the boys, Jay and Chris, endured so that I might further my education, I shall be eternally grateful.
Finally, I extend thanks to Mrs. Kay Herring and Mrs. Judy Zeuner for their conscientious and patient typing of the manuscript.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>xiii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>xiv</td>
</tr>
</tbody>
</table>

## I  INTRODUCTION

1.1. Channel Model | 1 |
1.2. Additive Errors | 5 |
1.3. Synchronization Errors | 12 |
1.4. Previous Approaches to Synchronization-Error Correction | 18 |
1.5. The Synchronization Technique Introduced in this Dissertation | 34 |

## II  BASIC CONCEPTS IN THE THEORY OF LINEAR BLOCK CODES

2.1. Introduction | 36 |
2.2. Preliminary Concepts | 36 |
2.3. Linear Codes | 39 |
2.4. Cyclic Codes | 42 |
2.5. Bose-Chaudhuri Codes | 47 |
2.6. Examples of BCH Codes | 49 |
Chapter  

III A NEW TECHNIQUE FOR CORRECTING SYNCHRONIZATION ERRORS. 54  
3.1. Introduction ........................................ 54  
3.2. The Technique. ....................................... 60  
3.3. A Simple Expression for the Truncated Word .... 64  
3.4. Determination of the Translation Vector and  
the Subcode: the Fundamental Theorem ............ 66  

IV THE EXTENDED HAMMING BOUND ......................... 79  
4.1. Introduction ......................................... 79  
4.2. The Extended Hamming Bound ..................... 84  
4.3. Comparison of the Ordinary and the Extended  
Hamming Bounds ........................................... 93  

V EXAMPLES ILLUSTRATING THE NEW SYNCHRONIZATION TECHNIQUE. 98  
5.1. Introduction ......................................... 98  
5.2. Table for Constructing \((t, t_1)-\text{Error-Correcting}  
\text{Bose-Chaudhuri Codes, for } p = 2 \text{ and } n \leq 255.\) .... 99  
5.3. Application of the Technique to a BCH Binary  
Code of Length \(n = 15.\) ............................ 104  
5.4. Application of the Technique to BCH Binary  
Codes of Lengths \(n = 63 \text{ and } n = 255.\) .......... 115  
5.5. The Use of a Reducible Factor \(m(x)\) for Some  
Ranges of \(t_s.................................. 118  
5.6. Use of the Synchronization-Error-Correcting  
Capability of the Code to Enable Additional  
Additive-Error Correction. .......................... 120  

VI COMPARISON OF THE NEW SYNCHRONIZATION TECHNIQUE WITH  
PREVIOUS TECHNIQUES. ................................ 130  
6.1. Introduction ......................................... 130  
6.2. Comparison with Previous Techniques............ 131  
6.3. Some Remarks Concerning the Occurrence  
of Uncorrectable Synchronization Errors Using  
the New Technique. ............................... 157
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIBLIOGRAPHY</td>
<td>161</td>
</tr>
</tbody>
</table>
ABSTRACT

A new technique for correcting synchronization errors in the transmission of discrete-symbol information is developed. The technique can be applied to any $t_a$-additive-error-correcting Bose-Chaudhuri-Hocquenghem code. The synchronization error is corrected at the first complete received word after the word containing the synchronization error, even if this following word contains up to $t_a$ additive errors. A basic feature of the technique is that to each codeword are adjoined redundant symbols depending on the codeword. An upper bound is derived for the capability of a class of codes that can simultaneously correct synchronization errors and additive errors. This upper bound is an extension of the Hamming upper bound for the capability of additive-error-correcting codes. The capability of the new synchronization technique is compared with the upper bound. The technique can be applied to provide protection against synchronization errors involving a range of symbol losses or gains. It is shown how the redundancy associated with the synchronization-error-correcting ability can be sacrificed to enable additional additive-error correction. Examples are presented illustrating in detail the application of the technique. A table is presented which provides the information necessary to apply the technique to binary Bose-Chaudhuri-Hocquenghem codes of length up to 255. The new technique is compared with previous techniques, and parameter ranges and situations in which the new technique is more efficient than other techniques are discussed.
SUMMARY

Coding theory is concerned with the development of techniques by which information can be efficiently and accurately transmitted over noisy channels, i.e., channels in which the transmitted signal can be disturbed, or altered, so that the receiver does not receive the same message as was sent by the transmitter. In the case of a discrete noisy channel, the transmitter sends discrete symbols over the channel. Until recently, discrete coding theory has been concerned primarily with developing codes for detecting or correcting additive errors, i.e., errors in which transmitted symbols can be changed into other symbols. The noise causing such errors is known as additive noise, or amplitude noise. Another source of error in digital information transmission systems is so-called "time noise," by which transmitted symbols can be lost or gained, rather than simply changed, as in the case of additive noise. An error (such as a symbol loss or gain) due to time noise is called a synchronization error. The descriptor "synchronization" arises due to the fact that individual message symbols generally have meaning to the receiver only as parts of long sequences of symbols. If symbols are lost or gained, the receiver incorrectly separates the incoming train of symbols into sequences, and reception is said to be out of synchronization with transmission.

Sophisticated codes for additive-error correction have been developed using the mathematics of finite fields. The most important
class of additive-error-correcting codes is the class of Bose-Chaudhuri-Hocquenghem codes. The Bose-Chaudhuri-Hocquenghem codes are block codes. That is, blocks of $k$ symbols are transformed into blocks of $n = k + r$ symbols, which are then sent over the channel. Thus $r$ redundant symbols are added to each sequence of $k$ information symbols, to form a codeword of $n$ symbols. The receiver receives the sequence of $n$ symbols, some of which may be in error, and attempts to determine which codeword was sent. Correction of certain errors is possible because of the fact that the set of codewords is a proper subset of the set of all sequences of length $n$; the occurrence of a few errors does not necessarily change a codeword into another codeword. A block code of length $n$ which can correct any combination of up to $t_a$ additive errors occurring in a word is called a $t_a$-additive-error-correcting code. Bose-Chaudhuri-Hocquenghem codes exist for a range of values of $t_a$, for suitable values of $n$. An essential characteristic of block coding is that the redundant symbols added to each $k$-sequence depend on the $k$-sequence.

Developments in synchronization-error-correcting codes have been less impressive than those in additive-error-correcting codes. The procedures employed include the use of special synchronizing symbols (which are themselves code symbols), and synchronization sequences, which are fixed patterns of code symbols adjoined to each codeword. Little theoretical work has been done to determine upper bounds to the efficiency of synchronization techniques. An essential feature of the previous synchronization techniques (such as the synchronization sequence technique) which add redundant symbols to codewords is that the redundant symbols are fixed, and are thus independent of the
codeword, in contrast to the additive-error correction situation.

This dissertation will present a new technique for correcting synchronization errors. The technique can be applied to any $t_a$-additive-error-correcting Bose-Chaudhuri-Hocquenghem code. The synchronization error is corrected at the first complete received word after the word containing the synchronization error, even if this following word contains up to $t_a$ additive errors. As in the case of additive-error correction, to each codeword are added redundant symbols depending on the codeword. An upper bound is derived for the capability of a class of codes that can simultaneously correct synchronization errors and additive errors. This upper bound is an extension of the Hamming upper bound for the capability of additive-error-correcting codes. Comparison of the capability of the new synchronization technique with the upper bound indicates that the new technique is an efficient procedure for correcting synchronization errors that are not large compared with the codeword length.

Just as Bose-Chaudhuri-Hocquenghem codes exist for a range of values of $t_a$, the new synchronization technique can be applied to provide protection against synchronization errors involving a range of symbol losses or gains. If the new technique is chosen so that up to $t_f$ symbol losses can be corrected and up to $t_r$ symbol gains can be corrected, then we say that the code to which the technique is applied is a $t_s$-synchronization-error correcting code, where $t_s = t_f + t_r$. If the technique is applied to a $t_a$-additive-error-correcting Bose-Chaudhuri-Hocquenghem code, we call the resulting code a $(t_a, t_s)$-error-correcting code. The resulting code can simultaneously correct synchronization errors and additive errors,
and we refer to such a code as a synchronizable error-correcting code.

Chapter I constitutes an introduction to the synchronization problem, and discusses various aspects of coding theory. The channel model is discussed, examples of synchronization and additive errors are provided, and some previous approaches to the synchronization problem are described.

In Chapter II, those aspects of the theory of additive-error-correcting linear block codes are presented which are necessary to the derivation of the new technique, and an understanding of its application. Since the new synchronization technique is designed for application to Bose-Chaudhuri-Hocquenghem codes, the theory of those codes is summarized, and an example of such a code is given.

Chapter III presents the new technique and proves the fundamental theorem upon which the new technique is based. The procedures for applying the technique are described.

An extension of the Hamming bound is proved in Chapter IV, and it is indicated how the capability of the new technique compares with the bound. A figure is presented showing the form of the upper bound.

Several examples illustrating the new technique are presented in Chapter V, and a table is presented providing the information necessary to apply the new technique to Bose-Chaudhuri-Hocquenghem codes of length less than or equal to 255, in the binary case.

Finally, in Chapter VI, the new technique is compared with previous techniques. Parameter ranges and situations in which the new technique is superior to the other techniques considered are discussed.
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Redundancy of the New Synchronization Technique Corresponding to Specified Synchronization-Error-Correction Capability.</td>
<td>95</td>
</tr>
<tr>
<td>5.1</td>
<td>Minimum Functions in GF($2^m$).</td>
<td>100</td>
</tr>
<tr>
<td>5.2</td>
<td>Nonzero Elements of GF($2^4$), Expressed as Powers of the Root $\gamma = (0,1,0,0)$ of the Minimum Function $m_2(x) = x^2 + x + 1$.</td>
<td>105</td>
</tr>
<tr>
<td>5.3</td>
<td>Codewords of the Subcode $C_S$.</td>
<td>110</td>
</tr>
<tr>
<td>5.4</td>
<td>Codewords of the Translated Code $C_T$.</td>
<td>111</td>
</tr>
<tr>
<td>6.1</td>
<td>Approximate Upper Bound for the Probability, $P(1)$, of False Synchronization Using the New Synchronization Technique (Approximate Probability of Uncorrectable Additive Errors).</td>
<td>135</td>
</tr>
<tr>
<td>6.2</td>
<td>A Lower Bound for the Probability of False Synchronization Using $L$ Arbitrary Synchronization Symbols, for $n = 255$.</td>
<td>138</td>
</tr>
<tr>
<td>6.3</td>
<td>An Approximate Lower Bound for the Probability of False Synchronization Using the Suggested Barker-Sequence Method for Synchronization.</td>
<td>146</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparison of the Redundancy for Synchronization for the Suggested Altered Codes Synchronization Procedure and the New Synchronization Technique.</td>
<td>156</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.1</td>
<td>Detailed Representation of a Binary Communication or Storage System</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Standard Array, or Decoding Table, for a Hamming (7,4) Single-Error-Correcting Code</td>
<td>9</td>
</tr>
<tr>
<td>1.3</td>
<td>Simplified Illustration of the Occurrence of a Synchronization Error.</td>
<td>15</td>
</tr>
<tr>
<td>1.4</td>
<td>Two Methods for Placing Synchronization Symbols.</td>
<td>20</td>
</tr>
<tr>
<td>1.5</td>
<td>The Effect of a Synchronization Error on a Serially-Transmitted Iterated Code</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Bounds on the Proportion, k/n, of Information Symbols for n Large, for a t -Additive-Error-Correcting Code</td>
<td>81</td>
</tr>
<tr>
<td>4.2</td>
<td>Bounds on the Proportion, k/N, of Information Symbols for N Large, for a (t, t')-Error-Correcting Code, if the Receiver Deletes a' s Symbols from Each Received Word</td>
<td>92</td>
</tr>
<tr>
<td>Symbol</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>a,b,c,...</td>
<td>elements of GF(p)</td>
<td></td>
</tr>
<tr>
<td>α,β,γ,...</td>
<td>elements of GF(q=p^m)</td>
<td></td>
</tr>
<tr>
<td>α,α²,α³,...,α²t a</td>
<td>roots of a t-additive-error-correcting BCH code</td>
<td></td>
</tr>
<tr>
<td>b_L</td>
<td>Barker sequence of length L, b_L = (b_1,b_2,...,b_L)</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>coset generator, c = (c_1,c_2,...,c_n)</td>
<td></td>
</tr>
<tr>
<td>c_A</td>
<td>augmented coset generator, c_A = (c_1,c_2,...,c_n,c_1,c_2,...,c_t_a)</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>a code, generally an additive-error-correcting code, always a block code (i.e., a fixed-word-length code)</td>
<td></td>
</tr>
<tr>
<td>C_A</td>
<td>an augmented code</td>
<td></td>
</tr>
<tr>
<td>C_S</td>
<td>a subcode</td>
<td></td>
</tr>
<tr>
<td>C_T</td>
<td>a translated code</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>minimum distance of a code</td>
<td></td>
</tr>
<tr>
<td>deg[m(x)]</td>
<td>degree of m(x)</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>order of an element of a Galois field; also, base of natural logarithms</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>additive-error vector, e = (e_1,e_2,...,e_n)</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>is contained in, as in &quot;x ∈ C&quot;</td>
<td></td>
</tr>
<tr>
<td>f(x)</td>
<td>arbitrary code polynomial, either f(x) = f_0 + f_1x + f_2x^2 + ... + f_{n-1}x^{n-1} or f(x) = f_1 + f_2x + f_3x^2 + ... + f_nx^{n-1}</td>
<td></td>
</tr>
<tr>
<td>g(x)</td>
<td>generator polynomial of a (cyclic) code</td>
<td></td>
</tr>
<tr>
<td>g_S(x) = g(x)m(x)</td>
<td>generator polynomial of a subcode</td>
<td></td>
</tr>
</tbody>
</table>
G                  generator matrix of a (linear) code (k x n, rank k)
G_S                generator matrix of a subcode
GCD(a,b)            greatest common divisor of a and b
GF(p)               Galois field of p elements: 0,1,2,...,p-1 (p prime)
GF(q), GF(q=p^m), GF(p^m)  Galois field of q = p^m elements (p prime, m an integer greater than zero)
H                  parity check matrix of a (linear) code (r x n, rank r)
H^*                matrix whose rows generate the null space of a code (n columns, rank r)
H_S = [H
      H_2]
parity check matrix of a subcode of a code whose parity check matrix is H
H_2                as in H_S, given above
i                  arbitrary integer; index
I                  identity matrix; an ideal
j                  index
k                  number of information symbols in codewords of an additive-error-correcting code, C; also, in Chapter IV, used to denote the number of information symbols in a synchronization-error-correcting code
k_A, k_S, k_T      number of information symbols in codewords of the codes C_A, C_S, and C_T
L                  length of a synchronization sequence
LCM[m_1(x), m_2(x),...,m_u(x)]  least common multiple of m_1(x), m_2(x),..., and m_u(x)
m(x)                minimum function of z = \alpha^s; factor of g_S(x) = g(x)m(x)
m_1(x)              minimum function of \alpha^1
n                  length of codewords of an additive-error-correcting code
n_A, n_S, n_T      length of the codewords of the codes C_A, C_S, and C_T
(n, k) as in "(n, k) code," a block code whose words have length n, and contain k information symbols.

N length of words of an additive-and-synchronization-error-correcting code (i.e., of a synchronizable error-correcting code).

p a prime; also, probability of additive error in a transmitted symbol.

P \(= N - L\); also, probability of more than \( t_a \) additive errors in a received word.

p(1) probability of false synchronization using the new synchronization technique.

p(2) probability of false synchronization using arbitrary synchronization symbols.

p(3) probability of false synchronization using a suggested synchronization method employing Barker sequences.

PG(r-1, q) projective geometry of \( r-1 \) dimensions, based on \( GF(q) \).

q \(=p^m\), as in \( GF(q) \).

r redundancy of an additive-error-correcting code (i.e., number of redundant symbols in codewords of an additive-error-correcting code).

\( r_A, r_S, r_T \) redundancies of the codes \( C_A, C_S, \) and \( C_T \).

t_a maximum number of correctable additive errors in a received word.

\( t_L \) maximum order of correctable left-shift synchronization errors in a received word.

\( t_r \) maximum order of correctable right-shift synchronization errors in a received word.

\( t_s = t_L + t_r \) maximum number of correctable synchronization errors in a received word.

\((t_a, t_s)\) as in "(t, t_a)-error-correcting code," a code that can simultaneously correct up to \( t_a \) additive errors and \( t_s \) synchronization errors in a received word.

u arbitrary integer.

\( u \) received word, \( u = (u_1, u_2, \ldots, u_N) \).
translated received word, $y = u - c_A$

arbitrary code vector

code vector, $x = (x_1, x_2, \ldots, x_n)$

code vector of a subcode

augmented code vector, $x_A = (x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_t)$

translated code vector, $x_T = x_A + c_A$

$x(i)$ \(x(i) = (x_{i+1}, x_{i+2}, \ldots, x_n, x_1, x_2, \ldots, x_i)\), where $x = (x_1, x_2, \ldots, x_n)$, i.e., $x(i)$ is the left cyclic shift of order $i$ of $x$

truncated word

truncated word, with additive errors corrected

truncated word, with additive and synchronization errors corrected

$z = \alpha^S$ \(\text{root of } m(x)\)

arbitrary code vector

$[\cdot]$ \(\text{as in }"[x]"," \text{greatest integer not greater than } x \text{ (x a real number)}\)

$\{\cdot\}$ \(\text{as in }"\{x\}"," \text{least integer not less than } x \text{ (x a real number)}\)
CHAPTER I

INTRODUCTION

1.1. Channel Model

1.1.1. The Channel

Coding theory is concerned with the development of techniques by which information can be transmitted accurately and efficiently over noisy channels. A model of a typical communication channel is shown in Figure 1.1. This dissertation will be concerned with discrete channels, in which sequences of discrete symbols are transmitted. Noise, or disturbance, is assumed to be present in the channel in such a way that two types of errors, to be described later, can occur in the transmitted sequences. The channel consists of a discrete information source (or message source), an encoder, a transmitter, a transmission medium (with a source of noise), a receiver, an error corrector, a decoder, and a destination (or message destination). These components will now be described in some detail.

The source produces a continual sequence of discrete symbols of an input alphabet, and it is desired to transmit these symbols (called information symbols, or message symbols) to the destination. If the symbols are transmitted directly over the channel to the destination, however, it is assumed that (due to noise) the probability of transmission errors is unacceptably high. For this reason, the generated sequence of information symbols is divided into subsequences of length
Figure 1.1. Detailed representation of a binary communication or storage system.
k, and each k-sequence (called a transmitted message, or source message) is transformed, or encoded, in a one-to-one fashion, by the encoder into a longer sequence of length n (called a codeword), which is sent (symbol by symbol) over the channel instead of the k-sequence. The codeword represents, in essence, the k information symbols together with r = n - k redundant symbols. By choosing the correspondence between the information sequences and the codewords in an appropriate fashion, it is possible to correct certain transmission errors which may occur. In this manner, we can transmit with increased accuracy. The problem of finding a correspondence, or code, which enables transmission at the required level of accuracy, and yet is efficient (i.e., the redundancy, r, is small compared to n) is the central problem of coding theory, and is appropriately called the coding problem.

It is noted at this point that the codeword set also is called the code. Further, a code having codeword length n and redundancy r = n - k is called an (n, k) code, or an (n, k) block code (since the codewords are "blocks" of n symbols).

Next in the channel model is the transmitter, which converts the symbols of the encoded sequence (or codeword) into a signal (e.g., a radio signal) which can be sent over the channel medium (e.g., the air). (The transmitter is said to modulate the signal, and is hence called a modulator.) The codeword, being the sequence corresponding to the signal sent by the transmitter, is also called the transmitted word. The receiver detects the signal and converts it into a received sequence of symbols, called the received word. (The receiver is said to demodulate the signal, and is hence also called a demodulator.) If the received word is not identical to the transmitted word, we say
that an error has occurred in transmission. (It is noted that in a real situation, the received signal will always differ somewhat from the transmitted signal; an error occurs when the difference is so great that the received signal is interpreted to correspond to a sequence other than the transmitted word.)

The received word is next examined by the error corrector, which corrects whatever errors the code is capable of correcting. The corrected received word is identical with some codeword, and it is called the received codeword (which may differ from the received word). If a correctable error (i.e., an error that the code is capable of correcting) has occurred, then this received codeword is identical with the transmitted word. Otherwise, the received codeword is some other (incorrect) codeword. The decoder then decodes the received codeword, by identifying the sequence of k information symbols that corresponds to it. This sequence of k symbols, called the received message, is what reaches the destination.

1.1.2. Types of Errors to be Considered

We shall now describe the two types of errors which are permitted in the channel model considered in this dissertation, namely, additive errors and synchronization errors. An additive error (or amplitude error) is said to have occurred if a received symbol differs from its corresponding transmitted symbol. A synchronization error (or sync error, or timing error, or phase error) is said to have occurred if a received sequence of symbols is of different length from the corresponding transmitted sequence (i.e., the received sequence has either fewer or more symbols than does the transmitted sequence). If an
additive error occurs, we say that a symbol was changed in transmission. If a synchronization error occurs, we say that digits were lost or gained (or deleted or inserted, or dropped or added) in transmission. In aerospace telemetry, thermal or electrical disturbances are two sources of the "additive noise," which causes additive errors. The so-called "time noise," or noise resulting in timing errors, arises from such occurrences as variations in the transmission rate, instabilities in the channel medium, and unpredictable Doppler shifts \[23\].

In this dissertation we shall be concerned with the correction of synchronization errors which result in a net change in the length of a transmitted word. If two synchronization errors occur in a given transmitted word, one inserting symbols and one deleting the same number of symbols, there is no net change in the length of the word. Such "compensating" synchronization errors as these shall not here be considered as synchronization errors; any resulting errors in symbols in corresponding positions of the transmitted and received words would in effect be considered to be the result of additive errors.

We shall now describe the two types of errors in further detail.

1.2. Additive Errors

1.2.1. Example of Additive Errors

The following example is given to illustrate the occurrence of an additive error.

Example 1.1. We assume that the input alphabet is the ternary alphabet

\[1\] The number in square brackets refer to the bibliography listed at the end.
consisting of the symbols 0, 1, and 2. Suppose that the sequence (or vector)\(^2\)
\[ x = (021012210) \]
is transmitted over the channel, and that the sequence
\[ y = (021010210) \]
is received. Then we say that an additive error has occurred in the sixth position. We can represent the error by a sequence
\[ e = y - x = (021010210) - (021012210) = (000001000) \]
where the sum or (or difference) of two sequences is given by the sequence whose \(i\)-th member is the sum (or difference), modulo 3, of the \(i\)-th members of the given sequences. The motivation underlying the use of the term "additive error" becomes clear when we write
\[ y = x + e, \]
i.e., the received sequence is the sum of the transmitted sequence and the error sequence (or error vector).

Until recently, developments in discrete block coding theory have been concerned primarily with the correction of additive errors, under the assumption that no synchronization errors occur. In other words, individual symbols may be received in error, but the positions of the received sequences with respect to the transmitted sequences are known, i.e., the received and transmitted sequences are correctly synchronized.

\(^2\)We adopt the customary practice of generally writing a sequence of single digit numbers without commas. A sequence of letters will be written with commas, e.g., \((x_1, x_2, \ldots, x_n)\).
1.2.2. **Correction of Additive Errors**

Correction of the additive errors which occur in a word corresponds to identification of the error vector. In order to do this, it is necessary that codewords differ from each other in a certain number of positions. Let

\[ \mathbf{x} = (x_1, x_2, \ldots, x_n) \]

denote a codeword, or code vector. We define the weight of \( \mathbf{x} \), denoted by \( w(\mathbf{x}) \), as the number of nonzero positions, or coordinates, in \( \mathbf{x} \).

If

\[ \mathbf{y} = (y_1, y_2, \ldots, y_n) \]

is also a codeword, we define the Hamming distance between \( \mathbf{x} \) and \( \mathbf{y} \), denoted by \( d(\mathbf{x}, \mathbf{y}) \), as the number of positions in which \( \mathbf{x} \) and \( \mathbf{y} \) differ.

Clearly we have

\[ d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y}) = w(\mathbf{y} - \mathbf{x}) . \]

It is easy to show [43] that if we want to be able to correct up to \( t_a \) additive errors in a codeword (i.e., any combination of \( t_a \) or less additive errors which occur in a word), then the Hamming distance between any two words of the code must be at least \( d = 2t_a + 1 \).

Such a code is said to be a code with minimum distance \( d \), or a \( d \)-code, or a \( t_a \)-additive-error-correcting code. The minimum weight of the non-null codewords of a code having minimum distance \( d \) is \( d \). For example, if all codewords are a distance of three apart, then any single error can be corrected by decoding the received word into the closest (in the Hamming distance sense) codeword, for the received word will be at Hamming distance one from the correct codeword, and at distance at least two from all other codewords. Such decoding is called minimum-distance decoding, or nearest-neighbor decoding. The error vector is
interpreted to be the difference between the received word and the nearest codeword.

1.2.3. **Additive-Error-Correcting Codes with Desirable Mathematical Properties**

1.2.3.1. **Group Codes**

In order to find codes having a specified minimum distance, it has been necessary to require that the code possess certain mathematical properties. Generally, the code is required to be a group under the operation of addition defined by

\[ x + y = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n) \]

where \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \) are codewords. Such a code is called a group code. Since a group code is a subgroup of the group of all \( n \)-vectors, we can form the cosets of the code. (If \( G \) is a subgroup of a group \( G^* \) and \( c \) is an element of \( G^* \) but not of \( G \), then the set of all elements of the form \( c + x \), where \( x \) is a member of \( G \), is called the coset of \( G \) generated by the element \( c \).) Error correction is generally accomplished by associating each vector of a coset with a particular codeword, in some systematic fashion. (It is noted here that error correction is sometimes called decoding. This use of the term "decoding" should not be confused with transformation of a codeword into a sequence of information symbols.) Generally, the set of all \( n \)-vectors is arranged in a matrix, with codewords in the first row and the cosets in the following rows. A received word in a particular column of the matrix is decoded into the codeword at the head of the column. Such a matrix is called a standard array, a decoding table, or a Slepian table; an example is shown in Figure 1.2. The
<table>
<thead>
<tr>
<th>Codewords</th>
<th>0000000</th>
<th>1000111</th>
<th>0100110</th>
<th>0010101</th>
<th>0001011</th>
<th>1100001</th>
<th>1010010</th>
<th>1001100</th>
<th>0111000</th>
<th>1110100</th>
<th>1101010</th>
<th>1011001</th>
<th>0110011</th>
<th>0101101</th>
<th>0011110</th>
<th>1111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other received words</td>
<td>1000000</td>
<td>0000111</td>
<td>1100110</td>
<td>1010101</td>
<td>1001011</td>
<td>0100001</td>
<td>0010010</td>
<td>0001100</td>
<td>1111000</td>
<td>0110100</td>
<td>0101010</td>
<td>0011001</td>
<td>1110111</td>
<td>0101110</td>
<td>1011011</td>
<td>0111111</td>
</tr>
<tr>
<td>0100000</td>
<td>1100111</td>
<td>0000110</td>
<td>0110101</td>
<td>0101011</td>
<td>1000001</td>
<td>1100101</td>
<td>1000100</td>
<td>0110000</td>
<td>1110010</td>
<td>1101000</td>
<td>1010101</td>
<td>1001100</td>
<td>0111001</td>
<td>0101110</td>
<td>0011110</td>
<td>1101111</td>
</tr>
<tr>
<td>0010000</td>
<td>1010111</td>
<td>0110110</td>
<td>0000101</td>
<td>0011001</td>
<td>1100001</td>
<td>1001001</td>
<td>1010000</td>
<td>0101010</td>
<td>0011010</td>
<td>1100100</td>
<td>1110011</td>
<td>1101100</td>
<td>1011010</td>
<td>0101111</td>
<td>0011110</td>
<td>1101111</td>
</tr>
<tr>
<td>0001000</td>
<td>1001111</td>
<td>0101110</td>
<td>0011101</td>
<td>1110011</td>
<td>1101011</td>
<td>1010011</td>
<td>1000110</td>
<td>0110100</td>
<td>0111010</td>
<td>1110000</td>
<td>1101101</td>
<td>1011100</td>
<td>1001110</td>
<td>0111101</td>
<td>0011110</td>
<td>1101111</td>
</tr>
<tr>
<td>0000100</td>
<td>1000111</td>
<td>0100010</td>
<td>0010001</td>
<td>0001111</td>
<td>1100101</td>
<td>1010101</td>
<td>1001000</td>
<td>0110010</td>
<td>1110101</td>
<td>1101000</td>
<td>1011011</td>
<td>0101001</td>
<td>0011011</td>
<td>0111001</td>
<td>0011101</td>
<td>1111101</td>
</tr>
<tr>
<td>0000010</td>
<td>1000101</td>
<td>0100100</td>
<td>0010111</td>
<td>0001001</td>
<td>1100111</td>
<td>1010000</td>
<td>1001110</td>
<td>0110101</td>
<td>1110111</td>
<td>1101000</td>
<td>1011111</td>
<td>0101101</td>
<td>0011011</td>
<td>0011100</td>
<td>0111110</td>
<td>1111110</td>
</tr>
</tbody>
</table>

Correctable errors

Figure 1.2. Standard array, or decoding table, for a Hamming (7,4) single-error-correcting code.
null vector is placed as the first entry in the first row, and the first entry in each following row is called a coset leader. Each coset is arranged in order corresponding to the order of the codewords. For any received word, the error vector is the coset leader of the coset in which the received word appears. If each coset leader is a vector of minimum weight in its coset, then the decoding table is a minimum distance decoder. (Note, however, that some cosets could have more than one word of the same minimum weight.)

1.2.3.2. Linear Codes

A more restrictive requirement than that of requiring the code to be a group is to require the code to be a subspace of the vector space of vectors having \( n \) coordinates, where the coordinates (code symbols) are elements of a Galois field. Such a code is called a linear code. If \( \mathbf{x} = (x_1, x_2, \ldots, x_n) \) and \( \mathbf{y} = (y_1, y_2, \ldots, y_n) \) are codewords, where the coordinates are elements of a finite field (i.e., Galois field) of prime or prime power order, then we define

\[
\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)
\]

and

\[
\alpha \mathbf{x} = (\alpha x_1, \alpha x_2, \ldots, \alpha x_n),
\]

where \( \alpha \) is an element of the field. Note that in the binary case, where the symbol field is the binary field of integers modulo 2 (i.e., consisting of the elements 0 and 1, called bits, short for binary digits), then a group code is automatically a linear code. This situation does not hold in general (i.e., there are group codes that are not linear codes).

Since a linear code is a vector space, it can be specified by a
set of basis vectors generating the space. A matrix whose rows are
basis vectors of the code space is called a generator matrix of the
code (generally denoted by G). Also, the code can be specified by
the vector space orthogonal to the code space. A matrix whose rows
are basis vectors of the orthogonal space is called a parity check
matrix (generally denoted by H). The distance properties of a linear
code can be expressed in terms of properties of the matrices G and
H, and decoding can be effected by considering certain equations
involving the received word and H. Such procedures will be describ-
ed in greater detail in Chapter II.

1.2.3.3 Cyclic Codes

A particularly important class of linear codes are the cyclic
codes. A cyclic code is a code such that every cyclic permutation of
a codeword is also a codeword. That is, if
\[ x = (x_1, x_2, \ldots, x_n) \]
is a codeword, then
\[ (x_{i+1}, x_{i+2}, \ldots, x_n, x_1, x_2, \ldots, x_i) \]
is also a codeword, for \( i = 1, 2, \ldots, n-1 \). The best of the known classes
of codes for correcting additive errors in discrete channels in which
errors affect symbols independently is a class of cyclic codes discover-
ed by R. C. Bose and D. K. Ray-Chaudhuri [4], [5] and also independen-
tly by A. Hocquenghem [27]. They are known as Bose-Chaudhuri codes,
Bose-Chaudhuri-Hocquenghem codes, or simply as BCH codes. They will
be described in greater detail in Chapter II.

Cyclic codes are particularly easy to implement, as regards elec-
tronic equipment for encoding, and it is for this reason, together with
the fact that good cyclic codes are available, that they are very important. Peterson [43] presents the theory underlying encoding and decoding of cyclic codes, and illustrates a correction procedure for the BCH codes.

For further details on the subjects discussed in Section 1.2, the reader is referred to Peterson [43].

1.3. Synchronization Errors

1.3.1. Example of Synchronization Errors

The following example will illustrate the occurrence of a synchronization error.

**Example 1.2.** Suppose that the sequence of nine symbols

\[ x = (021012210) \]

is transmitted over the channel, and that the sequence of eight symbols

\[ y = (02101210) \]

is received. We then say that a synchronization error of order one has occurred, or more specifically, we can say that a symbol has been lost. (If we had received ten symbols when only nine had been sent, we would say that a symbol had been gained; such an error would also be called a synchronization error of order one.) In general, if \( i \) symbols are lost or gained, we say that a synchronization error of order \( i \) has occurred. The use of the term "synchronization error" for errors which may be considered as losses or gains of symbols is due to the considerations in the following paragraph.

A long sequence of transmitted symbols is in fact composed of a number of codewords, or blocks of symbols. Suppose that the transmitted symbols are sent seriatim, so that a second word begins immediately
after the first word ends. If there are no special symbols or spaces used to indicate word ends, and no special mechanism is used to distinguish word ends, then the receiver must rely on the number of symbols received to partition the received sequence of symbols correctly into words. For example, if the codewords are four symbols in length, then every fourth symbol after the first is taken to be the initial symbol of a new word. This concept is illustrated by the following example.

Example 1.3. Suppose that

\[ ...01|0210|1101|2111|0010|1211|0012|11... \]

is part of a sequence of transmitted symbols, with word beginnings indicated by vertical bars. Suppose that in the second word there occurs a synchronization error such that the third symbol, 0, of that word is lost. Then the sequence

\[ ...01|0210|1112|1110|0101|2110|0121|1... \]

is received, and word marks are placed after every fourth symbol, as indicated. Then all the symbols after the error are partitioned into words incorrectly. That is, the assigned position of each symbol in a word is incorrect. The words after the word containing the error have all been shifted one position to the left (with respect to the receiver's "frame" of word marks), so that the first symbol of every transmitted word now appears as the last symbol of a received word.

We describe the preceding synchronization error as a left-shift error of order one. Similarly, we say that a right-shift error of order \( i \) has occurred if \( i \) symbols are gained instead of lost.

Thus, whenever a synchronization error occurs in a word and the receiver is unaware of the error, the symbols of words following the word in which the error occurred will be incorrectly partitioned into
words by the receiver. This condition is described by saying that word reception is out of synchronization with word transmission. When word partitioning is correctly done, we say that word reception is in synchronization with word transmission, or simply that reception is synchronized.

1.3.2. **Brief Description of the Physical Causes of Synchronization Errors**

A mechanism by which synchronization errors can occur is illustrated in a highly simplified fashion in Figure 1.3. The signal corresponding to the transmitted sequence of binary symbols has for some reason been received in such a fashion that the receiver has received fewer symbols than intended by the transmitter. Synchronization is discussed in books on telemetry, such as Stiltz [53] and Krassner and Michaels [32]. A panel discussion [25] on synchronization considered the importance of synchronization and the various methods for accomplishing it. Loss of synchronization can occur, for example, due to variations in the rate of transmission of the symbols. In addition, in telemetry applications (such as satellite-to-Earth communication), the transmitter is generally moving, sometimes very rapidly, and the effect of Doppler shift on the received symbol rate can be significant.

It should be emphasized here, perhaps, that the synchronization errors considered here involve losses or gains of an integral number of symbols. The synchronization situation with which we are concerned is known in pulse-code-modulation (PCM) applications as frame synchronization [38], [32], or framing [36], [46] (and also as group synchroni-
Figure 1.3. Simplified illustration of the occurrence of a synchronization error. (The signal waveform illustrated is called a non-return-to-zero level (NRZL) waveform.)
zation [46], [1]). In typical telemetry application, a fixed number of binary-digit numbers (of length \(u = 8\), for example) representing physical measurements are grouped together with check symbols to form a codeword. The physically meaningful \(u\)-bit numbers are called words, and the codeword is called a frame. We are concerned with frame synchronization, but since the technique to be developed depends on the nature of the code, we shall use the term "codeword synchronization" instead of the term "frame synchronization". The term "codeword synchronization" is in general not equivalent to the term "word synchronization", which usually refers [32], [53] to synchronization of the words of the frame. The term "word synchronization" may however, refer [23] to codeword synchronization; whichever meaning is intended should be clear from context. We shall always intend "word synchronization" to mean codeword synchronization. (The term "character synchronization" [23] refers to synchronization of fixed-length symbol sequences which represent particular characters (such as alphanumeric characters), rather than simply numerical measurements.) The term "block synchronization" could mean either word synchronization [23] or codeword synchronization [19].

Also of importance in telemetry applications is synchronization of individual symbols (message acquisition), called symbol synchronization or, in the binary case, bit-synchronization [32], [53] or bitrate synchronization [46]. (Loss of word or frame synchronization may, of course, be due to loss of symbol synchronization.) The problem of synchronized reception of each symbol, however, is well-handled by various instrumentation techniques. Codeword synchronization, on the other hand, is generally maintained by adding redundant symbols such
that the transmitted sequences of symbols possess certain observable properties. An approach in aerospace telemetry, for example, is to place one or more bits between every two subsequences of a given length.

For a more detailed discussion of frame synchronization, word synchronization, and bit synchronization in PCM applications, the reader is referred to [32] or to [53]. See Golomb [21] for a recent discussion of synchronization.

1.3.3. The Hereditary Nature of Synchronization Errors

Correct partitioning of symbols into words is essential to proper interpretation of the received message, and hence, since a synchronization error in one word can cause other words to be interpreted incorrectly, the occurrence of undetected synchronization errors is more serious than the occurrence of undetected additive errors, which affect only the words in which they occur. Correct additive error correction itself is dependent upon proper synchronization.

It is noted that if a synchronization error occurs in a cyclic code, it is very likely that, even if the receiver watched for an apparent increase in the observed additive-error rate (as is expected if a synchronization error occurs), the synchronization error might not be noticed for quite a while, particularly if it is of small order. This is due to the fact that, since cyclic shifts of codewords are also codewords, many overlaps (i.e., incorrectly synchronization received words) are likely to be identical or similar to codewords.

We shall here be concerned with the correction of synchronization to the extent that the number of symbol losses or the number of symbol gains is identified. Such correction corrects the erroneous synchro-
nization of the words following the word in error, but does not correct (i.e., reconstruct) the word containing the error by identifying where to insert or delete symbols in that word. Thus we might more precisely say that we are concerned with the correction of lost synchronization which has resulted from symbol losses or gains, but not with actual correction of the symbol loss or gain itself.

1.4. Previous Approaches to Synchronization-Error Correction

1.4.1. The Use of Special Synchronizing Symbols

In practice, the methods employed to cope with synchronization errors are at present rather straightforward, and are not generally based on sophisticated mathematical considerations. A straightforward approach to synchronization is to employ a special synchronizing symbol, which differs from the symbols of the codewords, to distinguish the beginnings of words. Examples [19] of this approach are the Morse code letter space and the teletype start and stop pulses. In telemetry, it is desirable to restrict transmission to sending only the symbols which make up codewords. For example, in the binary case, only 0's and 1's are sent, and no third symbol is used as a synchronizing symbol. In this dissertation, we shall be concerned with synchronization for this type of transmission, and will assume that all transmitted symbols can appear anywhere in codewords.

1.4.2. Interlaced Synchronization Symbols and Synchronization Sequences

If the only symbols which are sent over the channel are the code symbols (i.e., special synchronizing symbols differing from the code are not allowed), then the receiver must depend upon the received symbols themselves to determine synchronization.
Two procedures of this type are the following:

1. the interlaced synchronization symbols method \([46]\), in which intermittent (fixed) positions in each word contain fixed symbols; and

2. the synchronization sequence method, in which a fixed sequence of given length precedes each codeword.

These two procedures are illustrated in Figure 1.4.

With the interlaced symbols method, the receiver simply checks every received word to make sure that the designated symbols appear as they should. If synchronization is lost, other symbols will appear in the specified positions, and the receiver resynchronizes the words so that the proper symbols again match.

For the synchronization sequence method, sequences (called Barker \([1]\) sequences) are available that possess the property that the correlation of the Barker sequence with overlaps of the Barker sequence with the null sequence (and hence with random sequences) is small. Because of this property, it is very unlikely that the Barker sequence will be observed slightly too early or slightly too late. For example, if the sequence 1111111 is used as a synchronization sequence, the probability is 1/2 (assuming random bits) that it will be preceded by a 1, resulting in the occurrence of the synchronization sequence. If, however, the Barker sequence 1110010 is used, the probability is very small that the preceding or following six or fewer bits will combine with additive errors to produce the Barker sequence. If the Barker sequence is of length \(L\), then for each \(L\)-tuple preceding each successive received symbol, the receiver calculates the correlation of the Barker sequence with the \(L\)-tuple. It is small with high proba-
Figure 1.4. Two methods for placing synchronization symbols.
bility immediately before and after the synchronization sequence and maximum when the Barker sequence is received. The receiver thus obtains a reliable and easily observable indication of the correct synchronization position, whenever the Barker sequence occurs. (Note that the observation of any specified sequence of L symbols in a random sequence of symbols depends only on the length L, and not on the arrangement of symbols comprising the given sequence. Barker sequences provide additional protection, however, against synchronization errors within L-1 digits of the correct position.)

The preceding methods of synchronization have been introduced into practice because of their simplicity and ease of implementation, with little theoretical consideration of the degree of protection obtained versus the amount of redundancy present for synchronization purposes. If synchronization sequences or synchronization symbols are periodically inserted in the message, the protection varies according to the number of symbols used for synchronization purposes per codeword. However, it does not appear that methods (such as those above) which do not take into account the nature of the individual codewords would be as efficient as a procedure that takes the words into account. (See [25] for a generalization of Barker sequences.)

1.4.3. A Synchronization Procedure for the Continuous Channel

In connection with the synchronization sequences discussed above, it is of interest to note a certain synchronization technique associated with the continuous channel. Using this approach, the transmitter repeatedly and continually transmits a fixed sequence simultaneously with the data sequence, but in a second channel [51]. The data and the synchronization sequences are synchronized, of course,
since they are sent at the same time over the same channel medium. The receiver maintains synchronization by determining the phase and starting position of the synchronization sequence (which is easily done if the synchronization sequence is properly chosen); since the two channels are synchronized, the receiver knows the synchronization of the data sequence. The best synchronization sequences to use for this method are called pseudo-noise sequences [22], [51], and they possess the property that the correlation between the sequence and cyclic permutations of itself is minimized. For recent work dealing with correlation properties of sequences, the reader is referred to [33] and [54].

1.4.4. Sellers' Bit Loss and Gain Correction Technique for Burst-Error-Correcting Codes

With the use of inserted synchronization sequences, synchronization is reestablished after a synchronization error as soon as the receiver correctly recognizes a synchronization sequence. The method does not indicate where symbols should be inserted or removed to correct the erroneous word. Sellers [47] devised a method in which a periodic synchronization sequence is used in conjunction with an additive-burst-error-correcting code, enabling correction of specific symbol losses or gains. (See Peterson [43] for the definition of a burst-error-correcting code.) The redundancy of the method is considerable, however. To correct a bit loss or gain of \( l \) bits requires

\[ j + k(2l + 1)/j \]

redundant bits, where \( k \) is the number of information bits, and \( j \) is the size of the burst that the burst-error-correcting code must be able to correct. The value of \( j \) that minimizes the redundancy is
\[ j = \sqrt[3]{k(2^k+1)} \]

Sellers' method can be used without a burst-error-correcting code, and in that case the lost synchronization can be corrected but the specific bit loss or gain cannot be determined.

Sellers' method will now be briefly described. (In Sellers' paper, the symbols of received sequences are written right to left in order of occurrence, instead of left to right, as in this dissertation. As a result, the following sequences will be reverses of those found in Sellers' paper.) To illustrate the method, suppose that \( j = 1 \), i.e., we wish to correct gains or losses of a single bit. We insert the sequence 100 after every \( j \) symbols of a codeword, and the receiver examines every triplet of received symbols that occurs after every \( j \) intervening symbols. If no bit loss or gain occurs, each triplet is 100. We assume that no additive errors occur in a triplet. If a bit is lost between triplets, the next triplet will be either 000 or 001, and if a bit is gained, the next triplet will be either 010 or 110. If a bit is lost in the triplet, either 101, 100, 000, or 001 occurs. If a bit is gained in the triplet, either 010, 110, 100, or 101 occurs. The sequences 011 and 111 cannot occur. Note that 100 and 101 can occur as the result of certain bit losses or gains in the triplet.

The receiver observes the received triplets, and decides on an appropriate course of action. If 101 occurs, then there is either a bit loss or gain in the triplet, and the next triplet will indicate the error. If 100 occurs, there either is no error or there is an error in the triplet, and the next triplet will indicate the error. If 010 or 110 occurs, the receiver interprets that a bit has been gained,
and a bit is arbitrarily removed from the preceding sequence of \( j + 1 \) symbols. If 001 or 000 occurs, the receiver interprets that a bit has been lost, and a bit is arbitrarily inserted in the preceding sequence of \( j - 1 \) symbols. The synchronization error has thus been corrected. However, if the corrective bit was inserted or deleted at the wrong place in the sequence preceding the triplet, many or even all of the bits of the sequence may be in error, since the corrective action may have in fact resulted in a "compensating" synchronization error. Since the code is a \( j \)-burst-error-correcting code, however, the sequence can be correctly reconstructed.

The code can also correct a burst of additive errors, provided that the bit loss or gain, the corrective action, and the additive errors all occur within a span of \( j \) bits, not including a triplet.

1.4.5. **Comma-Free Codes**

An alternative approach to synchronization has been to determine codes whose words indicate by themselves whether or not synchronization is being maintained. Work in this area has included the study of comma-free codes, i.e., codes in which no overlaps of any two codewords are in turn codewords. (See [10], [24], and [28] for general theory regarding comma-free codes.) An overlap of order \( i \) is any vector of the form \((x_{i+1}, x_{i+2}, \ldots, x_n, y_1, y_2, \ldots, y_1)\) or \((x_{n-i+1}, x_{n-i+2}, \ldots, x_n, y_1, y_2, \ldots, y_{n-i})\), where \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \) are any two codewords. With such a code, a synchronization error of any order can be detected and corrected with the first complete word immediately following the error, provided that no additive errors occur. This property holds since incorrectly synchronized words will be overlaps of codewords and will hence differ from codewords. Also considered
have been comma-free codes of index u, i.e., codes in which word overlaps differ from codewords in at least u positions [28]. In this case, synchronization errors can be corrected in the presence of additive errors.

Eastman [12] developed a construction which produces, for any code whose words are of odd length n and whose symbol alphabet is of size q, comma-free codes having a maximal number of words.

Two disadvantages of the first comma-free codes are that their additive-error-correcting properties are either nonexistent or unknown, and they do not possess desirable algebraic properties (such as forming groups or group cosets) which would allow efficient encoding and decoding. It is observed that a group code cannot be comma-free of any order, since it contains the null vector consisting of n zeros, and any overlap of that vector with itself is also the null vector.

Kendall and Reed [31] have investigated a subclass of comma-free codes which have a property called path-invariance. For path-invariant comma-free codes, division of an unsynchronized sequence of codewords into codewords is easier than for general comma-free codes. The number of words in a path-invariant comma-free code (having q input alphabet symbols and wordlength n) is \( (q-1)^{[n/2]} \cdot q^{([n-1]/2)} \) if \( n \geq 4q/3 \) (where \([x]\) denotes the greatest integer not greater than \(x\)). (See [52] for additional recent work in comma-free codes.)

\[3\] The symbol \([x]\) shall be used in all that follows to denote the greatest integer not greater than \(x\).
1.4.6. Stiffler's Coset Codes

Recently, Stiffler [50] has investigated the synchronization properties of certain codes which are suitably chosen cosets of group codes. These codes are as easy to use as the original group code from which they are derived, and of course possess all of the additive-error-correcting ability of the parent code, since the distance properties of a coset and the group are identical. A coset code can correct synchronization errors of order up to $t_s$ if overlaps of order $t_s$ or less are not coset words (i.e., overlaps differ from coset words in at least one position). Stiffler has derived general conditions which must be satisfied if the coset is to be able to correct synchronization errors of a given order. His conditions insure that coset word overlaps differ from coset words in at least one position. If additive errors can occur, it will in general not be known whether the errors are simply additive errors or indicate the occurrence of a synchronization error. To decide whether a synchronization error has occurred or not, a trial-and-error procedure must be employed if additive errors are allowed. Under this procedure, a sequence of received words is examined and the (periodic) word mark pattern is placed in the sequence, at various starting positions. The apparent additive-error rate (i.e., the average number of corrected additive errors) is expected to be greater for incorrect placement of the word marks (i.e., for incorrect synchronization) than for correct placement, and thus that position is chosen for synchronization for which the additive-error rate is least. Generally, Stiffler's conditions for synchronizability of a coset are difficult to check, and he presents only simple examples illustrating the method.
Stiffler defines a code to be invulnerable to synchronization at position \( i \) if the sequence \( (x_{i+1}, x_{i+2}, \ldots, x_n, y_1, y_2, \ldots, y_i) \) is not a codeword, where \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \) are codewords (not necessarily different). Using this terminology, we would say that a code can correct synchronization errors of order \( t \) if it is invulnerable to synchronization at all positions \( i \) satisfying \( i \leq t \) or \( i \geq n - t \). Stiffler's conditions are given in the following theorem, which is taken from [50].

If an \((n,k)\) group code has a coset which is invulnerable to synchronization at the position \( m \), its nullspace \( H \) must contain three vectors \( h, h^1, \) and \( h^2 \) such that

\[
h^1_i = \begin{cases} h_{i+n-m}, & i = 1, 2, \ldots, m \\ 0, & i = m+1, \ldots, n \end{cases}
\]

and

\[
h^2_i = \begin{cases} 0, & i = 1, 2, \ldots, m \\ h_{i-m}, & i = m+1, \ldots, n \end{cases}
\]

where \( h^j_i = (h^j_1, h^j_2, \ldots, h^j_n) \), for \( j = 1 \) and \( 2 \).

If the preceding condition is satisfied for some vector \( h \), all cosets of group for which

\[
\sum_{i=1}^{n} h^j_i b_i = 1
\]

are invulnerable to synchronization at position \( m \). The vector \( b \)

is defined by \( b = (c_1, c_2, \ldots, c_n) + (c_{m+1}, c_{m+2}, \ldots, c_n, c_1, c_2, \ldots, c_n) \), where \( c = (c_1, c_2, \ldots, c_n) \) is a generator of the coset.

Stiffler also proves the following results, which Tong [55] proves in a different manner.

Any \((n,k)\) binary cyclic code can be made invulnerable to synchronization at all positions \( u \) satisfying \( k+1 \leq u \) or \( u \leq n-(k+1) \) by simply complementing (i.e., changing zeros to ones and ones to zeros)
the first coordinate of each codeword.

All cosets of any \( (n,k) \) binary cyclic code are vulnerable to synchronization at all positions \( u \) satisfying \( n-k \leq u \) and \( u \leq k \).

If \( k < (n-1)/2 \), any \( (n,k) \) binary cyclic code can be made comma-free. If \( k \geq (n-1)/2 \) no coset of any \( (n,k) \) cyclic code is comma-free.

Frey [16] has described implementation of coset codes, showing how (if no additive errors occur) a synchronization error can be detected and the correct synchronization position located by observing \( n \) successive \( n \)-sequences (where, as usual, \( n \) denotes word length).

1.4.7. **Levy's "Altered" Cyclic Codes**

Levy [34] has considered a technique which can be considered to be an extension of Stiffler's use of cosets. He considered cyclic codes, which can be specified in terms of a generator polynomial, \( g(x) \), of an ideal of polynomials modulo the polynomial \( x^n-1 \), with coefficients in a finite field of order two (i.e., binary coefficients). (Such specification of cyclic codes will be described in Chapter II.) If \( a(x) \) is a polynomial of degree \( \leq k-1 \) whose coefficients are information symbols, then \( a(x)g(x) \) is the corresponding codeword. If it is desired to transmit \( a(x)g(x) \), then \( v(x) = a(x)g(x) + r(x) \) is sent instead, where \( r(x) \) is a suitably chosen fixed polynomial that is not a code polynomial.

Levy calls the code whose vectors are of the form \( v(x) \) an altered code. If no synchronization error occurs, the receiver receives \( v(x) + e(x) \), where \( e(x) \) is the polynomial whose coefficients represent the additive error pattern. If there has been a synchronization error the receiver receives a polynomial which can be written as
\[ x^t v(x) + u_t(x) + e(x), \]
in which the degree of \( u_t(x) \) is less than \( t \), where \( t \leq \lceil n/2 \rceil \) is the order of the synchronization error which has occurred. If we subtract \( r(x) \) from the received polynomial, we obtain
\[ y(x) = x^t a(x) g(x) + (x^t - 1) r(x) + u_t(x) + e(x). \]
Now the quantity \( x^t a(x) g(x) \) is a code polynomial, since the code is cyclic, and the above expression will therefore differ from any code polynomial whenever
\[ (x^t + 1) r(x) + u_t(x) + e(x) \]
is not equal to a code polynomial (note that \( +1 = -1 \), since we are working over the binary field). If this condition holds for all \( e(x) \) having no more than \( s \) nonzero coefficients (i.e., no more than \( s \) additive errors can occur in any sequence of \( n \) symbols) and for all \( t \leq s \) or \( t \geq n - s \), then Levy says that such a code possesses the "slip-detecting characteristic \([s, \delta]\)." Although Levy does not propose an explicit procedure by which a code with slip-detecting characteristic \([s, \delta]\) can be used to detect or correct synchronization error, he states that there exists the possibility of implementation to restore synchronization during an interval relatively free from additive errors.

Levy gives a necessary condition that a code with slip-detecting characteristic \([s, \delta]\) must satisfy: the minimum weight of that part of \( r(x)(x^t + 1) \) consisting of terms of degree not less than \( t \) should be at least \( \delta \) for all \( t \leq s \). He obtains the following sufficient condition:

Let there be an \( r(x) \) of degree \( z \) such that the weight of that part of \( r(x)(x^t + 1) \) of degree greater than or equal to \( t \), for \( t = 1, 2, \ldots, s \), is
at least 8. If such an \( r(x) \) is added to each word of a cyclic code of minimum weight \( d \), and if \( d-(s+z+1) \geq 8 \), then the resultant altered code will have the slip-detecting characteristic \([s, 8]\).

1.4.8. Tong's Development of Altered Codes

Tong [55] also considered the use of altered codes for synchronization, and developed procedures for correcting synchronization errors and additive errors using the altered-codes technique. Tong's approach does not always require additional check symbols for synchronization purposes, beyond those included for additive-error protection. He uses the technique with shortened codes (see [43] for the definition of a shortened code), taking advantage of unused bits to determine the synchronization error. In addition, he shows how to correct a synchronization error consisting of a single bit loss or gain, using a Bose-Chaudhuri-Hocquenghem code.

A disadvantage of Tong's procedure for simultaneous correction of additive and synchronization errors is that, generally, if it is desired to correct synchronization errors of order up to \( i \) in either direction, then the receiver must sacrifice the ability to correct \( 2i \) additive errors. For simultaneous detection of synchronization errors and correction of additive errors, the capability for detecting \( i + 1 \) additive errors must generally be sacrificed.

1.4.9. The Use of An Iterated Code for Synchronization Control

Maxwell and Kutz [38] propose an interesting technique for synchronization-error correction based on the use of an iterated code (see [43] for the definition of an iterated code). Consider the example illustrated in Figure 1.5, where (in a correctly synchronized
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$r_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_8$</td>
<td>$x_9$</td>
<td>$x_{10}$</td>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
<td>$x_{14}$</td>
<td>$r_2$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_3$</td>
<td>$c_4$</td>
<td>$c_5$</td>
<td>$c_6$</td>
<td>$c_7$</td>
<td>$r_3$</td>
</tr>
</tbody>
</table>

Properly synchronized frame, corresponding to the transmitted sequence

$x_1 x_2 x_3 x_4 x_5 x_6 x_7 r_1 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} r_2 c_1 c_2 c_3 c_4 c_5 c_6 c_7 r_3$

(note that we could denote $r_3$ as $c_8$).

<table>
<thead>
<tr>
<th>$r_3^*$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$x_8$</td>
<td>$x_9$</td>
<td>$x_{10}$</td>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
<td>$x_{14}$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_3$</td>
<td>$c_4$</td>
<td>$c_5$</td>
<td>$c_6$</td>
<td>$c_7$</td>
</tr>
</tbody>
</table>

Improperly synchronized received frame, corresponding to the received sequence

$r_3^* x_1 x_2 x_3 x_4 x_5 x_6 x_7 r_1 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} r_2 c_1 c_2 c_3 c_4 c_5 c_6 c_7 r_3$

where $r_3^*$ is the last symbol of the preceding frame.

Figure 1.5. The effect of a synchronization error on a serially-transmitted iterated code.
frame) $c_i$ denotes the parity check for the $i$-th column, and $r_j$ denotes the parity check for the $j$-th row. Ordinarily, the parity checks are used to correct additive errors, in the usual fashion. If a synchronization error occurs, note that the row parity checks may be affected, but that all column parity checks, except the check for the first (received) column, are unaffected.
Such a situation results in unusual behavior of the numbers of row and column parity checks in successive words, and the receiver eventually decides that a synchronization error has occurred. Synchronization is reestablished at the position corresponding to proper behavior of the row and column parity checks. Whenever synchronization is correct, the parity checks are used to correct additive errors. Maxwell and Kutz performed a computer simulation of the procedure, and show probabilities of correct synchronization as a function of additive-error probability and the number of observed frames.

1.4.10. Other Work in Synchronization Error Correction

A probabilistic approach to synchronization, using signal-flow graph techniques, has been investigated by Bender [2]. Gallager [17] has considered sequential coding for binary channels with additive and synchronization errors. (See [59] for the theory of sequential coding.) Other work in the field of synchronization-error correction is included in [11], [13], [18], [19], [29], [30], [35], [41], [44], [56], [57], and [58].

1.4.11. A Note Concerning Variable-Length Codes, and Synchronization Terminology

All the preceding work cited above, except the sequential coding, has been concerned with block codes, i.e., codes having fixed-length words. Synchronization is at least as essential to the proper reception of codes having variable-length words as it is for block codes, and this problem has received attention also (see [6], [7], [20], [39], [40], [42], and [45]). (Calabi [7] includes an extensive bibliography on variable-length codes.) A comment is in order regarding the
terminology employed in both the variable-length and fixed-length cases. Variable-length codes that can correct synchronization errors are called self-synchronizing [20], error-limiting [40], or self-
resynchronizing [42]. Fixed-length codes that can resynchronize (i.e., that can correct synchronization errors) are said to be synchronizable [12], [13], self-synchronizing [23], [34], [51], or invulnerable to synchronization at (specified) incorrect positions [50]. We use the term "synchronizable" in this dissertation (cf. Eastman [12]). (Due to the frequent use of the word "resynchronize" the descriptor "resyn-
chronizing" also has merit. The term 'synchronizable' seems more ap-
propriate for the codes introduced here, since resynchronization is de-
pendent upon a certain amount of analysis and subsequent action by the receiver; synchronization is not automatically maintained by simple observation (as by a pattern recognizer.) By the term "synchroniza-
ble error-correcting code" is meant "synchronizable additive-error-
correcting code", or, more specifically,"synchronization-error-and-add-
itive-error-correcting code." Fixed-length codes such that no overlaps of codewords are codewords are called comma-free. Variable-length codes such that a sequence of symbols formed by a string of codewords can always be broken into codewords correctly are called separable,
decodable, or uniquely decipherable.

1.5. The Synchronization Technique Introduced in This Dissertation

This dissertation will present a new approach to the codeword synchronization problem for cyclic additive-error-correcting codes.

A method is developed which is capable of correcting synchronization

---

4 In recent (1965) research in variable-length codes published by Parke Mathematical Laboratories, Calabi and Myrvaagres use the term "synchronizable."
errors immediately after their occurrence, and additive errors as well, even in the same word in which the synchronization error is corrected. The procedure consists of two stages: first the additive errors are corrected, and then the synchronization error is corrected. Bounds are derived which indicate that the procedure is an efficient one for correcting small synchronization errors. The method consists in repeating at the end of each codeword a certain number of the initial symbols of the word and adding a particular sequence to this extended word. The receiver subtracts this sequence from each received word, and on the basis of this new word corrects any additive errors that have occurred; the receiver then corrects whatever synchronization error has occurred (provided, of course, that the errors are such that the code has been designed to correct). There is no confusion as to whether apparent additive errors represent synchronization errors or true additive errors. It is shown how the synchronization-error-correcting capability can be sacrificed for some words in order to correct additive errors in addition to those for which the code would ordinarily be used.

The technique is similar in one respect to the procedure considered by Levy and Tong, in that transmitted words are members of a certain coset of a group code, and the receiver subtracts the coset generator from the received words. The construction of the group code, the method of choosing the coset, and the procedure for error correction constitute a fundamentally new contribution to the field of synchronization-error correction.
CHAPTER II

BASIC CONCEPTS IN THE THEORY OF LINEAR BLOCK CODES

2.1. Introduction

The development of the new synchronization-error-correction technique introduced here depends rather heavily on a number of concepts in the theory of linear block codes. Furthermore, effective use of the technique is dependent on an understanding of certain results of linear coding theory. The results of coding theory that are necessary for both the preceding purposes will be presented in this chapter. A straightforward development of the method in the next chapter will thus be facilitated, and the results necessary to apply the technique will be conveniently available in useful form. At the risk of excessive redundancy, some of the concepts already introduced in Chapter I will be repeated in order to enable a continuous exposition.

2.2. Preliminary Concepts

It is assumed that the symbols of the input alphabet are the elements of a finite field, or Galois field, of prime or prime power order, \( q = p^m \), where \( p \) is prime. (See [8] for a presentation of Galois field theory.) We denote the Galois field of order \( q = p^m \) by \( \text{GF}(q) \), or by \( \text{GF}(q = p^m) \). We denote the elements of \( \text{GF}(p) \) by 0, 1, 2, \ldots, \( p-1 \), or by lower case letters, such as \( a, b, c \). The symbol \( \alpha \) will often be used to denote an element of \( \text{GF}(q) \). In the theory of block, or fixed-
length, codes, the general procedure is to associate a given sequence
of length \( k \) with a particular sequence of length \( n > k \). In an encoded
sequence, we say that there are \( k \) information symbols and \( r = n - k \) re-
dundant symbols. The ratio \( k/n \) is sometimes referred to as the transmis-
sion rate (if the resource symbols are equiprobable). The set of \( n \-
sequences (called code vectors, or codewords) is called a codeword set
and the correspondence is called a code, or more specifically, an \((n,k)\)
code. The codeword set itself is also called a code. The particular
\( n \)-sequence corresponding to a given \( k \)-sequence generated by the source
is transmitted over the channel. Because of the possibility that
errors occur in transmission, the received sequence can be any one of
the \( q^n \) possible \( n \) sequences. Since there

are \( q^k \) sequences of length \( k \) and \( q^n > q^k \) sequences of length \( n \), the
decoding problem consists conceptually in partitioning the set of \( q^n \)
sequences into \( q^k \) subsets such that, in decoding, all the \( n \)-sequences
in a given subset are associated with a particular \( k \)-sequence. In
the usual situation there are \( q^n/q^k = q^{n-k} = q^r \) words in each partition
set, so that the \( q^n \) words can be tabulated with the \( q^k \) words at the
heads of columns, each with \( q^r \) entries. If a particular \( n \)-sequence is
received, it is interpreted as the word at the top of the column in
which it appears. An example of a code thus constructed is given in
Figure 1.2, where we are using the binary alphabet \((q=2)\) consisting of
elements 0 and 1.

The code has \( n=7 \) and \( k=4 \), and is thus a \((7,4)\) code. It is called
a Hamming code, and will be discussed in further detail later. If we
assume that errors affect the symbols independently and identically,
the method of decoding used above is called maximum-likelihood decoding, since a received word is decoded into that codeword having the greatest conditional probability that it was the transmitted word, given the word received.

The general procedure in block code construction is to construct codeword sets that are capable of correcting a certain number of errors per received word. We recall the definition of Hamming distance given in Chapter I. For a code to be able to correct any combination of \( t_a \) errors or less, it is necessary and sufficient that the codewords be at least a distance \( d=2t_a+1 \) from each other. When the words are at least distance \( d \) from each other, a received word with \( t_a \) errors will still be closer to the transmitted word than to any other codeword. Hence, if we decode a received word into the nearest codeword, we will decode correctly so long as no more than \( t_a \) errors occur.

A code in which the codewords are at least distance \( d \) from each other is said to be a code of distance \( d \), or a \( d \)-code. (It is easy to see that a code is of distance \( d \) if and only if the minimum weight of its nonnull words is \( d \).) Thus the problem of constructing a \( t_a \)-additive-error-correcting code is that of finding a \( d \)-code with \( d=2t_a+1 \). Information words (of \( k \) symbols) are then associated in a one-to-one fashion with the codewords (of \( n \) symbols). From a practical point of view, the problem of associating the source messages with the words of the codeword set is as important as the problem of constructing the codeword set. Coding theory is concerned not only with finding the codeword set, but also with determining the correspondence with source messages in such a way that efficient implementation is possible.
2.3. Linear Codes

The most fruitful codes yet considered are linear codes, in which the codeword set is a subspace of the vector space of all n-tuples under the operations of vector addition, defined by

\[ \mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n), \]

and scalar multiplication, defined by

\[ a \mathbf{x} = (a x_1, a x_2, \ldots, a x_n), \]

where the indicated additions and multiplications are modulo p. It is noted that a set of vectors over GF(2) which is a group is also a subspace. Thus a binary group code is a linear code; that a linear code is a group code is obviously true. It is common terminology to call a binary linear code a group code. (For general p, a linear code is a group, but a group of vectors over GF(q) is not necessarily a linear code.)

A code C is specified by a basis, or linearly independent set of vectors. The basis may be written in matrix form:

\[ G = \begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{k1} & a_{k2} & \cdots & a_{kn}
\end{bmatrix}, \]

where the rows of G are the basis vectors and G is of full rank, k. The matrix G is called the generator matrix of the code C. If \( \mathbf{u} = (u_1, u_2, \ldots, u_k) \) is a k-tuple of information symbols, an easy method of encoding consists simply in encoding \( \mathbf{u} \) into the codeword \( \mathbf{y} G \).

Alternatively, C is specified by an \((n-k) \times n\) matrix H, of full rank \( n - k = r \), which is orthogonal to G. The matrix H is called the parity check matrix of C, and every code vector \( \mathbf{x} \) must satisfy
\[ x H' = 0 \]

where \( 0 \) denotes the \( r \)-coordinate null vector.

The space of all vectors orthogonal to the space generated by the rows of a given matrix is called the null space of the matrix. Thus a code is the null space of its parity check matrix. (The code \( C_D \) generated by a parity check matrix of a given code, \( C \), is the null space of the generator of \( C \). The codes \( C \) and \( C_D \) are called dual codes. A parity check matrix, by definition, is of full rank, i.e., it contains no dependent rows. Its rows are thus a basis of the null space of the code, rather than simply a generating set. A parity check matrix, \( H \), for a given code can be obtained from any given matrix, \( H^* \), whose rows generate the null space of the code, by deleting suitable rows of \( H^* \) in order that the rows of the remaining matrix, \( H \), are a basis of the null space. (Both matrices are of the same rank, of course.)

It is easy to prove that \( C \) is of distance \( d \) if and only if every combination of \( d-1 \) or fewer columns of \( H \) are linearly independent. Thus the problem of constructing a \( d \)-code is equivalent to finding an \( r \times n \) matrix \( H \) of rank \( r \) such that no \( d-1 \) or fewer columns are linearly dependent. If we assume that the code symbols are elements of \( GF(q=p^m) \), where \( p \) is prime, then we must find a set of points in \( PG(r-1, q) \) such that no \( d-1 \) are dependent. To keep the redundancy, \( r \), as small as possible, so that the ratio \( (n-r)/n \) of information symbols to transmitted symbols is as large as possible, we wish such sets of points to be large. (The problem of finding the largest such set for specified \( r \) is called the packing problem.)

The generator matrix, \( G \), of the code shown in Figure 1.2 is given by
\[ G = [I_k, P] = \begin{bmatrix} 1000111 \\ 0100110 \\ 0010101 \\ 0001011 \end{bmatrix}, \]

and the parity check matrix, \( H \), is given by
\[ H = \begin{bmatrix} 1110100 \\ 1101010 \\ 0011001 \end{bmatrix} = [-P', I_3]. \]

Two codes that differ only in the arrangement of the symbols of the codewords are said to be equivalent. Thus the matrix \( H \) given above is thus equivalent to the matrix \( H_1 \) given by
\[ H_1 = \begin{bmatrix} 0001111 \\ 0110011 \\ 1010101 \end{bmatrix} \]

The matrices \( H \) and \( G \) above are said to be in canonical form, due to the location of the identity matrix in the right-hand part of \( H \) and the left-hand part of \( G \). Every generator matrix can be reduced to canonical form by elementary row operations and column permutations, and a generator matrix formed by row operations on a given generator matrix of course generates the same code as the original matrix. If \( G \) is in canonical form, and the \( k \)-tuple \( u = (u_1, u_2, \ldots, u_k) \) is encoded as \( x = u G \), then the information symbols \( u_1, u_2, \ldots, u_k \) are the first \( k \) coordinates of \( x \), and the last \( r \) coordinates are parity check symbols.

Using the matrix \( H \) given above, it is easy to correct single errors. For example, suppose that the codeword
\[ x = (x_1, x_2, \ldots, x_7) \]
was sent, and that the error

\( e = (0000100) \)

occurred, so that the word

\[ y = x + e \]

is received. If we now calculate the quantity

\[ y H' = (x + e)H' \]
\[ = x H' + e H' \]
\[ = e H' \]

we obtain

\[ y H' = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \]

which is the fifth column of \( H \). Thus we interpret that the error occurred in the fifth position of \( x \). Since the alphabet is binary, we can correct the error simply by complementing the fifth coordinate of the received vector \( y \).

In general, if \( y = x + e \) is a received vector, then the vector \( y H' \) is called the syndrome of \( y \), and we have the relation

\[ y H' = x H' + e H' \]
\[ = e H'. \]

The syndrome \( e H' \) is different for each of the correctable error patterns \( e \), and is the basis for correcting the error \( e \).

2.4. Cyclic Codes

2.4.1. Polynomial Representation of Codewords

A very important class of linear codes is the class of cyclic codes. A linear code is said to be cyclic if, whenever \( x = (x_1, x_2, \ldots, x_n) \) is a codeword, then \( (x_{i+1}, x_{i+2}, \ldots, x_n, x_1, x_2, \ldots, x_i) \)
is also a codeword, for \( i = 1, 2, \ldots, n - 1 \). In the study of cyclic codes, it is convenient to consider an \( n \)-tuple as an element of the algebra of polynomials modulo \( x^n - 1 \). (This can be done for any linear code, whether it is cyclic or not.) For completeness, this algebra will be briefly described. It is noted that the terms "\( n \)-tuple" and "vector" (or more precisely, "vector of \( n \) components") are used interchangeably.

If \( \mathbf{a} = (a_1, a_2, \ldots, a_n) \) is an \( n \)-tuple, then the corresponding polynomial is

\[
a(x) = a_1 + a_2 x + a_3 x^2 + \ldots + a_n x^{n-1}.
\]

The polynomial \( f(x) \) is in the same equivalence class as \( g(x) \) if there exists a polynomial \( q(x) \) such that

\[
f(x) - g(x) = q(x)(x^n - 1).
\]

Equivalence classes are called residue classes. If \( f(x) \) is a polynomial, then the residue class containing \( f(x) \) is denoted by \([f(x)]\).

If the polynomial \( f(x) \) has degree less than \( n \), it is said to be the standard representative of its residue class; no other distinct member of its class has degree less than \( n \). It is customary to denote the residue class whose standard representative is \( f(x) \) simply by \( f(x) \).

The use of the term "residue class" for equivalence class becomes clear if one observes that, if \( g(x) \) is a polynomial, then the standard representative of \( g(x) \) is given by the corresponding \( f(x) \) of degree less than \( n \) that satisfies

\[
f(x) = g(x) + q(x)(x^n - 1).
\]

That is, \( f(x) \) is the residue of \( g(x) \) modulo \( x^n - 1 \).
2.4.2. Description of a Cyclic Code in Terms of its Generator Polynomial

It is easy to prove that a subspace (i.e., a linear code) in the algebra of polynomials modulo $x^n - 1$ is cyclic if and only if it is an ideal [43]. An ideal, $I$, in the algebra of polynomials modulo $x^n - 1$ is the set of all polynomials which are multiples of a given polynomial, $g(x)$ (called the generator of the ideal), which divides $x^n - 1$. If $r$ is the degree of $g(x)$, then the rank of the subspace of $n$-tuples corresponding to the ideal of polynomials generated by $g(x)$ is $n - r$. Thus the code has redundancy $r$.

A cyclic code is thus an ideal, and has a generator $g(x)$ which divides $x^n - 1$. Any standard representative which is a multiple of $g(x)$ is a code polynomial.

If

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{n-1} x^{n-1}$$

is the generator of a code (where $g_i = 0$ for $i > r$), then $\{g(x), xg(x), x^2g(x), \ldots, x^{n-r-1}g(x)\}$ are all code vectors, and hence the rows of the $(n-r) \times n$ matrix

$$
\begin{bmatrix}
g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\
0 & g_0 & g_1 & \cdots & g_{r-1} & g_r & 0 & \cdots & 0 \\
& & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{r-1} & g_r \\
\end{bmatrix}
$$

are all code vectors. Since the rows are linearly independent, the rank of $G$ is $n - r$, which is the dimension of the code. Therefore the row space of $G$ is the code space, and $G$ is a generator matrix of the code.
2.4.3. Description of a Cyclic Code in Terms of the Roots of the Generator Polynomial

An alternate method of specifying a code is to require that a polynomial \( f(x) \) be in the code if and only if it has certain specified roots, \( \alpha_1, \alpha_2, \ldots, \alpha_u \), assumed distinct for our purposes. Thus,

\[
f(x) = f_0 + f_1 x + f_2 x^2 + \ldots + f_{n-1} x^{n-1}
\]

is in the code if and only if

\[
0 = f(\alpha_i) = f_0 + f_1 \alpha_i + f_2 \alpha_i^2 + \ldots + f_{n-1} \alpha_i^{n-1}
\]

\[
= (f_0, f_1, f_2, \ldots, f_{n-1})(1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1})'.
\]

Hence the code is the null space of the matrix

\[
H^* = \begin{bmatrix}
1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{n-1} \\
1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha_u & \alpha_u^2 & \ldots & \alpha_u^{n-1}
\end{bmatrix}
\]

If \( H^* \) is not of full rank (i.e., contains dependent rows), then a parity check matrix, \( H \), of the code is obtained from \( H^* \) by deleting suitable rows of \( H^* \) in order that \( H \) is of full rank (equal to the rank of \( H^* \)).

The monic polynomial of smallest degree that has \( \alpha_i \) for a root is called the minimum function (or minimum polynomial) of \( \alpha_i \), and is denoted by \( m_i(x) \). If \( f(x) \) is a polynomial such that \( f(\alpha_i) = 0 \), then \( f(x) \) is divisible by \( m_i(x) \). Thus the requirement that \( f(x) \) is a code vector if and only if \( \alpha_1, \alpha_2, \ldots, \alpha_u \) are roots of \( f(x) \) is equivalent to the requirement that \( f(x) \) be divisible by the least common multiple (LCM) of \( m_1(x), m_2(x), \ldots, m_u(x) \). The LCM of \( m_1(x), m_2(x), \ldots, m_u(x) \) is the product of all the different minimum functions of the elements \( \alpha_1, \alpha_2, \ldots, \alpha_u \). Thus, if several of the \( \alpha_i \)'s have the same minimum function,
that minimum function appears only once as a factor of the LCM. (It
is noted that two minimum functions having a common root are identical.)

Thus the code is the ideal generated by

\[ g(x) = \text{LCM}[m_1(x), m_2(x), \ldots, m_u(x)]. \]

The following considerations show how to determine the length of the
code generated by \( g(x) \). The order of a nonzero element \( \alpha \) is the
least integer \( e \) greater than 0 such that \( \alpha^e = 1 \). (Since \( \alpha^{q-1} = 1 \)
for all \( \alpha \) in GF(\( q \)), the order of every (nonzero) element is \( \leq q - 1 \).)
Since \( g(x) \) must divide \( x^n - 1 \), it follows that \( \alpha_1, \alpha_2, \ldots, \alpha_u \) must all be
roots of \( x^n - 1 \). That is, \( \alpha_i^n = 1 \). But since \( e_i \) (the order of \( \alpha_i \)) is
the least integer such that \( \alpha_i^{e_i} = 1 \), \( e_i \) must therefore divide \( n \), for
\( i = 1, 2, \ldots, u \). Thus the order of each \( \alpha_i \) divides \( n \), and we can choose
\( n \) to be the LCM of the orders of the \( \alpha_i \). (If \( \alpha \) is primitive, then the
order of \( \alpha \) is \( q - 1 \), and then \( n \) assumes the maximum value \( n = q - 1 \).)

It is always possible to express several nonzero elements of GF(\( q \))
as powers of a single element of GF(\( q \)). (In fact, there exists at least
one element, \( \alpha \), such that \( \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{q-1} = 1 \) are all the differ-
ent nonzero elements of GF(\( q \)). Such an element is called a primitive
element, and it is obviously of order \( q - 1 \).) Suppose that the roots
\( \alpha_1, \alpha_2, \ldots, \alpha_u \) of the generator \( g(x) \) are given as powers of the single
element \( \alpha \) of order \( e \). If \( \alpha_i = \alpha^{u_i} \), then the degree of the factor
\( m_i(x) \) of \( g(x) \) is the number of distinct residues modulo \( e \) of \( u_i, u_i^p, u_i^p^2, \ldots \). The number of factors of \( g(x) \) is the number of different
\( m_i(x) \).

We note at this point that every root of a given irreducible
polynomial has the same order. The order of the roots of an irreduc-
bile polynomial is called the exponent to which that polynomial belongs.
Finally, if \( \alpha \) is a root of an irreducible polynomial \( p(x) \) of degree \( m \) with coefficients in \( \text{GF}(p) \), then \( \alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{m-1}} \) are all the roots of \( p(x) \).

2.5. **Bose-Chaudhuri Codes**

The error-correcting-codes that are as a class the best of the known constructive codes for channels in which errors affect successive symbols independently are the Bose-Chaudhuri codes. They were also discovered independently by Hocquenghem, and are also called Bose-Chaudhuri-Hocquenghem codes, or simply BCH codes. We shall use the descriptor BCH. The BCH codes are cyclic codes, and they can be used for correcting multiple additive errors in a codeword. The symbols are elements of \( \text{GF}(q=p^m) \).

The BCH codes are most easily defined in terms of the roots of the generator polynomial. If \( m_0 \) is any integer, and \( \alpha \) is any element of \( \text{GF}(p^m) \), then the code consisting of all polynomials \( \{f(x)\} \) over \( \text{GF}(q) \) for which (the column vectors of \( m \) components)

\[
\alpha^{m_0}, \alpha^{m_0+1}, \ldots, \alpha^{m_0+d-2}
\]

are roots of \( f(x) \) is a BCH code. Generally, \( m_0 \) is taken equal to 0 or 1. The length \( n \) of the code is the order \( e \) of \( \alpha \). The minimum distance of the BCH code is \( d \). The parity check matrix, \( H \) is derived from

\[
H^* = \begin{bmatrix}
1 & \alpha^{m_0} & (\alpha^{m_0})^2 & \cdots & (\alpha^{m_0})^{n-1} \\
1 & \alpha^{m_0+1} & (\alpha^{m_0+1})^2 & \cdots & (\alpha^{m_0+1})^{n-1} \\
& & & \cdots & \\
1 & \alpha^{m_0+d-2} & (\alpha^{m_0+d-2})^2 & \cdots & (\alpha^{m_0+d-2})^{n-1}
\end{bmatrix},
\]

where \( \mathbf{1} \) denotes the column vector \( (1, 0, \ldots, 0)' \) of \( m \) components, by
deleting suitable dependent rows.

The most important BCH codes correspond to the case where \( m_0 = 1 \), \( d = 2t + 1 \), and \( \alpha \) is a primitive element of \( \text{GF}(2^m) \). Then \( f(x) \) is a code polynomial if and only if

\[
(2.1) \quad \alpha, \alpha^2, \ldots, \alpha^{2t}
\]

are all roots of \( f(x) \). As noted at the end of Section 2.4, however, if \( \alpha \) is a member of \( \text{GF}(q=p^m) \) and has minimum function \( m(x) \) of degree \( m \), then

\[
\alpha, \alpha^p, \alpha^{2p}, \ldots, \alpha^{p^{m-1}}
\]

are distinct and are all the roots of \( m(x) \). Thus, for \( p=2 \), every even power of \( \alpha \) is a root of the minimum function of a lower odd power of \( \alpha \). Hence

\[
\alpha, \alpha^2, \ldots, \alpha^{2t}
\]

are roots of \( f(x) \) if and only if

\[
(2.2) \quad \alpha, \alpha^3, \ldots, \alpha^{2t-1}
\]

are roots of \( f(x) \), i.e., we may delete the even roots from the roots listed in (2.1). Thus an equivalent specification of the code is the statement that \( f(x) \) is a code polynomial if and only if

\[
\alpha, \alpha^3, \ldots, \alpha^{2t-1}
\]

are all roots of \( f(x) \). Thus the generator of the code is

\[
g(x) = \text{LCM}[m_1(x), m_3(x), \ldots, m_{2t-1}(x)].
\]

where \( m_i(x) \) is the minimum function of \( \alpha_i \). It is remarked that, as noted previously, the polynomials \( m_i(x) \) have no factors in common, unless they are identical (which will be the case if two of the \( \alpha_i \) have the same minimum function). There may be additional elements \( \alpha_i \) of (2.2) which have the same minimum function as lower powers of \( \alpha \), and these may be deleted from (2.2) without changing the code. The
quantity \( \alpha^j \) will have the same minimum function as \( \alpha^i \), \( j < i \), provided that there is an integer \( u \) such that \( jp^u \equiv i \) (modulo \( p^m - 1 \)). If
\[
\alpha_1, \alpha_2, \ldots, \alpha_l, \quad l + 1 \leq t,
\]
are all the powers of \( \alpha \) in (2.2) which have different minimum functions, then \( g(x) \) is given by
\[
g(x) = m_1(x), m_1^2(x), \ldots, m_1^l(x).
\]
Each factor \( m_i(x) \) has degree less than or equal to \( m \) and the degree of \( g(x) \) is therefore at most \( mt \). The redundancy is thus at most \( mt \).
Thus for any choice of \( m \) and \( t \) there is a \( t \)-error-correcting BCH code of length \( n=2^m-1 \) which has no more than \( mt \) parity check symbols.

2.6. Examples of BCH Codes


The simplest example of a BCH code is a binary single-error-correcting code, discovered by Hamming. We take \( m_0 = 1 \), \( d = 3 \), and let \( \alpha \) be a primitive element of \( \text{GF}(2^m) \). Then the code is determined by the requirement that \( (f(x)) \) is a code polynomial if and only if \( \alpha \) is a root of \( f(x) \). (Simply set \( t = 1 \) in (2.2).) Suppose that we take \( m = 3 \). Then the elements of \( \text{GF}(2^3) \) are all eight polynomials of the form
\[
a_0 + a_1 x + a_2 x^2,
\]
where \( a_i = 0 \) or 1. These polynomials, and the corresponding coefficient vectors, are shown below:
\[
\begin{align*}
0 & \quad (0,0,0) \\
1 & \quad (1,0,0) \\
x & \quad (0,1,0) \\
x^2 & \quad (0,0,1) \\
1 + x & \quad (1,1,0) \\
1 + x^2 & \quad (1,0,1) \\
x + x^2 & \quad (0,1,1) \\
1 + x + x^2 & \quad (1,1,1)
\end{align*}
\]

The element \( \alpha = x = (0,1,0) \) is a primitive element of the Galois field \( \mathbb{GF}(2^3) \) of polynomials modulo the irreducible polynomial \( x^3 + x + 1 \), since

\[
\begin{align*}
\alpha &= x \\
\alpha^2 &= x^2 \\
\alpha^3 &= x + 1 \\
\alpha^4 &= x^2 + x \\
\alpha^5 &= x^3 + x^2 = x^2 + x + 1 \\
\alpha^6 &= x^3 + x^2 + x = x^2 + 1 \\
\alpha^7 &= x^3 + x = 1 = x^0
\end{align*}
\]

are all the nonzero elements of \( \mathbb{GF}(2^3) \). Thus \([f(x)]\) is a code polynomial if \( \alpha = (0,1,0) \) is a root. The minimum function of \( \alpha \) is

\( m(x) = x^3 + x + 1 \). The length of the code is the order of \( \alpha \), and since \( \alpha \) is primitive, this is \( n = 2^3 - 1 = 7 \). The generator \( g(x) \) is

\[ g(x) = m(x) = 1 + x + x^3, \]

and hence the generator matrix \( G \) is the matrix whose rows are the coefficient vectors of \([g(x)], [xg(x)], [x^2g(x)], \) and \([x^3g(x)]\) considered as polynomials of degree \( n-1 = 6 \):
$G = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}.$

The parity check matrix is

$$H = [1, \alpha, \alpha^2, \ldots, \alpha^6] = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{bmatrix}.$$ 

This matrix is the matrix $H$ on page 41 with the order of the columns changed, and the code is thus equivalent to the code given in Figure 1.2.

It does not matter which irreducible polynomial we take to define the Galois field $GF(2^3)$. (Of course, if the irreducible polynomial is not primitive, then no root of it will be a primitive element). For example, we can choose $GF(2^3)$ to be the polynomials with coefficients in $GF(2)$, modulo $x^3 + x^2 + 1$. This polynomial is also primitive, and so its root $\alpha = x = (0,1,0)$ is primitive, and we have

$$\begin{align*}
\alpha &= x \\
\alpha^2 &= x^2 \\
\alpha^3 &= x^2 + 1 \\
\alpha^4 &= x^3 + x = x^2 + x + 1 \\
\alpha^5 &= x^3 + x^2 + x = x + 1 \\
\alpha^6 &= x^2 + x \\
\alpha^7 &= x^3 + x^2 + 1 = x^0 \\
\end{align*}$$

Again, $(f(x))$ is a code polynomial if $\alpha = (0,1,0)$ is a root. The minimum function of $\alpha$ is $1 + x^2 + x^3$, and the generator matrix is thus
\[ G = \begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}. \]

The parity check matrix is
\[ H = \begin{bmatrix}
1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 1
\end{bmatrix} \]

which is combinatorially equivalent to the matrix \( H \) on page 41. The two codes are thus equivalent.

It is noted that
\[ x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1). \]

Any generator \( g(x) \) must divide \( x^7 - 1 \). Since \( \alpha, \alpha^2, \alpha^4 \) are all different and are all the roots of \( g(x) \), the degree of \( g(x) \) is three. Thus \( g_1(x) = x^3 + x + 1 \) and \( g_2(x) = x^3 + x^2 + 1 \) are the only two possibilities for \( g(x) \).

2.6.2. A BCH Binary Two-Error-Correcting Code

We now consider an example of a BCH code that is capable of correcting any combination of \( t=2 \) additive errors. We take \( m_0 = 1 \) and \( d = 2t+1 = 5 \), and let \( \alpha \) be an element of \( \text{GF}(2^m) \). If we take \( m = 5 \), then the elements of \( \text{GF}(2^5) \) are the 32 polynomials of the form
\[ a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4. \]

Suppose that we take \( \alpha \) as a primitive element of \( \text{GF}(2^5) \), so that \( n = 2^5 - 1 = 31 \). Then
\[ \alpha, \alpha^3 = \alpha^{2^t-1} \]
must be roots of \( g(x) \). Let \( \alpha \) be a root of the primitive polynomial \( x^5 + x + 1 \). From the tables in Peterson [37], the minimum functions of \( \alpha \) and \( \alpha^3 \) are seen to be

\[ m_1(x) = x^5 + x^4 + x^3 + x^2 + 1 \]

and

\[ m_3(x) = x^5 + x^4 + x^2 + x + 1. \]

Thus \( g(x) = m_1(x) \cdot m_3(x) = 1 + x + x^6 + x^7 + x^{10} \).
CHAPTER III

A NEW TECHNIQUE FOR CORRECTING SYNCHRONIZATION ERRORS

3.1. Introduction

3.1.1. Notation

This chapter will develop a new technique for correcting synchronization errors in cyclic codes (such as the BCH codes). The technique will enable the receiver to reestablish proper word synchronization with the word immediately following the word in which the synchronization error actually occurs, while also enabling the receiver to correct additive errors which occur in this immediately following word.

Before proceeding further, it is desirable to introduce some terminology. Suppose that \( \underline{w} = (w_1, w_2, \ldots, w_N) \), \( \underline{x} = (x_1, x_2, \ldots, x_N) \), and \( \underline{z} = (z_1, z_2, \ldots, z_N) \) are three (not necessarily different) words that are sent successively over the channel. Thus the transmitter sends the sequence

\[ \ldots, w_1, w_2, \ldots, w_N, x_1, x_2, \ldots, x_N, z_1, z_2, \ldots, z_N, \ldots \]

over the channel. If no synchronization errors occur, suppose that the received words corresponding to these transmitted words are \( \underline{w} + e_\underline{w} \), \( \underline{x} + e_\underline{x} \), and \( \underline{z} + e_\underline{z} \), where \( e_\underline{w} = (e_{w_1}, e_{w_2}, \ldots, e_{w_N}) \), \( e_\underline{x} = (e_{x_1}, e_{x_2}, \ldots, e_{x_N}) \), and \( e_\underline{z} = (e_{z_1}, e_{z_2}, \ldots, e_{z_N}) \) are N-vectors representing the additive errors that occur in transmission. Suppose now that, in addition to the additive errors, a synchronization error occurs such that \( \ell \) symbols are lost in the vector \( \underline{w} + e_\underline{w} \), so that \( \underline{w}^* = (w_1^* \ldots) \).
is received instead of $w + e_w$, and suppose further that no synchronization error occurs in $x + e_x$. Then the receiver receives the sequence

$$\ldots, w^*_1, w^*_2, \ldots, w^*_N - 1, x_1 + e_{x1}, x_2 + e_{x2}, \ldots, x_N + e_{xN}, z_1 + e_{z1}, z_2 + e_{z2}, \ldots, z_N + e_{zN}, \ldots$$

and partitions it into words as

$$\ldots, (w^*_1, \ldots, w^*_N - 1, x_1 + e_{x1}, \ldots, x_i + e_{xi}), (x_{i+1} + e_{x(i+1)}, \ldots, x_N + e_{xN}, z_1 + e_{z1}, \ldots, z_i + e_{zi}), (z_{i+1} + e_{z(i+1)}, \ldots)$$

As in the usual (additive) case, we shall call these words the received words.

We shall not be concerned with the first of these received words, namely

$$(w^*_1, \ldots, w^*_N - 1, x_1 + e_{x1}, \ldots, x_i + e_{xi}),$$

which contains the word with the actual synchronization error. We shall, instead, examine the received word

$$u = (x_{i+1} + e_{x(i+1)}, \ldots, x_N + e_{xN}, z_1 + e_{z1}, \ldots, z_i + e_{zi})$$

$$= (x_{i+1}, \ldots, x_N, z_1, \ldots, z_i) + (e_{x(i+1)}, \ldots, e_{xN}, e_{z1}, \ldots, e_{zi})$$

in order to determine which synchronization error has occurred. That is, $u$ will be used to determine how many symbols were lost or how many symbols were gained prior to $x$, but not to determine which specific symbols prior to $x$ were lost or gained. If $x_R$ denotes the vector that is received corresponding to $x$ when no additive errors occur in $x$ or $z$, but a left-shift error of order $i$ occurs prior to $x$, we have

$$x_R = (x_{i+1}, \ldots, x_N, z_1, \ldots, z_i),$$

and so
\( u = x_R + e_N \)

where

\[ e_N = (e_{x(i+1)}, \ldots, e_{xN}, e_{z1}, \ldots, e_{zi}) \]  

All received words following the received word containing the synchronization error are called **overlaps**. Thus, for example, \( u \) and the received word following \( u \) are overlaps. Since the synchronization error is a left-shift error (i.e., symbol loss) of order \( i \), as described on page 13, the above overlaps are more specifically called left-shift overlaps of order \( i \).

At this point, we introduce notation to describe the vector which is obtained by cyclically shifting a given vector \( i \) places to the left. If \( a = (a_1, a_2, \ldots, a_h) \) is an arbitrary vector, we shall denote

\[
\underline{a}(i) = (a_{i+1}, a_{i+2}, \ldots, a_h, a_1, a_2, \ldots, a_i)
\]

for \( i = 0, 1, 2, \ldots, h-1 \), and define \( \underline{a}(-1) = \underline{a}(h-1) \). We adopt the convention that \( a_{i+bh} = a_i \) where \( b \) is any integer (positive, negative, or zero) and \( h \) is the word length. We can then write \( \underline{a}(i) = \underline{a}(i+bh) \) where \( b \) is any integer.

Since \( \underline{x} = (x_1, x_2, \ldots, x_N) \), we have

\[
\underline{x}(i) = (x_{i+1}, x_{i+2}, \ldots, x_N, x_1, x_2, \ldots, x_i)
\]

and hence we can write

\[
\underline{x}_R = \underline{x}(i) + \underline{e}_{N-1}
\]

Thus, if a left-shift error of order \( i \) occurs, we have

\[
\underline{x}_R = \underline{x}(i) + \underline{e}_{N-1}
\]

where the first \( N-1 \) coordinates of \( \underline{e}_{N-1} \) are zero and the last \( i \) coordinates are \( z_{i-1} - x_{i-1}, z_{i-2} - x_{i-2}, \ldots, z_1 - x_1 \).

Suppose now that the words \( w, x, \) and \( z \) are transmitted, in that
order and that, instead of i symbols having been lost, i symbols have
been gained, so that instead of receiving \( w + e_w \), the receiver
receives \( (w_1^*, w_2^*, \ldots, w_{N+1}^*, \ldots, w_N^*) \). That is, corresponding to
the transmitted sequence
\[
\ldots, w_1^*, w_2^*, \ldots, w_{N+1}^*, x_{1}^*, x_{2}^*, \ldots, x_{N}^*, z_{1}^*, z_{2}^*, \ldots, z_{N}^*
\]
the receiver receives
\[
\ldots, w_1^*, w_2^*, \ldots, w_{N+1}^*, x_{1}^* x_{1}^* + e_{x_{1}} x_{2}^* + e_{x_{2}} \ldots, x_{N}^* x_{N}^* + e_{x_{N}} z_{1}^* + e_{z_{1}} z_{2}^* + e_{z_{2}} \ldots, z_{N}^* + e_{z_{N}} \ldots
\]
and partitions it into the received words
\[
\ldots, (w_1^*, w_2^*, \ldots, w_N^*), (w_{N+1}^*, w_{N+2}^*, \ldots, w_{N+1}^*, x_{1}^* x_{1}^* + e_{x_{1}} x_{2}^* + e_{x_{2}} \ldots, x_{N}^* x_{N}^* + e_{x_{N}} z_{1}^* + e_{z_{1}} z_{2}^* + e_{z_{2}} \ldots, z_{N}^* + e_{z_{N}} \ldots)
\]
We ignore the first word, and shall consider the first overlap (i.e.,
the first received word after the synchronization error), namely
\[
u = (w_{N+1}^*, w_{N+2}^*, \ldots, w_{N+1}^*, x_{1}^* x_{1}^* + e_{x_{1}} x_{2}^* + e_{x_{2}} \ldots, x_{N}^* x_{N}^* + e_{x_{N}} x_{N-1}^*)
\]
to determine which synchronization error has occurred.

If (as in the case of left-shift errors) \( x_R \) denotes the vector
that is received corresponding to \( x \) when no additive errors occur in
\( x \), then if a right-shift error of order i occurs prior to \( x \), we have
\[
x_R = (w_{N+1}^*, \ldots, w_{N+1}^*, x_1^*, \ldots, x_{N-1}^*)
\]
and
\[
e_N = (0, 0, \ldots, 0, e_{x1}, \ldots, e_{x(N-i)})
\]
so that we can write
\[
(3.4) \quad u = x_R + e_N
\]
Since the synchronization error is a right-shift error (i.e., symbol
gain) of order \( i \), as described on page 13, the overlap \( u \) is called a right-shift overlap of order \( i \).

Since

\[
x(\text{N-1}) = (x_{\text{N-1+1}}, \ldots, x_N, x_1, \ldots, x_{\text{N-1}})
\]

we can write

\[
x_R - x(\text{N-1}) = (w_{\text{N+1}-x_{\text{N-1+1}}}, w_{\text{N+2}-x_{\text{N-1+2}}}, \ldots, w_{\text{N+i}-x_N}, 0, 0, \ldots, 0).
\]

Thus if a right-shift error of order \( i \) occurs, we have

\[
(3.5) \quad x_R = x(\text{N-1}) + h_{\text{N-1}},
\]

where the last \( N-1 \) coordinates of \( h_{\text{N-1}} \) are zero and the first \( i \) coordinates are \( w_{\text{N+1}-x_{\text{N-1+1}}}, w_{\text{N+2}-x_{\text{N-1+2}}}, \ldots, w_{\text{N+i}-x_N} \).

Whenever there is no danger of confusion both types of synchronization errors described above, the right-shift and the left-shift errors of order \( i \), are loosely referred to as synchronization errors of order \( i \).

Later in this chapter it will be convenient to consider all synchronization errors as left-shift errors of a particular order; that is, right-shift errors of order \( i \) will be considered to be left-shift errors of order \( N-i \). Note that if no synchronization error has occurred then \( x_R = x(0) = x \), and the receiver receives

\[
u = x_R + e_N = x + e_N.
\]

3.1.2. General Nature of the New Synchronization Technique

One of the primary considerations in developing the present technique was the desire to maintain the cyclic nature of the coding process. Thus, for example, it would not be acceptable to alter cyclic codes in such a fashion that linear switching circuits could no longer be used for implementation.
The approach used was to determine a method such that if a synchronization error were to occur, then the succeeding overlaps would possess some observable property that valid received words do not possess. Further, in order to be able to correct the synchronization error, that is, identify the direction (left or right) and the order (i) of the synchronization error, it must be possible to derive this additional information from the overlaps, once it has been determined that a synchronization error has occurred.

The technique developed can be applied to any $t_a$-additive-error-correcting cyclic code. The code to which the technique is applied will be called the parent code. The two basic requirements described above have been met by adding redundancy of two different types to the codewords of the parent code, and adding to the new codewords, before transmission, a certain vector which is subtracted by the received from the received words.

At this point it must be noted that a certain cost is associated with the ability to correct synchronization errors. Synchronization errors cannot be corrected without the utilization of redundancy incorporated into the codewords. Either the redundancy of the codewords must be increased above the minimum amount needed for the additive-error correction, or else some of the additive-error-correcting ability of the code must be sacrificed by using some of the redundancy generally used for additive-error correction to correct synchronization errors instead. In other words, we cannot get something for nothing.

It is shown in Chapter IV that the present technique is a reasonably efficient one for correcting synchronization errors of order that
are small compared with the word length. Furthermore, the technique is flexible in that it can be applied in such a fashion that some of the redundancy that is added for correction of synchronization errors can be used at the receiver's option for correction of additional additive errors instead of synchronization errors.

3.2. The Technique

3.2.1. The Subcode

Let us suppose that we have a $t_a$-additive-error-correcting cyclic code, denoted by $C$, which we will assume to be a BCH code of length $n$. Let $x = (x_1, x_2, ..., x_n)$ denote a word of the code $C$, and let us assume that we wish to be able to correct right-shift errors of order up to $t_x$ and left-shift errors of order up to $t_x$. Let $t_s = t_x + t_s$. Now, a left-shift error of order $i$ results in the same overlaps as a right-shift error of order $n-i$. Thus if we can correct left-shift errors of order up to $i$, whence we also correct right-shift errors of orders $n-1$, $n-2$, ..., $n-i$, we would wish additionally to correct right-shift errors of order at most $n-i-1$. Thus we may set $t_x + t_s \leq i + n - i - 1$, that is, $t_s \leq n - 1$.

We shall assume that $\alpha$ is a primitive element of $GF(q=p^m)$, and that the roots of the generator polynomial of the parent BCH code are $\alpha, \alpha^2, \alpha^3, ..., \alpha^{2ta}$. (The assumption that $\alpha$ is primitive may be relaxed; this assumption is made to simplify the development of the technique.) Denote the minimum function of $\alpha^i$ by $m_i(x)$ for $i = 1, 2, ..., 2t_a$. Let $\alpha^s(s \geq 2t_a + 1)$ be any element of $GF(q)$ that has a minimum function $m(x)$ that is different from the minimum functions $m_i(x)$, $i=1,2,...,2t_a$. We also re-
quire that \( s \) is so chosen that

\[
(\text{order of } \alpha^s) > t_s
\]

This condition is always satisfied if \( \alpha^s \) is primitive (i.e., of maximum order, \( n \)), since \( t_s \leq n - 1 \). If \( \alpha^s \) is not primitive, but is instead of order \( e < n - 1 \), then the requirement \( t_s < e \) will impose a real restriction on the value of \( t_s \).

Consider now the code \( C_s \) derived from \( C \) by taking the generator of \( C_s \) to be

\[
g_s(x) = g(x)m(x),
\]

where

\[
g(x) = \text{LCM}[m_1(x)m_2(x), \ldots, m_{2t_a}(x)]
\]

and where \( m(x) \neq x - 1 \) is not a factor of \( g(x) \). That is, \( C_s \) is the code that has

\[
\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t_a}, \alpha^s
\]

as roots. Clearly every word of \( C_s \) belongs to \( C \). We shall call \( C_s \) the subcode derived from the parent code \( C \).

The subcode \( C_s \) will be used for encoding information symbols. If \( r \leq mt \) is the redundancy of \( C \), whence \( r_s = r + \text{degree}[m(x)] \) is the redundancy of \( C_s \), then to obtain \( C_s \) we have in effect replaced \( r_s - r \) of the information symbols of words of \( C \) by redundant symbols. Hence, in the development of our synchronization technique, we have thus far relinquished \( r_s - r \) information symbols to redundancy. The number of information symbols in a word of \( C_s \) is \( k_s = n - r_s \).

3.2.2. Augmentation of the Subcode

Consider the code \( C_A \) that is derived from the subcode \( C_s \) by repeating the first \( t_s \) digits of each codeword of \( C_s \). That is, if
\( x = (x_1, x_2, \ldots, x_n) \) is a codeword of \( C_S \), then the corresponding codeword of \( C_A \) is \( x_A = (x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_s) \). We shall call the code \( C_A \) the **augmented subcode** derived from the subcode \( C_S \). Thus we have added \( t_s \) redundant symbols to the codewords of \( C_S \), increasing the word length to \( n + t_s \).

### 3.2.3. Translation of the Augmented Subcode

Having augmented the subcode, we incorporate no further redundancy into the codewords. The next step in the technique consists of translating the augmented subcode by adding to every codeword a certain vector, denoted by \( c_A \) and called the **translation vector**. The vector \( c_A \) is of the form

\[
c_A = (c_1, c_2, \ldots, c_n, c_1, c_2, \ldots, c_s).
\]

If we denote

\[
c = (c_1, c_2, \ldots, c_n),
\]

then \( c \) is chosen such that \( c \) is a nonnull codeword of the parent code \( C \), but not a codeword of the subcode \( C_S \). Note that \( c_A \) is thus not a codeword of the code \( C_A \). The set of translated codewords does not contain the null vector, and is hence not a linear code. We shall call the set of translated codewords of the augmented subcode the **translated code**, denote it by \( C_T \), and formally write

\[
C_T = C_A + c_A.
\]

The set of translated codewords is in fact a coset of the code \( C_A \). The words of the translated code \( C_T \) are the words actually sent over the channel.

It is noted that if we add \( c \) to the words of the subcode \( C_S \) and then augment the resulting code, we also obtain \( C_T \).
Before proceeding to an explanation of the decoding procedure, we shall briefly summarize the encoding procedure. Suppose that \( s = (s_1, s_2, \ldots, s_{k_S}) \), \( k_S = n - r_S \), is a vector of \( k_S \) information symbols generated by the source, and that \( x = (x_1, x_2, \ldots, x_n) \) is the codeword of \( C_S \) that corresponds to the information vector \( s \). The augmented vector

\[
x_A = (x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_{t_S})
\]

is formed, and then translated, to obtain

\[
x_T = x_A + c_A
\]
\[
= (x_1 + c_1, x_2 + c_2, \ldots, x_n + c_n, x'_{t_S} + c_1, x'_2 + c_2, \ldots, x'_{t_S} + c_{t_S}).
\]

The vector \( x_T \) is then sent over the channel. We can represent the above process by

\[
S \rightarrow X \rightarrow X_A \rightarrow X_T
\]

where

\[
x \in C_S, \quad x_A \in C_A, \quad \text{and} \quad x_T \in C_T.
\]

We shall introduce some terminology at this point. Let us define \( x_{n+1} = x_i \) for \( i = 1, 2, \ldots, t_S \), and denote \( N = n + t_S \) so that we may write \( x_A = (x_1, x_2, \ldots, x_N) \). We shall refer to a left-shift synchronization error of order \( i \leq t_S \) as a correctable left-shift error, and refer to a right-shift synchronization error of order \( i \leq t_S \) as a correctable right-shift error. A synchronization error of either of the preceding types will be referred to as a correctable synchronization error. An additive error \( e_N = (e_1, e_2, \ldots, e_N) \) such that \( t_a \) or fewer of the symbols \( e_{t_a + 1}, e_{t_a + 2}, \ldots, e_{N-t_a} \) are nonzero is called a correctable additive error.
3.2.4 Retranslation by the Receiver

Suppose that the vector

\[ x_T = x_A + e_A \]

has been transmitted over the channel. Suppose that, corresponding to \( x_T \), the receiver receives the vector

\[ u = x_R + e_N, \]

as described on pages 56 and 57, where \( e_N = (e_1, e_2, \ldots, e_N) \) is an error vector representing any additive errors that may have occurred in the digits of \( x_R \). We assume that no synchronization error has occurred in \( x_T \), but that a synchronization error may have occurred in a word prior to \( x_T \), so that \( x_T \) may be received out of synchronization.

Upon receiving the word

\[ u = x_R + e_N \]

the receiver subtracts \( c_A \) from it, to obtain

\[ y = x_R - c_A + e_N = (v_1, v_2, \ldots, v_N). \]

The vector \( y \) is the received word, and the vector \( y \) is called the translated received word. The receiver now deletes the first \( t_r \) coordinates and the last \( t_l \) coordinates of \( y \) to obtain

\[ y = (v_{t_r+1}, v_{t_r+2}, \ldots, v_{N-t_l}). \]

The vector \( y \) thus obtained is called the truncated translated received word, or simply the truncated word. According to the properties of the truncated word \( y \), the receiver will now correct any correctable additive errors that have occurred, and correct any correctable synchronization error that has occurred.

3.3. A Simple Expression for the Truncated Word

If a left-shift error of order \( i \leq t_l \) has occurred, we have
\[ y = \bar{x}_T(i) + \bar{e}_{N-1} - \bar{c}_A + e_N \]
\[ = \bar{x}_A(i) + c_A(i) + \bar{e}_{N-1} - \bar{c}_A + e_N \]
so that
\[ (3.6) \quad y = (x_{t+1}^{t+1}, x_{t+2}, \ldots, x_{t+n}) \]
\[ + (c_{t+1}, c_{t+2}, \ldots, c_{t+n}) \]
\[ - (c_{t+1}, c_{t+2}, \ldots, c_{t+n}) + e_n \]
\[ = x(t+1) + c(t+1) - c(t) + e_n, \]
where \( e_n = (e_{t+1}, e_{t+2}, \ldots, e_{N-1}) \). A term corresponding to \( \bar{e}_{N-1} \) does not appear in the expression for \( y \) since \( \bar{e}_{N-1} \) contains zeros as the first \( N-1 \) coordinates, while the remaining \( i \) coordinates, not necessarily zero, are removed by the right-hand truncation, since \( i \leq t_A \). Similarly, if a right-shift error of order \( i \leq t_A \) has occurred, we have
\[ y = x_T(N-1) + \bar{h}_{N-1} - \bar{c}_A + e_N \]
\[ = x_A(N-1) + c_A(N-1) + \bar{h}_{N-1} - \bar{c}_A + e_N \]
so that
\[ (3.7) \quad y = (x_{t+1}, x_{t+2}, \ldots, x_{t+n-1}) \]
\[ + (c_{t+1}, c_{t+2}, \ldots, c_{t+n-1}) \]
\[ - (c_{t+1}, c_{t+2}, \ldots, c_{t+n}) + e_n \]
\[ = x(t-1) + c(t-1) - c(t) + e_n \]
\[ = x(t+n-1) + c(t+n-1) - c(t) + e_n. \]
A term corresponding to \( \bar{h}_{N-1} \) does not appear in the expression for \( y \) since \( \bar{h}_{N-1} \) contains zeros as the last \( N-1 \) coordinates, while the first \( i \) coordinates, not necessarily zero, are removed by the left-hand truncation, since \( i \leq t_A \).

For vectors of length \( n \), a right-shift of order \( i \) can be regarded
as a left-shift of order $n-i$. We observe from (3.6) and (3.7) that,
in similar fashion, a right-shift error of order $i$ results in the
same vector $y$ as a left-shift error of order $n-i$. It follows that,
if we so desire, we can restrict consideration exclusively to left-
shift errors. That is, allowing left-shift errors of order $i \leq t_f$ and
right-shift errors of orders $i \leq t_r$ is the same as allowing left-shift
errors of orders $i \leq t_f$ or $i \geq n - t_r$ (i.e., $i = 1, 2, \ldots, t_f$,
$n-t_r, n-t_r+1, \ldots, n-1$).

3.4. Determination of the Translation Vector and the Subcode: the
Fundamental Theorem

3.4.1. The Fundamental Theorem

We shall now prove the theorem that shows how the technique can
be used to correct both additive and synchronization errors.

Theorem 3.1. Consider an $(n,k)$ $t_a$-additive-error-correcting BCH
code $C$ with generator polynomial

$$g(x) = \text{LCM}[m_1(x), m_2(x), \ldots, m_{2t_a^2}(x)]$$

of degree $r = n - k$, where $m_i(x)$ is the minimum function of $\alpha^i$, and $\alpha$
is a nonzero element of $\text{GF}(q=p^m)$. Consider the cyclic code $C_S$, a
subcode of $C$, which has generator polynomial

$$g_S(x) = g(x) m(x),$$

where $m(x)$ is the minimum function of $z = \alpha^s (s \geq 2t_a + 1)$ and $\alpha^s$
is not a root of $g(x)$. Let $c = (c_1, c_2, \ldots, c_n)$ be a codeword of $C$ but
not of $C_S$. Consider the code $C_T$, the words of which are formed by
adjoining the first $t_s$ symbols ($t_s < n$) of each word of $C_S$ to itself,
and adding to the result the vector $c_A = (c_1, \ldots, c_n, c_1, \ldots, c_{t_s})$.
Then the code $C_T$ can correct up to $t_a$ additive errors, left-shift
synchronization errors of order up to $t_f$ and right-shift synchroniza-
tion errors of order up to $t_r$, where $t'_r + t'_r = t_s$, subject to $t_s \leq s - 1$, where $e$ is the order of $z$. The synchronization error is corrected in the first complete received word after the word in which the synchronization error occurs. The code $C_T$ is an $(n_T, k_T)$ code, where $n_T = n + t_s = N$, and $k_T = k$-degree $[m(x)]$.

(Note that since $t_s \geq 1$, the requirement $t_s \leq e - 1$ implies $e \geq 2$. Thus $z \neq 1$ and therefore $m(x) \neq x - 1$. (Also, $z \neq 1$ implies $\alpha \neq 1$ and $\alpha \neq 0 \pmod{n}$.) If a code can correct right-shift errors of order up to $t_r$, and left-shift errors of order up to $t'_r$, then we simply say that the code can correct $t_s$ synchronization errors, where $t_s = t'_r + t_r$. If a code $C_T$ can correct up to $t_a$ additive errors, and $t_s$ synchronization errors, we shall call the code a $(t_a, t_s)$-error-correcting code. Note that $g(x)$ may be equal to zero, so that $t_a = 0$, and $s = 1$.)

Proof. The notation developed earlier in this chapter will be used in the proof. As we showed in the preceding section, we can write

$$\gamma = x(t_r + i) + c(t_r + 1) - c(t_r) + e_n$$

where $\gamma$ is the truncated word and $i = 1, 2, ..., t'_r$, $n - t_r$, $n - t_r + 1, ..., n - 1$. That is, we consider all synchronization errors as left-shift errors, as discussed in the preceding section. By the definition of the code $C$, a polynomial $f(x)$ is in the code $C$ if and only if $f(x) = a(x) \cdot g(x)$, where the degree of $a(x)$ is less than or equal to $k - 1$, or, equivalently, if and only if $f(x)$ has roots $\alpha, \alpha^2, \alpha^3, ..., \alpha^{2t_a}$. Let us denote the parity check matrix of $C$ by $H$. If we denote

$$H^* = \begin{bmatrix}
1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\
1 & \alpha^2 & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{n-1} \\
1 & \alpha^{2t_a} & (\alpha^{2t_a})^2 & (\alpha^{2t_a})^3 & \cdots & (\alpha^{2t_a})^{n-1}
\end{bmatrix}$$
where the element 1 in the matrix denotes the m-coordinate column vector
\[
\begin{bmatrix}
1 \\
0 \\
\vdots \\
0
\end{bmatrix},
\]
then H is obtained from \( H^* \) by deleting suitable dependent rows. If we denote the parity matrix of \( C_S \) by \( H_S \), then we can write \( H_S \) in the form
\[
H_S = \begin{bmatrix}
H \\
H_2
\end{bmatrix}
\]
where \( H_2 \) is derived from
\[
H_2^* = [1, \alpha^8(\alpha^3)^2(\alpha^3)^3 \ldots (\alpha^3)^{n-1}]
= [1, z, z^2, z^3, \ldots, z^{n-1}]
\]
by striking out any dependent rows of \( H_2^* \), so that \( H_2 \) is of full rank (i.e., the number of rows = \( \deg[m(x)] \)). In this proof we shall be using the matrix \( H_2^* \) rather than \( H_2 \), in order to express certain syndromes in terms of the \( z \)'s. Since the vector \( c \) is in the code \( C \), but not in the code \( C_S \), we have
\[
c^t H' = 0 \quad \text{and} \quad c^t H_S' \neq 0,
\]
or, more specifically,
\[
c^t H' = 0 \quad \text{and} \quad c^t H_2' \neq 0.
\]
Also, we have \( c^t H_S^* \neq 0 \). Since \( C \) and \( C_S \) are cyclic codes, and \( a(i) \) is a cyclic permutation of \( a \) for any vector \( a \), we also have
\[
c^t (t^r) H' = 0, \quad c^t (t^r+1) H' = 0,
\]
and
\[
c^t (t^r) H_2' \neq 0, \quad c^t (t^r+1) H_2' \neq 0.
\]
Also
\[ c(t_r)H_s^i \neq 0, \quad \text{and} \quad c(t_r+1)H_s^{i'} \neq 0. \]
Similarly, since \( x \) is in \( C_S \), so is \( x(t_r+1) \), and we have
\[ xH_s^i = 0 \quad \text{and} \quad x(t_r+1)H_s^{i'} = 0. \]
Let us calculate the syndrome \( \chi H' \), which we will call the additive-error syndrome. We have
\[
\begin{align*}
\chi H' &= (x(t_r+1) + c(t_r+1) - c(t_r) + e_n) H' \\
&= x(t_r+1)H' + c(t_r+1)H' - c(t_r)H' + e_nH' \\
&= e_nH'.
\end{align*}
\]
Now the code \( C \) is a \( t_a \)-additive-error-correcting code, which means that any equation of the form
\[ a = e_nH', \]
where \( a \) is a known \( 1 \times r \) row vector, can be solved for the unknown vector \( e_n \), provided \( w(e_n) \leq t_a \). But the equation (3.8) is precisely of this form, since \( \chi H' \) is known. Hence we can determine the additive-error vector \( e_n \) in (3.8), and then correct the additive errors in \( x \) by calculating
\[
\chi_c = \chi - e_n \\
= x(t_r+1) + c(t_r+1) - c(t_r).
\]
Let us now calculate the syndrome \( \chi_c H_s^{i'} \), which we will call the synchronization-error syndrome. Since
\[ x(t_r+1)H_s^i = 0, \]
we have
\[
\begin{align*}
\chi_c H_s^{i'} &= (x(t_r+1) + c(t_r+1) - c(t_r))H_s^{i'} \\
&= x(t_r+1)H_s^{i'} + c(t_r+1)H_s^{i'} - c(t_r)H_s^{i'} \\
&= c(t_r+1)H_s^{i'} - c(t_r)H_s^{i'}.
\end{align*}
\]
Now we allow right-shift errors of orders 1, 2, ..., \( t_r \) and left-
shift errors of orders 1, 2, ..., \( t_x \). As discussed earlier, this is the same as allowing left-shift errors of orders 1, 2, ..., \( t_x \), \( n-t_x \), \( n-t_x+1 \), ..., \( n-1 \). Thus, in (3.9), \( i \) takes on the values \( i=0, 1, 2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \). Now in order to correct specified synchronization errors the syndromes (3.9) must be different for the different shift-errors allowed; that is, the syndrome (3.9) must be distinct for each value that \( i \) can assume. Hence

\[(3.9.1) \quad (c(t_x+i) = c(t_x))H_x^t) \]

must be distinct for \( i = 0, 1, 2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \). That is,

\[(3.10) \quad c(t_x+i)H_x^t \]

must be distinct for \( i = 0, 1, 2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \).

We observe that

\[c(t_x+i)H_x^t = c(t_x+i)[H_x^t H_x^t] \]

\[= [0_x, c(t_x+i)H_x^t], \]

where \( 0_x \) denotes the \( r \)-coordinate null vector. Thus the condition that the vectors (3.10) be distinct for \( i=0,1,2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \) is equivalent to the condition that

\[(3.11) \quad c(t_x+i)H_x^t \]

be distinct for \( i = 0, 1, 2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \) and this is equivalent to the condition that

\[(3.11.1) \quad c(t_x+i)H_x^t \]

be distinct for \( i = 0, 1, 2, ..., t_x, n-t_x, n-t_x+1, ..., n-1 \). We have

\[c(t_x+i)H_x^t = c(t_x+i)[1 z z^2 ... z^{n-1}], \]

\[= c_{t_x+i+1} + c_{t_x+i+2}z + c_{t_x+i+3}z^2 + ... + c_{t_x+i+n}z^{n-1} \]

where \( 1 \) denotes the \( m \)-coordinate unit vector.
\[ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} . \]

Since \( z^n \rightarrow z \) and \( c_i = c_{n+i} \) by convention, the expression for (3.11.1) can be written

\[
c(t + i)^H z = c_{t + i + 1} z^n + c_{t + i + 2} z^{n+1} + c_{t + i + 3} z^{n+2} + \ldots + c_{t + i + n} z^{2n-1} + c_{t + i + n} z^{2n-1} = z^{n-t_r-i} \left[ c_{t + i + 1} z^{t_r+i} + c_{t + i + 2} z^{t_r+i+1} + \ldots + c_{n} z^{n-1} + c_{1} z + c_{2} z^{n+1} + c_{3} z^{n+2} + \ldots + c_{t + i + n} z^{n+t_r+i-1} \right] = z^{n-t_r-i} \left[ c_{1} z^{1} + c_{2} z^{2} + \ldots + c_{n} z^{n-1} \right] = z^{n-t_r-i} c_{H_z} .
\]

Now \( c_{H_z} \neq 0 \), and \( z^{n-t_r-i} \) is nonzero since \( z \neq 0 \). From \( i = 0, 1, 2, \ldots, t_f, n-t_r, n-t_r+1, \ldots, n-1 \) we have, by rearranging, \( i = n-t_r, n-t_r+1, \ldots, n-1, 0, 1, 2, \ldots, t_f \). Hence \( z^{n-t_r-i} \) assumes the values

\[ z^{n-t_r-(n-t_r)}, z^{n-t_r-(n-t_r+1)}, \ldots, z^{n-t_r-(n-1)}, z^{n-t_r}, z^{n-t_r-1}, \ldots, z^{n-t_r-t_f} \]

or, since \( z^n = 1 \),

\[ z, z^{-1}, z^{-2}, \ldots, z^{n-t_r}, z^{-t_r}, \ldots, z^{-t_r-t_f} \]

and these are \( t_f + t_s + 1 = t_f + 1 \) consecutive descending powers of \( z \). But the order of \( z \) is \( e \), and \( t_f + 1 \leq e \) by hypothesis. Hence these \( t_s + 1 \) consecutive powers of \( z \) are all distinct. Hence the quantities \( c(t + i)^H z \) are distinct for the permitted values of \( i \). It follows that the
quantities
\[ c(t_r+1)H'_2 \]
are distinct for the permitted values of \( i \), and hence the quantities
\[ c(t_r+1)H'_S \]
are distinct for the permitted values of \( i \).

Hence
\[ Y_{c}H'_S \]
is distinct for the permitted distinct values of \( i \), i.e., for every distinct correctable synchronization error. Hence we can correct the synchronization error. We observe that if \( i=0 \), i.e., if no synchronization error has occurred, then the synchronization-error syndrome \( Y_{c}H'_S \) is zero.

It remains to prove the final statement of the theorem, that \( c_T \) is an \( (n_T, k_T) \) code, where \( n_T=n+t_S \), \( k_T=k \)-degree \([m(x)]\). That \( n_T=n+t_S \) is obvious, since the codewords of \( c_T \) are \( n+t_S \) symbols in length. To show that \( k_T = k\)-deg \([m(x)]\), we note that the degree of \( g_s(x) \) is \( r + \text{deg } [m(x)] \). Hence \( k_T = n + (r + \text{deg } [m(x)]) = (n-r) - \text{deg } [m(x)] = k\)-deg \([m(x)]\). This completes the proof of Theorem 3.1.

Note that \( \text{deg } [m(x)] \leq m \), since the degree of any minimum function is less than or equal to \( m \).

Observe that we can write (3.9.1) as
\[
(c(t_r+1)-c(t_r))H'_S = (c(t_r+1)-c(t_r))[H'_S,H'_2] \\
= [c(t_r),(c(t_r+1)-c(t_r))H'_2] .
\]

Both the quantities in (3.9.1), and the quantities
(3.9.1.1) \( c(t_r+1)-c(t_r))H'_2 \)
will be referred to as proper synchronization-error syndromes. We shall now give a number of corollaries to the above theorem.
Corollary 3.1.1. If \( z = \alpha^s \) is primitive, then the code \( C_m \) can correct synchronization errors of any desired order (by choosing \( t_s \) appropriately).

Proof. If \( z = \alpha^s \) is primitive, then

\[
\text{order of } z = n.
\]

The requirement that \( t_s \leq e - 1 \) hence becomes \( t_s \leq n - 1 \). But this means that synchronization errors of any desired order \( t_s \) and \( t_r \) can be corrected (see discussion on pp. 60, 69) simply by choosing \( t_s = t_s + t_r \).

Corollary 3.1.2. If the order \( e \) of \( z \) satisfies

\[ e = p^u - 1 \]

where \( u < m \) is a divisor of \( m \), then \( k_T = k - \log_p (e + 1) = k - u \).

Proof. It can be proved (Peterson [43], Theorem 6.25) that the order \( e \) of an element divides \( p^v - 1 \) but no smaller number of the form \( p^v - 1 \) where \( v \) is the degree of the minimum function of the given element. Hence, if \( e \) is the order of \( z \) and if \( e = p^u - 1 \), then \( u \) must be the degree of the minimum function, \( m(x) \), of \( z \): \( u = \deg[m(x)] \).

Since

\[ g_0(x) = g(x)m(x) \]

is thus of degree \( r + u \), we have

\[ k_T = n - (r + u) = n - r - u = k - u, \]

which proves the corollary.

Note that if \( u \) is the degree of the minimum function of \( z \), then the order of \( z \) is at most \( p^u - 1 \). Thus if the degree of the minimum function of \( z \) is \( u \), then the maximum value for \( t_s \) is \( p^u - 2 \), and this maximum is attainable only if the order \( z \) is in fact \( p^u - 1 \).

We can indicate some related facts concerning the form of the matrix \( H^*_2 \). The order of \( z \) is \( e \), whence \( z^e = 1 \), and \( e \) must hence divide
\[ n = p^m - 1 \] (Peterson [43], page 99). Hence we can write
\[
H^*_2 = \begin{bmatrix}
1 & z & z^2 & \ldots & z^{e-1} \\
1 & z & z^2 & \ldots & z^{e-1} \\
1 & z & z^2 & \ldots & z^{e-1} \\
\end{bmatrix}
\]
that is, the matrix \( H^*_2 \) consists of \( n/e \) cycles of the columns
\[(3.12) \ \ \ \ 1, z, z^2, \ldots, z^{e-1}.
\]
The \( m \)-vectors (3.12) together with the null \( m \)-vector, \( \mathbf{0} \), form a subfield of \( GF(p^m) \), and this subfield is isomorphic to \( GF(e+1) = GF(p^n) \). The matrix \( H^*_2 \), and hence the matrix \( H_2 \), is thus of rank \( u \); this is a restatement of Corollary 3.1.2.

**Corollary 3.1.3.** A vector \( \mathbf{g} \) satisfying the above theorem is the coefficient vector of the polynomial \( c(x) = g(x) \).

**Proof.** Let us write
\[
g(x) = g_1 + g_2 x + g_2 x^2 + \ldots + g_n x^{n-1}
\]
and set
\[
\mathbf{g} = (g_1, g_2, \ldots, g_n).
\]
We must show that the vector \( \mathbf{g} \) satisfies
\[
g H' = \mathbf{0}
\]
and
\[
g H'_2 \neq \mathbf{0}.
\]
Clearly, \( g H' = \mathbf{0} \), since \( g(x) \) generates \( \mathbf{C} \), and \( \mathbf{g} \) is thus in the code \( \mathbf{C} \).

We now must show that \( g H'_2 \neq \mathbf{0} \). To do this we employ proof by contradiction. Suppose that \( g H'_2 = \mathbf{0} \). Then \( \mathbf{g} \) is in \( \mathbf{C}_S \), since
\[
g H'_S = g [H' H'_2] = \mathbf{0}.
\]
Hence
\[(3.13) \ \ \ \ g(x) = q(x) g_S (x) \quad q(x) g(x)m(x),
\]
where
\[ \text{deg}[q(x)] \leq k_{n-1} = k - \text{deg}[m(x)] - 1 \]

Since
\[
\begin{align*}
\text{deg}[q(x)g(x)m(x)] &< \text{deg}[q(x)] + \text{deg}[g(x)] + \text{deg}[m(x)] \\
&< k_{n-1} + \text{deg}[g(x)] + \text{deg}[m(x)] \\
&= k + r + 1 = n - 1,
\end{align*}
\]

the equality (3.13) implies that
\[ (3.14) \quad q(x)m(x) = 1. \]

But since \( q(x)m(x) \neq 0 \), we have
\[
\text{deg}[g(x)m(x)] = \text{deg}[q(x)] + \text{deg}[m(x)],
\]
and since \( \text{deg}[m(x)] > 0 \), we have
\[ (3.15) \quad \text{deg}[q(x)m(x)] = \text{deg}[q(x)] + \text{deg}[m(x)] > 0. \]

But (3.14) implies \( \text{deg}[q(x)m(x)] = \text{deg}[1] = 0 \). Thus (3.14) and (3.15) are contradictory, whence \( g H'_2 \neq \emptyset \), as was to be proved.

3.4.2. **Summary of the Error Correction Procedure**

This section will summarize the error-correction procedure that was developed in the proof of Theorem 3.1. We assume that the procedure has been followed as described in Section 3.2, and that the receiver has the \( n \)-vector \( y \).

The first step by the receiver is to correct any correctable additive errors that have occurred. To do this, the receiver calculates the additive-error syndrome \( y H' \). Using the relationship
\[ y H' = e H', \]
where \( e \) is an unknown vector, but \( y H' \) is known, the receiver determines the vector \( e \) by using the same error-correction procedure as for the \( t \)-additive-error-correcting BCH code \( C \). (See [3], [14], [29], and [15], [20], [31], and [32].)
[37] for BCH code error-correction procedures.) Having determined the error vector \( \overline{e} \), the receiver subtracts \( \overline{e} \) from \( \overline{y} \) to obtain the corrected vector \( \overline{y}_c \):

\[
\overline{y}_c = \overline{y} - \overline{e}.
\]

The receiver now calculates the synchronization-error syndrome \( \overline{y}_c H'_2 \). (Equivalently, the receiver could calculate

\[
\overline{y}_c H'_2 = [\overline{y}_c H'_2] = [0, \overline{y}_c H'_2].
\]

If the receiver obtains the value 0, then he decides that no synchronization error has occurred, and he decides that \( \overline{y}_c \) is correct. He thus interprets that \( \overline{x} = \overline{y}_c(n-t_r) \), and proceeds to the next word. If a correctable synchronization error has occurred, however, the receiver may obtain for the synchronization-error syndrome \( \overline{y}_c H'_2 \) any one of the \( t_s \) different (nonnull) proper synchronization-error syndromes

\[
[\overline{c}(t_{r+1}+1)-\overline{c}(t_r)]H'_2,
\]

\( i = 1, 2, \ldots, t_r, n-t_r, n-t_r+1, \ldots, n-1. \) These are all known, pre-calculated vectors having \( \deg[m(x)] \) components. By comparing the value of \( \overline{y}_c H'_2 \) with these \( t_s \) vectors, the receiver finds that it is identical to one of them, say the one corresponding to \( i=i^* \). The receiver thus decides that a left-shift synchronization error of order \( i^* \) has occurred, and he shifts the word marks \( i^* \) places to the left. (If \( i \geq n-t_r \), the synchronization error can be considered as a right-shift error of order \( n-i^* \), and the receiver then shifts the word marks \( n-i^* \) places to the right.) The synchronization error is thus corrected.

3.4.3. Alternative Augmentation Procedures

In developing the new synchronization technique \( t_s \) symbols were added to the end of each word \( \overline{x} \) of the subcode, to form the augmented
word $x_A$. It should be noted that the $t_s$ symbols need not be added all at the end of the word, but some or all of them could be added at the beginning of the word. All that is actually required is that the augmented word $x_A$ be some cyclic shift of the augmented word formed as specified in Section 3.2.2. A natural procedure for forming the words of the augmented subcode would be to place $t_r$ symbols before $x$ and $t_f$ symbols after $x$, resulting in

$$x_A = (x_{t_f+1}, x_{t_f+2}, \ldots, x_{t_f+s}, x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_f).$$

Using this definition of the augmented subcode and defining $c_A = (c_{t_f+1}, c_{t_f+2}, \ldots, c_{t_f+s}, c_1, c_2, \ldots, c_n, c_1, c_2, \ldots, c_f)$, the truncated word becomes (instead of (3.6))

$$x(i) + c(i) = c + s_n,$$

and the proper synchronization syndrome (3.9.1.1) becomes

$$(c(i) - c) H_2^T.$$

Instead of interpreting $x = x_c(n-t_r)$ (as in Section 3.4.2), the receiver interprets $x = x_c$.

If the $t_s$ symbols are added to the end of the word $x$, then the encoding procedure is somewhat simpler than if $t_r$ symbols are added before $x$ and $t_f$ symbols are added after $x$, but the receiver must cyclically shift $x_c$ to obtain $x$. If $t_r$ symbols are added before $x$ and $t_f$ symbols are added after $x$, then the encoding is slightly more complicated, but the receiver interprets $x_c$ to be $x$, without having to shift $x_c$ at all. Whichever procedure is preferred depends upon whether it is more desirable to keep the encoding or the decoding as simple as possible. In aerospace applications, it is generally desired to keep encoding as simple as possible, indicating a preference for the procedure used in the text, namely adding the $t_s$ symbols to the end of $x$. The
increased complexity of the suggested alternate method appears to be quite small, however.
CHAPTER IV

THE EXTENDED HAMMING BOUND

4.1. Introduction

4.1.1. Bounds on the Error-Correcting Capabilities of Linear Codes

In order to measure the efficiency of linear additive-error-correcting codes, various upper and lower bounds have been determined for the error-correcting capability of linear codes. Peterson [43] provides a detailed discussion of these bounds. In order to measure the efficiency of the technique set forth in this dissertation, an upper bound has likewise been determined for the capability of a certain class of codes that correct both additive and synchronization errors. Before deriving this bound, we shall describe certain well-known bounds for additive-error-correcting codes.

4.1.2. The Plotkin Upper Bound

The Plotkin bound provides an upper bound to the number of information symbols, k, possible in a code having specified minimum weight. In other words, the Plotkin bound is a lower bound on the redundancy (i.e., the number of parity check symbols), r, required to obtain a code having specified minimum distance. Specifically, the bound states:

If \( n \geq \frac{(qd - 1)}{(q - 1)} \), then the number of information symbols, k, of an \((n,k)\) linear code having minimum distance d must satisfy
\[ k \leq n - \frac{qd}{q-1} + \frac{1}{q-1} + 1 + \log q \cdot \]

That is, the redundancy, \( r = n - k \), is at least
\[ \frac{qd}{q-1} - \frac{1}{q-1} - 1 - \log q \cdot \]

If \( q \) is very large, the last three terms may be ignored, and the
bound asymptotically becomes
\[ k \leq n - \frac{qd}{q-1} \cdot \]

This asymptotic bound is plotted for the binary case in Figure 4.1
which corresponds to Figure 4.1 of Peterson [43]. The quantity
\[ \frac{k}{n} = 1 - \frac{d}{2n} (\frac{q}{q-1}) = 1 - \frac{2d}{n} = 1 - 4(\frac{d}{2n}) \]

is plotted versus \( d/2n \). Note that \( d/2n \approx t_a/n \), where \( t_a = [(d-1)/2] \) is
the number of additive errors that the code can correct. The Plotkin
bound holds for nonlinear codes also.

4.1.3. The Varshamov-Gilbert Lower Bound

The Varshamov-Gilbert bound is a lower bound on the number of
information symbols, \( k \), of an \((n,k)\) linear code with minimum distance
\( d \). That is, it is always possible to find an \((n,k)\) code having at
least \( k^* \) information symbols, where \( k^* \) is the lower bound. The
Varshamov-Gilbert lower bound states that:

It is always possible to construct an \((n,k)\) linear code with min-
imum distance \( d \), where \( k \) is the largest integer satisfying

\[ 1 + \sum_{i=1}^{d-2} \binom{n-1}{i} (q-1)^i \leq q^{n-k} \cdot \]
Figure 4.1. Bounds on the proportion, $k/n$, of information symbols, for $n$ large, for a $t_a$-additive-error-correcting code.
For the binary case, \( q = 2 \), we have the asymptotic inequality

\[
1 + \sum_{i=1}^{d-2} \binom{n-1}{i} \leq 2^{nH\left(\frac{n+1-d}{n-1}\right)} = 2^{nH\left(\frac{d-2}{n-1}\right)},
\]

where \( H(x) = -x \log_2 x - (1-x) \log_2 (1-x) \). If the relation

\[
(4.2) \quad 2^{nH\left(\frac{d-2}{n-1}\right)} \leq 2^{n-k}
\]

holds, then (4.1) certainly holds, so that there exists a binary \((n,k)\) linear code with minimum distance \(d\) if \(k\) satisfies (4.2). But (4.2) is equivalent to

\[
k \leq n \left[ 1 - H\left(\frac{d-2}{n-1}\right) \right],
\]

or, for \(d\) and \(n\) sufficiently large,

\[
(4.3) \quad k \leq n \left[ 1 - H\left(\frac{d}{n}\right) \right].
\]

Thus the largest \(k\) satisfying (4.3) is the asymptotic lower bound for given \(d\) and \(n\) in the binary case. This asymptotic bound is shown in Figure 4.1, where

\[
\frac{k}{n} = 1 - H\left(\frac{d}{n}\right)
\]

is plotted versus \(d/2n\).

4.1.4. The Hamming Upper Bound

Another upper bound is the Hamming bound, or sphere-packing bound. The Hamming bound states that the number of information symbols, \(k\), of an \((n,k)\) linear code with minimum distance \(d = 2t + 1\) must satisfy
\[(4.4) \quad k \leq n - \log \left[ 1 + \sum_{i=1}^{t_a} \frac{n_i}{q} (q - 1)^i \right].\]

An asymptotic form for the Hamming bound in the binary case is

\[(4.5) \quad k \leq n \left[ 1 - H\left(\frac{t_a}{n}\right) \right] \]

and this asymptotic bound is also shown in Figure 4.1, where

\[\frac{k}{n} = 1 - H\left(\frac{d}{2n}\right) \approx 1 - H\left(\frac{t_a}{n}\right)\]

is plotted versus \(d/2n\).

The Hamming bound gives an upper bound on the number of words in
an \((n,k)\) linear code with minimum distance \(d\), since \((4.4)\) implies

\[(4.6) \quad q^k \leq \frac{q^n}{1 + \sum_{i=1}^{t_a} \frac{n_i}{q} (q - 1)^i},\]

and \(q^k\) is the number of words in an \((n,k)\) linear code.

It is noted that the Hamming bound applies to nonlinear as well
as linear codes.

4.1.5. The Elias Bound

Shannon, Gallager, and Berlekamp [48] derive an upper bound,
proved by Elias in 1960 but still unpublished, that is asymptotically
uniformly stronger (i.e., lower) than either the Hamming or Plotkin
bounds. The asymptotic form for the bound is as follows:

\[\frac{d}{2n} \leq \lambda \left(1 - \lambda\right)\]

where \(\lambda\) is given by

\[H(\lambda) = 1 - \frac{k}{n}, \quad 0 \leq \lambda \leq \frac{1}{2} .\]

This asymptotic form of the bound is included in Figure 4.1.
4.2 The Extended Hamming Bound

We shall now derive an upper bound for the number of information symbols in a certain class of codes that can correct $t_e$ additive errors and $t_s$ synchronization errors. It will be assumed that a vector $c^*$ is added to each transmitted word, and subtracted from each received word, as in the technique of the previous chapter. For the purposes of the derivation of the extended Hamming bound, $c^*$ may assume any value desired, including $c^* = 0$. In other words, the proof does not depend on the value of $c^*$, or even on the use of the vector $c^*$. One very important assumption is made, however, in the proof of the bound, namely that, as in the new synchronization technique, $t_s$ of the symbols of the received word are dropped, and the receiver must use the remaining portion of the word for decoding and error correction. For arbitrary parity check structure, such a procedure seems desirable if the first $t_r$ (or the last $t_r$) symbols of codewords are essentially random, since in such a case the ends of words and the ends of overlapped words would possess no consistent structure which the receiver might observe to indicate a synchronization error. If encoding is done canonically, so that $r$ redundant symbols are added to the ends of $k$-sequences of random symbols, then such a situation exists, at least for $t_r \leq r$ (or $t_{\ell} \leq r$). Even though the bound derived is not a universal upper bound, applicable to all synchronization techniques, it is an extension of the ordinary Hamming bound to a case of both additive and synchronization errors that will provide an interesting indication of the efficiency of the present technique.

It is noted that the bounds shown above and the bound to be
derived all depend on the length of the code. The bounds shown above are bounds that would typically apply to additive-error-correcting codes such as $C$, and the symbol $n$ has accordingly been used to denote word length. The bound to be derived, however, applies to additive- and synchronization-error-correcting codes such as $C_T$, and for distinction we shall use the symbol $N$ to denote word length. When comparing the bound to be derived below with the bounds described above, a common symbol, $M$, will be used to denote word length.

**Theorem 4.1. (Extended Hamming Bound)** The number of words, $W$, of a $t_a$-additive-error-correcting, $t_l$-left-shift-error-correcting, $t_r$-right-shift-error-correcting (linear or nonlinear) code of length $N$ must satisfy

$$W \leq \frac{q^{N-t_s}}{t_a^{t_s} \cdot (t_s+1)[1+\sum_{i=1}^{i=t_s}(i^s)(q-1)]}, \quad (t_a \leq N - t_s),$$

where $t_s = t_l + t_r$, if the receiver ignores the first $t_r$ and the last $t_l$ symbols of each received word.

**Proof.** Suppose that $x$ is a vector in the code, which we will denote by $C$. (The code $C_T$ of the preceding chapter corresponds to this code $C$.) If $x$ is sent, then we wish to be able to correct received words of the following forms, without delay (i.e., without examining following words):

- additive errors only
  $$\begin{align*}
  (x_1, x_2, x_3, \ldots, x_{t_r}, x_{t_r+1}, \ldots, x_{N-t_l}, x_{N-t_l+1}, \ldots, x_{N-2}, x_{N-1}, x_N) \\
  + e_N
  \end{align*}$$
additive errors and right-shift errors

\[
(x, x_1, x_2, \ldots, x_{t_r-1}, x_t - 1, x_{N-t}\ldots, x_{N-3}, x_{N-2}, x_{N-1}) + \gamma(-1) + e_N
\]

\[
(x, x_1, x_2, \ldots, x_{t_r-2}, x_{t_r-1}, x_{N-t-2}\ldots, x_{N-4}, x_{N-3}, x_{N-2}) + \gamma(-2) + e_N
\]

\[\vdots\]

\[
(x_{xx}, x_1, \ldots, x_{N-t-2}, x_{N-t-1}, x_{N-t-3}, x_{N-t-2}, x_{N-t}) + \gamma(-t_r) + e_N
\]

additive errors and left-shift errors

\[
(x_2, x_3, x_4, \ldots, x_{t_r+1}, x_{t_r+2}, \ldots, x_{N-t}\ldots, x_{N-1}, x_n) + \gamma(1) + e_N
\]

\[
(x_3, x_4, x_5, \ldots, x_{t_r+2}, x_{t_r+3}, \ldots, x_{N-t+2}, x_{N-t+3}, x_{N}, xx) + \gamma(2) + e_N
\]

\[\vdots\]

\[
(x_{t_r+1}, x_{t_r+2}, x_{t_r+3}, \ldots, x_{t_r+t_s}, x_{t_r+t_s+1}, \ldots, x_n, \ldots, xx) + \gamma(t_s) + e_N
\]

Here \( e_N \) is used generically to denote any correctable additive-error pattern of length \( N \), i.e., \( w(e_N) \leq t_a \), and \( \gamma(i) \) is given by

\[
\gamma(i) = \bar{c}(i) - \bar{c}^*.
\]

As discussed prior to the statement of the theorem, correction of the received words is made using just the central portions (consisting of \( N - t_s \) symbols) of the received words. That is, we consider the vectors:

additive errors only

\[
(4.3.1) \quad (x_{t_r+1}, \ldots, x_{N-t}) + e_{N-t_s}
\]

additive errors and right-shift errors

\[
(4.3.2) \quad (x_{t_r}, \ldots, x_{N-t-1}) + \gamma(-1) + e_{N-t_s}
\]
\[(4.3) \quad (x_{t_r-1}, \ldots, x_{N-t_r+2}) + \gamma(-1) + e_{N-t_s}\]
\[
\vdots
\]
\[(4.8.1 + 1) \quad (x_1, \ldots, x_{N-t_r+1}) + \gamma(-t_r) + e_{N-t_s}\]

additive errors and left-shift errors
\[(4.8.2) \quad (x_{t_r+1}, \ldots, x_{N-t_r+1}) + \gamma(1) + e_{N-t_s}\]
\[(4.8.3) \quad (x_{t_r+2}, \ldots, x_{N-t_r+2}) + \gamma(2) + e_{N-t_s}\]
\[
\vdots
\]
\[(4.8.1 + 1) \quad (x_{t_r+t_r+1}, \ldots, x_N) + \gamma(t_r) + e_{N-t_s}\]

In every case, \(w(e_{N-t_s}) \leq t_a\). Note that there are
\[
E = 1 + (1^S) (q^-1) + (2^S) (q^-1)^2 + \ldots + (t^S_a) (q^-1)^{t_a}
\]
possible correctable additive-error patterns, which we shall denote by \(e^i\), for \(i = 1, 2, \ldots, E\).

Now in order to correct the synchronization error (without delay) and the additive error, it is necessary that all of the \(E(t_s+1)\) vectors represented by \((4.8.1)\) through \((4.8.1+1)\) be different. Thus, if \(x\) is in the code \(C\), then it is necessary that no other code vector have a central portion equal to any of the above \(E(t_s+1)\) vectors. But there are \(q^s\) \(N\)-vectors with a given central portion of \(N-t_s\) symbols.

So, if \(x\) is in the code, there are
\[(4.9) \quad (t_s+1) E q^s - 1\]
different \(N\)-vectors that cannot be in the code. (The term \(-1\) corresponds to the vector \(x_0\).) Suppose now that another vector \(x_o\) is in the code. Then none of the \(E(t_s+1)\) central portions corresponding to \(x_0\) can be equal to any of the \(E(t_s+1)\) central portions corresponding
to $x_0$ (or else we would not know whether to correct the received vector as $x$ or as $x_0$). Thus if $x_0$ is in the code, there are
\[(t_s+1) E q^s - 1\]
different $N$-vectors, different from those of (4.9), that cannot be in the code.

In general, if there are $W$ vectors in the code, then there are
\[W [(t_s+1) E q^s - 1]\]
different $N$-vectors that cannot be in the code. Since the total number of possible $q$-ary $N$-vectors is $q^N$, we have
\[
\text{(no. of $N$-vectors in code)} + \text{(no. of $N$-vectors ruled out of code)} \leq q^N
\]
or
\[W + W[(t_s+1) E q^s - 1] \leq q^N\]
giving
\[W \leq \frac{q^N}{(t_s+1) E q^s},
\]
so
\[W \leq \frac{q^{N-t_s}}{(t_s+1) E q^s}.
\]

This completes the proof.

Note that in the above proof we can set $\gamma(i) = 0$ for all $i$ if we desire. The purpose of $\gamma$ is to facilitate making the $E(t_s+1)$ vectors of the $t_s+1$ sets corresponding to one codeword different from the $E(t_s+1)$ vectors of the $t_s+1$ sets corresponding to another $x$. If we wish to find codes which approach the upper bound, it appears that we would have to choose $\gamma(i) \neq 0$ if $i \neq 0$. This is in fact what we have done in the synchronization technique developed in the preceding chapter.
Corollary 4.1.1. The number of information symbols, $k$, in a $t_a$-additive-error-correcting, $t_i$-left-shift-error-correcting, $t_r$-right-shift-error-correcting (linear or nonlinear) code of length $N$ must satisfy

\[(4.10) \quad k \leq N - t_s - \log_q(t_s + 1) - \log_q \left[ 1 + \sum_{i=1}^{t_a} (N-t_s)(q-1)^i \right], (t_a < N-t_s).\]

**Proof.** The number of words in an $(N,k)$ code is $q^k$. Substituting this for $W$ in the above theorem, we have

\[q^k \leq \frac{q^{N-t_s}}{(t_s + 1) \left[ 1 + \sum_{i=1}^{t_a} (N-t_s)(q-1)^i \right]}.
\]

Taking logarithms to the base $q$ gives the desired results.

Note that if $t_s = 0$, i.e., if it is desired to correct no synchronization errors, then the expression $(4.10)$ is the ordinary Hamming bound.

**Corollary 4.1.2.** For sufficiently large $N$, and for $q=2$, the right hand side of $(4.10)$ becomes approximately

\[(4.11) \quad N-t_s \cdot \log_2(t_s + 1) - (N-t_s) H\left(\frac{t_a}{N-t_s}\right).\]

where

\[H(x) = -x \log_2 x - (1-x) \log_2 (1-x).\]

**Proof.** Using Peterson's notation, ([43], p.52), define

\[R_H(n,\lambda) = \frac{1}{n} \log_2 \left[ 1 + \binom{n}{1} \lambda + \binom{n}{2} \lambda^2 + \ldots + \binom{n}{\lambda n} \right], 0 < \lambda < 1.\]

Then Corollary 4.1.1 implies that

\[1 - \frac{k + \log_2(t_s + 1)}{N-t_s} \geq R_H(N-t_s,\lambda),\]
where
\[ \lambda = \frac{t_a}{N-t_s}. \]

Peterson ([43], p. 52) shows that
\[ B_H(\lambda) = \lim_{n \to \infty} B_H(n,\lambda) = H(\lambda) = H\left(\frac{t_a}{N-t_s}\right). \]

Hence the above inequality becomes, asymptotically,
\[ 1 - \frac{k + \log_2(t_s + 1)}{N-t_s} \geq H\left(\frac{t_a}{N-t_s}\right), \]

so that, asymptotically,
\[ k \leq N-t_s - \log_2(t_s+1) - (N-t_s) H\left(\frac{t_a}{N-t_s}\right), \]

which was to be proved.

For \( t_s \) large (say > 64), the term \( \log_2(t_s + 1) \) in (4.11) can be ignored. The term \( \log_2(t_s + 1) \) can be ignored even for \( t_s \) small, provided \( t_s \) is negligible compared to \( N \), for in this case the term \( \frac{t_s + \log_2(t_s + 1)}{N} \) is negligible. Thus if either \( t_s \) is small compared to \( N \), or \( t_s \) is significant compared to \( N \), the term \( \log_2(t_s + 1) \) can be ignored for \( N \) large. In this case, (4.11) becomes
\[ (4.12) \quad (N - t_s) \left( 1 - H\left(\frac{t_a}{N-t_s}\right) \right), \]

so that for sufficiently large \( N \), in the case \( q = 2 \), the extended Hamming bound becomes
\[ (4.13) \quad k \leq (N-t_s) \left( 1 - H\left(\frac{t_a}{N-t_s}\right) \right). \]

It is of interest to compare this bound with the asymptotic form of the ordinary Hamming bound (4.5).
In Figure 4.2 we have plotted

\[ \frac{k}{N} = \left(1 - \frac{t_s}{N}\right) \left(1 - H\left(\frac{t_a}{N-t_s}\right)\right) \]

versus \( t_a/N \) for \( t_s = fN, f = 0, 1, 2, \ldots, 9, (N-1)/N \). The case \( f = 0 \)
is, of course, the ordinary Hamming bound.

Observe that as \( t_s \to N - 1 \) (assuming that \( t_s \leq N - t_a \)), \( k \to 0 \).
Thus the capability of the new synchronization technique (which is, of
course, bounded by the extended Hamming bound) asymptotically vanishes
as \( t_s \to N - 1 \). (In fact, we can observe directly from (4.7) that as
\( t_s \to N - 1 \), \( W \) is bounded by \( 1/N \); since \( W \) is an integer, this implies
\( W = 0 \).) This is true for any choice of \( q \). But it is known (Theorem
7 of (22)) that for a comma-free code,

\[ n - \log_q e n \leq k \leq n - \log_q n \]
as \( q \to \infty \), where \( e \) is the base of natural logarithms. This result
implies that, for \( t_s \) large compared to \( N \), there exist better synchro-
nization techniques than the technique introduced here.

It should be observed that, with the new technique, we have
\( N = n + L \), where \( L = t_s + \deg[m(x)] \). Thus, with the new technique,
we would choose \( L \) at most equal to \( n - 1 \), and so \( t_s \) would at most be of
the order of \( N/2 \) for large \( N \). Thus the lowest extended Hamming bound
curve in Figure 4.2 that applies to the new method is the one with
\( f = .5 \).

For large values of \( t_s \), the extended Hamming bound seems to be
discouraging evidence regarding the worth of the new technique. A
more general version of the extended Hamming bound would be useful to
indicate how much better other synchronization techniques might be for
large \( t_s \). The following section indicates that for values of \( t_s \) that
Figure 4.2. Bounds on the proportion, $k/N$, of information symbols, for $N$ large, for a $(t_s, t_e)$-error-correcting code, if the receiver deletes $t_s$ symbols from each received word.
are small compared to \( N \), the new technique appears to be an efficient one.

4.3. Comparison of the Ordinary and the Extended Hamming Bounds

The asymptotic expression (4.5) for the ordinary Hamming bound (binary case) can be written

\[
k \leq M - M H(\frac{t_s}{M})
\]

(4.14)

where \( M = n \) is the length of the code. Similarly, the asymptotic expression (4.11) for the extended Hamming bound (binary case) can be written as

\[
k \leq (M-t_s) \left( \frac{H(\frac{t}{M-t_s})}{M-t_s} \right) + (t_s + \log_2(t_s+1)),
\]

(4.15)

where \( M = N \) is the length of the code. If \( t_s \) is small with respect to \( M \), we can write (4.15) as, approximately,

\[
k \leq M - M H(\frac{t_s}{M}) - (t_s + \log_2(t_s+1))
\]

(4.16)

We observe that the bounds (4.14) and (4.16) differ in the term \( t_s + \log_2(t_s+1) \). This term approximately represents the number of information symbols that were sacrificed in order to correct the \( t_s \) synchronization errors. In other words, to correct \( t_s \) synchronization errors, increase the redundancy by approximately \( t_s + \log_2(t_s+1) \) symbols. This result is an extremely interesting one, since \( t_s + \log_2(t_s+2) \) is the number of redundant symbols added in the new synchronization technique in the case where \( e = 2^u - 1 \) and \( t_s = e - 1 \) (its maximum value, given \( e \)). (See Corollary 3.1.2) Even though this obser-
vation is made on approximate results based on asymptotic upper bounds, it is felt that it is an indication that the new synchronization-error-correcting technique is in a sense an optimal one for \( M \) large and \( t_s \) small with respect to \( M \), in the class of all synchronization techniques which delete \( t_s \) symbols of the received words. Since for \( t_s \) small compared to \( M \), the extended Hamming bound derived here would be only slightly lower than an upper bound for all synchronization techniques, it would appear that the new technique is an efficient one, compared to any other synchronization technique, for \( M \) large and \( t_s \) small with respect to \( M \).

Thus, given a BCH \( t_a \)-additive-error-correcting code (which of course may be far from optimum with respect to the redundancy used for additive-error correction), then for \( M \) large and \( t_s \) small compared to \( M \), the new technique appears to be a very efficient procedure by which synchronization errors can be corrected. Of course, for \( t_s \) large compared to \( M \), the technique is inefficient, and the number of redundant symbols is large compared to \( M \). It would appear that the new technique derives its efficiency more from the use of the subcode than from the augmentation of the subcode.

Table 4.1 shows approximately how many information symbols must be sacrificed (asymptotically) in order to correct a specified number, \( t_s \), of synchronization errors. Also included, in the last column, is the number actually required in the case where we are using a BCH code of length 255. The derivation of these numbers is not discussed here, but will be considered in Chapter V. Generally, \( m + t_s \) symbols are sacrificed using the new technique if \( t_s \) synchronization errors are to be corrected, where \( m \) is as in the expression \( GF(2^m) \). The number of
<table>
<thead>
<tr>
<th>Number of correctable synchronization errors $t_s = t_s + t_r$</th>
<th>Actual number of additional redundant symbols, using a BCH code with length $2^{25}$ $= t_s + \deg[m(x)]$</th>
<th>Minimum possible number of additional symbols using the new method $= t_s + n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

TABLE 4.1. REDUNDANCY OF THE NEW SYNCHRONIZATION TECHNIQUE CORRESPONDING TO SPECIFIED SYNCHRONIZATION-ERROR CORRECTION CAPABILITY (BINARY CASE)
<table>
<thead>
<tr>
<th>Number of correctable synchronization errors</th>
<th>Asymptotic number of additional redundant symbols (from upper bound)</th>
<th>Minimum possible number of additional redundant symbols using the new method</th>
<th>Actual number of additional redundant symbols, using a BCH code with length 255</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>26 = [25.46]</td>
<td>26</td>
<td>29</td>
</tr>
<tr>
<td>22</td>
<td>27 = [26.52]</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>23</td>
<td>28 = [27.58]</td>
<td>28</td>
<td>31</td>
</tr>
<tr>
<td>24</td>
<td>29 = [28.64]</td>
<td>29</td>
<td>32</td>
</tr>
<tr>
<td>25</td>
<td>30 = [29.70]</td>
<td>30</td>
<td>33</td>
</tr>
</tbody>
</table>

1. The number of "additional redundant symbols" is the number of redundant symbols added for synchronization purposes, beyond the number of redundant symbols used for additive-error correction.

2. The notation \([x]\) is used here to denote the least integer not less than \(x\).

3. The column entry is obtained by assuming that, if \(2^{u-1} - 2 < t_s \leq 2^u - 2\), then there exists a minimum function \((m(x))\) of \(z\) with degree \(u\) and order \(2^u - 1\). See Corollary 3.1.2 and the discussion following that corollary for additional explanation.

4. The \(m(x)\) used to obtain the column entries is the minimum function \((n = 255)\) of least degree having order \(\geq t_s + 1\).

5. The redundancies in parentheses correspond to a situation discussed in Chapter V, in which \(m(x)\) is the product of minimum functions. All other column entries correspond to the case in which \(m(x)\) is irreducible.
information symbols which must be sacrificed in a BCH code if \( t_a \) additive errors are to be corrected is generally \( mt_a \). Thus for a BCH code of length 255, eight symbols must generally be sacrificed for each additional additive error it is desired to correct. The comparison of the additive quantity \( m + t_s \) for synchronization errors with the multiplicative quantity \( mt_a \) for additive errors illustrates the fact that there are \( (\frac{t}{t_a}) \) different additive errors of \( t_a \) symbols (binary case), but only one synchronization error of specified order.

It is noted that if \( t_a = t_r = t \), say, so that \( t_s = 2t \), then an upper bound to \( t \), using the new technique, is \( t \leq (N - k - u)/2 \), where \( u = \deg [m(x)] \). (This result follows by taking \( t_a = 0 \), corresponding to \( g(x) = 0 \), \( g_s(x) = m(x) \), so that all of the redundancy is for synchronization purposes. In addition to the \( u \) redundant symbols corresponding to \( m(x) \), \( t_s \) redundant symbols are required if \( t_s = 2t \). Thus the redundancy \( N-k \) must exceed \( 2t + u \).) The greatest value for this upper bound for the new technique corresponds to \( u = 2 \)(restricting \( t_s \leq p^u - 1 = 3 \) in the binary case, \( p = 2 \).), in which case we have \( t \leq (N - k - 2)/2 \). Stiffler[50] (and Tong[55]), have proved that, in general, \( t \leq (N - k - 1)/2 \).
CHAPTER V

EXAMPLE ILLUSTRATING THE NEW SYNCHRONIZATION TECHNIQUE

5.1 Introduction

This chapter will present examples illustrating the use of the synchronization technique developed in Chapter IV. Three examples will be discussed, showing the ways in which the techniques can be used. The first example includes an explicit illustration of the error-correction procedure by considering an actual synchronization error. In order to enable the computations to be easily presented, the code used for that example is of very small length. The small length, together with high redundancy, render the code useless for practical purposes, and it is emphasized that the particular code used is chosen solely on the basis of its illustrative advantages. The other examples include illustrations of the different levels of possible synchronization-error correction corresponding to various choices of $z = \alpha^s$, for a practically useful code. A table is presented which provides the information necessary to enable application of the technique to any desired situation in which the code length is less than or equal to 255 and the code symbols are binary digits. Also included in this chapter is a method of using a reducible factor $m(x)$, enabling a decrease in the synchronization redundancy for some ranges of $t_s$. 
5.2. Table for Constructing \((t_s, t_g)\) -Error-Correcting Bose-Chaudhuri Codes, for \(p = 2\) and \(n \leq 255\)

Before giving examples of the procedure by which the synchronization technique is applied to BCH codes, we shall present a table, derived from tables in [43], providing the information to enable the construction of any \((t_a, t_g)\) -error-correcting BCH binary code of length not greater than 255, that is based on a primitive element. The table, Table 5.1, is divided into sections corresponding to the value of \(m\) (in the expression \(\text{GF}(2^m)\)) or, equivalently, the value of \(n = 2^m - 1\) (columns 1 and 2). The minimum function of the primitive element powers located in the third column is given in the fourth column. The sixth column then indicates the maximum value of \(t_s\) which is possible, using the corresponding minimum function for \(m(x)\) in the definition of \(g_\alpha(x)\) (i.e., \(g_\alpha(x) = g(x)m(x)\)). In each section, the minimum function corresponding to \(\alpha\) is the polynomial on which \(\text{GF}(2^m)\) is based. The order of every root of a minimum function is the same, and this order is given in the fifth column. Since \(\alpha\) is a primitive element of \(\text{GF}(2^m)\), the order \(e_i\) of \(\alpha^i\) is given by

\[
e_i = \frac{2^m-1}{\text{GCD}(2^m-1, i)}
\]

If \(u\) is the degree of the minimum function of \(\alpha^i\), then the order of \(\alpha^i\) is at most \(p^u - 1 = 2^u - 1\). The entry in the sixth column is simply one less than the order. Since the minimum functions of \(\alpha, \alpha^2, \alpha^5, \ldots\), are listed in order in the table, \(g(x)\) will be of the form

\[
g(x) = m_1(x) m_2(x) \ldots m_u(x),
\]
<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$\alpha$, $\alpha^2$, $\alpha^2^2$, $\alpha^2^3$, ...</th>
<th>$m_1(x)$</th>
<th>$t^4_a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>$1,2$ (i.e., the powers are $\alpha, \alpha^2$)</td>
<td>$x^2 + x + 1$</td>
<td>3 2 1</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$1,2,4$</td>
<td>$x^3 + x^4 + 1$</td>
<td>7 6 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$3,6,5$</td>
<td>$x^2 + x^3 + 1$</td>
<td>7 6 3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>$1,2,4,8$</td>
<td>$x_4^4 + x^5 + 1$</td>
<td>15 14 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$3,6,12,9$</td>
<td>$x_2^2 + x^3 + x^4 + 1$</td>
<td>5 4 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$5,10$</td>
<td>$x_4^4 + x^5 + 1$</td>
<td>5 2 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$7,14,13,11$</td>
<td>$x^5 + x^6 + 1$</td>
<td>15 14 7</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>$1,2,4,8,16$</td>
<td>$x_5^5 + x^2 + 1$</td>
<td>31 30 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$3,6,12,24,17$</td>
<td>$x_2^2 + x^3 + x^4 + x^1 + 1$</td>
<td>31 30 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$5,10,20,9,18$</td>
<td>$x_5^5 + x^2 + 1$</td>
<td>31 30 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$7,14,28,25,19$</td>
<td>$x_2^2 + x^3 + x^4 + x^1 + 1$</td>
<td>31 30 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$11,22,13,26,21$</td>
<td>$x^5 + x^2 + x + 1$</td>
<td>31 30 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$15,30,29,27,25$</td>
<td>$x^5 + x + 1$</td>
<td>31 30 15</td>
</tr>
</tbody>
</table>

1 Only the exponents, $1, 2i, 2^2i, 2^3i, ...$, are given.
2 Order of $\alpha^i =$ exponent to which $m_1(x)$ belongs.
3 Column entry is maximum possible value of $t^s_a$, using $m(x) = m_1(x)$.
4 $t^4_a$ for a BCH code with generator $g(x) = \prod_{j=1}^{4} m_j(x)$.
<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>( \alpha )</th>
<th>Minimum function</th>
<th>Order of ( \alpha )</th>
<th>Maximum value of ( t_s )</th>
<th>( t_a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>63</td>
<td>1, 2, 4, 8, 16, 32</td>
<td>( x_6^2 + x_5^2 + x_4 + x + 1 )</td>
<td>63</td>
<td>62</td>
<td>1</td>
</tr>
<tr>
<td>5, 10, 20, 40, 57, 34</td>
<td>( x_6^2 + x_5^2 + x_4 + x + 1 )</td>
<td>21</td>
<td>20</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7, 14, 28, 56, 49, 35</td>
<td>( x_6^2 + x_5 + x_4 + x + 1 )</td>
<td>63</td>
<td>62</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9, 18, 36</td>
<td>( x_6^2 + x_5^2 + x_4 + x + 1 )</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11, 22, 44, 25, 50, 37</td>
<td>( x_6^2 + x_5 + x_4 + x_3 + x + 1 )</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13, 26, 52, 41, 19, 38</td>
<td>( x_6^2 + x_5 + x_4 + x_3 + x_2 + x + 1 )</td>
<td>63</td>
<td>62</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15, 30, 60, 57, 51, 39</td>
<td>( x_6^2 + x_5 + x_4 + x_3 + x_2 + x + 1 )</td>
<td>21</td>
<td>20</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21, 42</td>
<td>( x_6^2 + x_5 + x_4 + x + 1 )</td>
<td>63</td>
<td>62</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23, 46, 29, 58, 53, 43</td>
<td>( x_6^2 + x_5 + x_4 + x + 1 )</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27, 54, 45</td>
<td>( x_6^2 + x_5 + x_4 + x + 1 )</td>
<td>63</td>
<td>62</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31, 62, 61, 59, 55</td>
<td>( x_6^2 + x_5 + x_4 + x + 1 )</td>
<td>7</td>
<td>6</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>1, 2, 4, 8, 16, 32, 64</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>1</td>
</tr>
<tr>
<td>3, 6, 12, 24, 48, 96, 65</td>
<td>( x_7^3 + x_6 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 10, 20, 40, 80, 33, 66</td>
<td>( x_7^3 + x_6 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7, 14, 28, 56, 112, 97, 67</td>
<td>( x_7^3 + x_6 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9, 18, 36, 72, 17, 34, 68</td>
<td>( x_7^3 + x_6 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11, 22, 44, 88, 49, 98, 69</td>
<td>( x_7^3 + x_6 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13, 26, 52, 104, 81, 35, 70</td>
<td>( x_7^3 + x_6^2 + x_5 + x + 1 )</td>
<td>127</td>
<td>126</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15, 30, 60, 120, 113, 99, 71</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19, 38, 76, 25, 50, 100, 73</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21, 42, 84, 41, 82, 37, 74</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23, 46, 92, 57, 114, 101, 75</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27, 54, 108, 89, 51, 102, 77</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29, 58, 116, 105, 85, 59, 78</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31, 62, 124, 121, 115, 105, 79</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43, 86, 45, 90, 53, 106, 85</td>
<td>( x_7^3 + x_6^2 + x_5 + x_4 + 1 )</td>
<td>127</td>
<td>126</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 5.1. -- Continued**
<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Powers of α</th>
<th>Minimum function</th>
<th>Order of α²</th>
<th>Maximum value of t₂</th>
<th>t₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>118</td>
<td>47, 94, 61, 122, 117, 107, 87</td>
<td>$x_1^3 + x_5 + x_6 + x_7 + 1$</td>
<td>118</td>
<td>118</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>(continued)</td>
<td>55, 110, 93, 59, 118, 109, 91</td>
<td>$x_1^7 + x_1^4 + x_3 + x_7 + 1$</td>
<td>127</td>
<td>126</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>65, 126, 125, 123, 119, 111, 95</td>
<td>$x_1 + 1$</td>
<td>127</td>
<td>126</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>1, 2, 4, 8, 16, 32, 64, 128</td>
<td>$x_8^1 + x_1^3 + x_2 + 1$</td>
<td>255</td>
<td>254</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3, 6, 12, 24, 48, 96, 192, 129</td>
<td>$x_8^8 + x_6 + x_9 + 1$</td>
<td>85</td>
<td>84</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5, 10, 20, 40, 80, 160, 65, 130</td>
<td>$x_8^8 + x_8 + x_5 + x_7 + 1$</td>
<td>51</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7, 14, 28, 56, 112, 224, 193, 131</td>
<td>$x_8^8 + x_9 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9, 18, 36, 72, 144, 33, 66, 132</td>
<td>$x_8^8 + x_6 + x_7 + 1$</td>
<td>85</td>
<td>84</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11, 22, 44, 88, 176, 97, 194, 133</td>
<td>$x_8^8 + x_1^4 + x_9 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13, 26, 52, 104, 208, 161, 67, 134</td>
<td>$x_8^8 + x_6 + x_1^4 + x_9 + 1$</td>
<td>255</td>
<td>254</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15, 30, 60, 120, 240, 225, 195, 135</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>17</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17, 34, 68, 136</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>15</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19, 38, 76, 152, 39, 98, 196, 137</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21, 42, 84, 168, 61, 162, 69, 138</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>85</td>
<td>84</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23, 46, 92, 184, 113, 226, 197, 139</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25, 50, 100, 200, 145, 35, 70, 140</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>51</td>
<td>50</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27, 54, 108, 216, 177, 99, 198, 141</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29, 58, 116, 232, 209, 165, 71, 142</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31, 62, 124, 248, 214, 227, 199, 143</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37, 74, 148, 41, 82, 164, 73, 146</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39, 78, 156, 57, 114, 228, 201, 147</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>85</td>
<td>84</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43, 86, 172, 89, 176, 101, 202, 149</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45, 90, 180, 105, 210, 165, 75, 150</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>17</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>47, 94, 188, 121, 242, 229, 205, 151</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>51, 102, 204, 153</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>5</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53, 106, 212, 169, 83, 166, 77, 154</td>
<td>$x_8^8 + x_6 + x_1^2 + x_7 + 1$</td>
<td>255</td>
<td>254</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(continued)</td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>

TABLE 5.1 -- Continued
<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Powers of α</th>
<th>Minimum function</th>
<th>Order of α</th>
<th>Maximum value of t</th>
<th>t</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>255</td>
<td>55,110,220,185,115,230,205,155</td>
<td>(x_8^7 + x_6^5 + x_2 + 1)</td>
<td>51</td>
<td>50</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>59,118,236,217,179,103,206,157</td>
<td>(x_8^7 + x_6^5 + x_2 + 1)</td>
<td>255</td>
<td>254</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>61,122,244,233,211,167,79,158</td>
<td>(x_8^6 + x_6^5 + x_4 + x_3 + x_2 + 1)</td>
<td>255</td>
<td>254</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>63,126,252,249,243,231,207,159</td>
<td>(x_2 + x + x + x + x + x + 1)</td>
<td>85</td>
<td>84</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>85,170</td>
<td>(x_8^6 + x_2 + 1)</td>
<td>3</td>
<td>2</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>87,174,93,186,117,234,213,171</td>
<td>(x_8^6 + x_6^5 + x_2 + 1)</td>
<td>85</td>
<td>84</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>91,182,109,218,181,107,214,173</td>
<td>(x_8^6 + x_6^5 + x_4 + x_2 + x + 1)</td>
<td>255</td>
<td>254</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>95,190,125,250,245,235,215,175</td>
<td>(x_8^6 + x_6^5 + x_4 + x_2 + x + 1)</td>
<td>51</td>
<td>50</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>111,222,189,123,246,237,219,183</td>
<td>(x_4 + x + x + x + x + x + 1)</td>
<td>85</td>
<td>84</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>119,238,221,187</td>
<td>(x_8^6 + x_2 + 1)</td>
<td>15</td>
<td>14</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>127,254,253,251,247,239,223,191</td>
<td>(x + x + x + x + 1)</td>
<td>255</td>
<td>254</td>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>
where the successive factors are taken in order from the table. (See [49] for a table of generators of BCH codes.) Thus, \( m(x) \) will always be a minimum function which occurs after the minimum functions \( m_1(x), m_3(x), \ldots, m_u(x) \) in the table. The last entry (the seventh) of any given row of the table gives the value of \( t_a \) which corresponds to choosing \( u = i \) (i.e., \( g(x) = m_1(x) m_3(x) \ldots m_i(x) \)), where \( i \) is the first exponent of \( \alpha \) listed (in the third column) in the given row.

5.3. Application of the Technique to A BCH Binary Code of Length \( n = 15 \)

**Example 5.1. The Case \( n = 15 \).**

To illustrate the new synchronization technique, we shall consider the BCH code of length \( n = 2^m - 1 = 2^4 - 1 = 15 \) which was presented in Bose and Ray-Chaudhuri's original paper [4]. Over the coefficient field \( GF(2) \), we see from Table 5.1 with \( m = 4 \) that we have the factorization

\[
x^{15} - 1 = m_1(x) m_3(x) m_5(x) m_7(x) (x+1)
\]

\[
= (x^4 + x^1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1)(x+1)
\]

and we choose

\[
g(x) = m_1(x) m_3(x)
\]

\[
= (x^4 + x^1)(x^4 + x^3 + x^2 + x + 1)
\]

\[
= 1 + x^1 + x^6 + x^7 + x^8
\]

for the generator polynomial of the code \( C \). The code \( C \) is an \((n,k) = (15, 7)\) code. We shall assume that the Galois field \( GF(2^4) \) is based on the primitive polynomial \( m_1(x) = x^4 + x + 1 \). All the nonzero elements of \( GF(2^4) \) can thus be written as powers of the root \( \alpha = (0, 1, 0, 0) \) of \( m_1(x) \), and they are shown in Table 5.2. The zero element is, of course, \((0, 0, 0, 0)\). The roots of the primitive
\textbf{TABLE 5.2.}

NONZERO ELEMENTS OF $\text{GF}(2^4)$, EXPRESSED AS POWERS OF THE ROOT $\alpha = (0,1,0,0)$ OF THE MINIMUM FUNCTION $m_1(x) = x^4 + x + 1$ \footnote{The polynomial expression for each power of $\alpha$ is obtained by using the relation $\alpha^3 = x$, $x^3 = x + 1$.}

\begin{align*}
\alpha^0 &= 1 & (1,0,0,0) \\
\alpha &= x & (0,1,0,0) \\
\alpha^2 &= x^2 & (0,0,1,0) \\
\alpha^3 &= x^3 & (0,0,0,1) \\
\alpha^4 &= 1 + x & (1,1,0,0) \\
\alpha^5 &= x + x^2 & (0,1,1,0) \\
\alpha^6 &= x^2 + x^3 & (0,0,1,1) \\
\alpha^7 &= 1 + x + x^3 & (1,1,0,1) \\
\alpha^8 &= 1 + x^2 & (1,0,1,0) \\
\alpha^9 &= x + x^3 & (0,1,0,1) \\
\alpha^{10} &= 1 + x + x^2 & (1,1,1,0) \\
\alpha^{11} &= x + x^2 + x^3 & (0,1,1,1) \\
\alpha^{12} &= 1 + x + x^2 + x^3 & (1,1,1,1) \\
\alpha^{13} &= 1 + x^2 + x^3 & (1,0,1,1) \\
\alpha^{14} &= 1 + x^3 & (1,0,0,1)
\end{align*}
polynomial $m_1(x)$ are
\[ \alpha, \alpha^2, \alpha^4, \alpha^8, \]
and the roots of $m_3(x)$ are
\[ \alpha^3, \alpha^6, \alpha^{12}, \alpha^{24} = \alpha^9. \]
Thus
\[ \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^6, \alpha^8, \alpha^9, \alpha^{12} \]
are all the roots of $g(x)$, and the code is 2-additive-error-correcting, since the first $2t_{sa} = 2 \cdot 2 = 4$ successive powers of $\alpha$ are roots of $g(x)$.

To use the new synchronization technique, we must choose $s \neq 0 \pmod{n}$ such that $\alpha^s$ is not a root of $g(x)$. Equivalently, we must choose a minimum function $m_1(x) \neq x - 1$ such that $m_1(x)$ is not a factor of $g(x)$. Now the factors of $x^{15} - 1$ other than $(x - 1)g(x)$ are $m_3(x)$ and $m_7(x)$. The factor $m_3(x)$ has roots $\alpha^5$ and $\alpha^{10}$, and the factor $m_7(x)$ has roots $\alpha^7$, $\alpha^{14}$, $\alpha^{13}$, and $\alpha^{11}$. Thus we can choose either $s = 5$ or $s = 7$, corresponding to $m(x) = m_3(x)$ or $m_7(x)$. Let us suppose further that we wish to correct a single left-shift or right-shift error, so that $t_f = t_r = 1$, and therefore $t_s = 2$. The final requirement on $s$ is that $t_s \leq e - 1$, where $e$ is the order of $\alpha^s$. (Equivalently, we require that $m(x)$ be such that $t_s \leq e - 1$, where $e$ is the exponent to which $m(x)$ belongs.) Now both $m_3(x)$ and $m_7(x)$ satisfy $2t_{sa} \leq e - 1$, since the order of $\alpha^5$ is 3 and the order of $\alpha^7$ is 15. Thus $m_3(x)$ and $m_7(x)$ are both acceptable choices for $m(x)$. However, in order to add as little redundancy as possible for synchronization purposes, we choose for $m(x)$ the acceptable polynomial of least degree satisfying $t_s \leq e - 1$. Hence we take $m(x) = m_3(x)$. Thus the
subcode $C_5$ has the generator polynomial

$$g_5(x) = g(x) \cdot m_5(x)$$

$$= (1 + x^4 + x^6 + x^7 + x^8) \cdot (x^2 + x + 1)$$

$$= 1 + x + x^2 + x^4 + x^5 + x^6 + x^10$$

and $C_5$ is thus an $(n_5, k_5) = (15, 5)$ code. The subcode $C_5$ has the roots

$$\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{12}.$$  

Such a code, if used solely for additive-error correction, would be a 3-additive-error-correcting BCH code, since the first six successive powers of $\alpha$ are roots of $C_5$. The generator matrix of $C_5$ is given by

$$G_5 = \begin{bmatrix}
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}.$$  

The code $C$ is the null space of the matrix $H^*$ given by

$$H^* = \begin{bmatrix}
1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 \\
1 & \alpha^3 & (\alpha^3)^2 & (\alpha^3)^3 & (\alpha^3)^4 & (\alpha^3)^5 & (\alpha^3)^6 & (\alpha^3)^7 \\
\alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\
(\alpha^3)^6 & (\alpha^3)^9 & (\alpha^3)^{10} & (\alpha^3)^{11} & (\alpha^3)^{12} & (\alpha^3)^{13} & (\alpha^3)^{14}
\end{bmatrix}.$$
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{bmatrix}
\]

We note that \( H^* \) has \( r = n - k = \deg [g(x)] = 8 \) rows, so that the parity check matrix \( H = H^* \). The matrix \( H_2^* \) is given by

\[
H_2^* = \begin{bmatrix}
1 & \alpha^5 & (\alpha^5)^2 & (\alpha^5)^3 & (\alpha^5)^4 & (\alpha^5)^5 & (\alpha^5)^6 & (\alpha^5)^7 \\
& (\alpha^5)^8 & (\alpha^5)^9 & (\alpha^5)^{10} & (\alpha^5)^{11} & (\alpha^5)^{12} & (\alpha^5)^{13} & (\alpha^5)^{14} \\
\end{bmatrix}
\]

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

Since the last row of \( H_2 \) is null, and the second and third rows are identical, we drop the last two rows of \( H_2^* \), and write

\[
H_2 = \begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\]

We note that \( H_S = \begin{bmatrix} \hat{H} \\ H_2 \end{bmatrix} \) has \( r_S = n_S - k_S = \deg[g_S(x)] = 10 \) rows. To encode a \( k_S \)-coordinate information vector \( s = (s_1, s_2, \ldots, s_{k_S}) \) into a codeword of \( C_S \), we correspond the vector \( s \) to the codeword \( s \in C_S \).

(Equivalently, we correspond the information polynomial \( s(x) = \)
\[ s_1 + s_2 x + s_3 x^2 + \ldots + s_{k_S} x^{k_S-1} \] to the code polynomial \( s(x)g_S(x) \).

For example, the vector \((10110)\) is encoded into \((101110)\).

\( G_S = (110010100001110) \). Table 5.3 contains a list of the 32 possible information vectors, \( s \), and the corresponding codewords of \( C_S \).

Since \( t_s = 2 \), the words of the augmented code \( C_A \) are obtained by repeating the first two symbols in each of the 32 words of \( C_S \).

The next step in setting up the procedure is to determine the translation vector \( c_A \). Suppose that we choose \( c = g \), where \( g = (1000101110000000) \) is the coefficient vector of \( g(x) \), considered as a polynomial of degree \( 14 \). Then we have

\[ c_A = (100010111000000010), \]

and we write

\[ C_T = C_A + c_A. \]

The words of the code \( C_T \) are shown in Table 5.4. The code \( C_T \) is an \((n_T, k_T) = (17,5)\) code. It is the words of \( C_T \) that are sent over the channel.

The code \( C_T \) can correct \( t_a = 2 \) additive errors (since \( C \) is a 2-additive-error-correcting BCH code) and \( t_s = 2 \) synchronization errors, and is thus called a \((t_a, t_s) = (2,2)\)-error-correcting code. Synchronization-error correction is accomplished as follows.

First, we calculate all possible proper synchronization-error syndromes which might occur. Since \( t_s = 2 \), there are only \( t_s + 1 = 3 \) proper synchronization-error syndromes. They are given by

\[ [c(t_R + i) - c(t_R)] H_2 \]

where \( i = n - t_R, n - t_R + 1, \ldots, n - 1, 0, 1, 2, \ldots, t_s \), that is,
<table>
<thead>
<tr>
<th>Information vector $s$</th>
<th>Corresponding codeword of $C_s$ $x = sG$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>111011001010000</td>
</tr>
<tr>
<td>01000</td>
<td>011101100101000</td>
</tr>
<tr>
<td>00100</td>
<td>001110110010100</td>
</tr>
<tr>
<td>00010</td>
<td>000111011010010</td>
</tr>
<tr>
<td>00001</td>
<td>000011101100101</td>
</tr>
<tr>
<td>11000</td>
<td>100110101111000</td>
</tr>
<tr>
<td>10100</td>
<td>110101111000100</td>
</tr>
<tr>
<td>10010</td>
<td>111100010011010</td>
</tr>
<tr>
<td>10001</td>
<td>111000100110101</td>
</tr>
<tr>
<td>01100</td>
<td>010011010111100</td>
</tr>
<tr>
<td>01010</td>
<td>011010111110010</td>
</tr>
<tr>
<td>01001</td>
<td>011110001001101</td>
</tr>
<tr>
<td>00110</td>
<td>001001101011110</td>
</tr>
<tr>
<td>00101</td>
<td>001101011110001</td>
</tr>
<tr>
<td>00011</td>
<td>000100110101111</td>
</tr>
<tr>
<td>11100</td>
<td>101000111011000</td>
</tr>
<tr>
<td>11010</td>
<td>100001111011000</td>
</tr>
<tr>
<td>11001</td>
<td>100101000111101</td>
</tr>
<tr>
<td>10110</td>
<td>110010100001110</td>
</tr>
<tr>
<td>10101</td>
<td>110110010100001</td>
</tr>
<tr>
<td>10011</td>
<td>111111111111111</td>
</tr>
<tr>
<td>01110</td>
<td>010100011110110</td>
</tr>
<tr>
<td>01101</td>
<td>010000111011001</td>
</tr>
<tr>
<td>01011</td>
<td>011001010000111</td>
</tr>
<tr>
<td>00111</td>
<td>001010001110001</td>
</tr>
<tr>
<td>11110</td>
<td>101111000100110</td>
</tr>
<tr>
<td>11101</td>
<td>1010111110001001</td>
</tr>
<tr>
<td>11011</td>
<td>100010011010111</td>
</tr>
<tr>
<td>10111</td>
<td>110001001101101</td>
</tr>
<tr>
<td>01111</td>
<td>010111000100111</td>
</tr>
<tr>
<td>11111</td>
<td>101100101000011</td>
</tr>
<tr>
<td>00000</td>
<td>000000000000000</td>
</tr>
<tr>
<td>Table 5.4.</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>CODEWORDS OF THE TRANSLATED CODE ( C_T )</td>
<td></td>
</tr>
<tr>
<td>(The codewords are listed in order corresponding to the words in Table 5.3.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01100111100100001</td>
</tr>
<tr>
<td>11111101110100011</td>
</tr>
<tr>
<td>10110000101010010</td>
</tr>
<tr>
<td>10010110000101010</td>
</tr>
<tr>
<td>1000101010010110</td>
</tr>
<tr>
<td>00010001011100000</td>
</tr>
<tr>
<td>01011100000010001</td>
</tr>
<tr>
<td>01111010101101001</td>
</tr>
<tr>
<td>011010011111010101</td>
</tr>
<tr>
<td>11000110111110011</td>
</tr>
<tr>
<td>11100000100110111</td>
</tr>
<tr>
<td>11110011000110111</td>
</tr>
<tr>
<td>10101101100110110</td>
</tr>
<tr>
<td>10111110011000110</td>
</tr>
<tr>
<td>10011000110111110</td>
</tr>
<tr>
<td>00101010101011000</td>
</tr>
<tr>
<td>00001100111010000</td>
</tr>
<tr>
<td>000111111101110100</td>
</tr>
<tr>
<td>01000011001111001</td>
</tr>
<tr>
<td>01010011011000101</td>
</tr>
<tr>
<td>01110100011111011</td>
</tr>
<tr>
<td>11011011011011011</td>
</tr>
<tr>
<td>11001000011100111</td>
</tr>
<tr>
<td>11111111010001111</td>
</tr>
<tr>
<td>10100011111001110</td>
</tr>
<tr>
<td>00110111111011000</td>
</tr>
<tr>
<td>001001001001000100</td>
</tr>
<tr>
<td>00000011001011100</td>
</tr>
<tr>
<td>01001111010110101</td>
</tr>
<tr>
<td>11010101101001111</td>
</tr>
<tr>
<td>00111001000011000</td>
</tr>
<tr>
<td>10001011100000100</td>
</tr>
</tbody>
</table>
1 = 14, 0, 1, since \( t = t_r = 1 \) and \( n = 15 \). Observe that the rows of \( H_2' \) are 2-vectors. The synchronization-error syndromes will hence also be 2-vectors. For \( i = 14, 0, \) and \( 1 \) the synchronization-error syndromes are

\[
\begin{align*}
[\bar{c}(t_r+14) - \bar{c}(t_r)] H_2' &= [\bar{c}(15) - \bar{c}(1)] H_2' \\
&= [\bar{c}(0) - \bar{c}(1)] H_2' \\
&= [(100010111000000) - (00010111000000)] H_2' \\
&= (100111001000001) H_2' \\
&= (100111001000001) (101101101101101) \\
&= (011011011011111) \\
&= (10); \\
[\bar{c}(t_r) - \bar{c}(t_r)] H_2' &= 0 H_2' \\
&= (0 0); \\
[\bar{c}(t_r+1) - \bar{c}(t_r)] H_2' &= [\bar{c}(2) - \bar{c}(1)] H_2' \\
&= [(001011100000010) - (000101110000001)] H_2' \\
&= (001110010000011) H_2' \\
&= (001110010000011) (101101101101101) \\
&= (011011011011111) \\
&= (1 1).
\end{align*}
\]

Thus, if a left-shift error of order 1 occurs, the synchronization-error syndrome is (11); if a right-shift error of order 1 occurs, the synchronization-error syndrome is (10); and if no synchronization error occurs, the synchronization-error syndrome is (00). Observe that the vector (01) is not a proper synchronization-error syndrome.

Suppose that the source has generated the information vector \( s = (10110) \). The word in \( C_S \) corresponding to \( s = (10110) \) is

\( x = (110010100001110) \), and the corresponding word in \( C_T \) is

\( x_T = (010000011100111001) \). Thus the word \( x_T = (010000011100111001) \) is
sent over the channel. Suppose that a left-shift error of order 1 has occurred, so that if no additive errors occurred, the sequence (10000011001110011) would be received, where we have assumed for definiteness that the first symbol in the word following $x_T$ was 1. In addition to the synchronization error, however, let us suppose that two additive errors occurred, so that the third and twelfth symbols of (010000011100111001), or the second and eleventh symbols of (10000011001110011), were complemented. Thus

$$u = (1000000111001110011) + (01000000001000000)$$
$$= (11000011000110011)$$

is the received word. The receiver subtracts $c_A$ from $u$ to form

$$y = u - c_A$$
$$= (11000011000110011) - (10001011110000010)$$
$$= (01001000100110011)$$

and drops the first and last symbols of $y$ (first $t_r$ and last $t_s$ symbols, with $t_r = t_s = 1$), to obtain

$$y = (100100010011000).$$

The receiver calculates the additive-error syndrome

$$y H' = (1001000100110000) H'$$
$$= (11010111).$$

Since the additive-error syndrome is nonzero, the receiver interprets that an additive error has occurred, and proceeds to correct it. To do this, the receiver would employ a procedure for correcting additive errors for the BCH code $C$, using the syndrome (11010111). (See [3], [9], [15], [37], and [43] for decoding procedures for BCH codes.) Since we are not here directly interested in the additive-error correction procedure, we shall not employ such a procedure here, but
instead note that

\[(100000000100000) H' = (11010111).\]

Thus the receiver would reach the conclusion that the additive-error pattern in \(y\) is

\[e = (1000000000100000).\]

The receiver then calculates the corrected vector

\[\hat{y}_c = \hat{y} - e\]

\[= (100100010011000) - (1000000000100000)\]

\[= (000100010111000)\]

and calculates the synchronization-error syndrome

\[\hat{y}_c H'_2 = (000100010111000) \begin{pmatrix} 1011011011011011 \\ 011011011011011 \end{pmatrix} = (11).\]

The receiver must now identify this nonzero synchronization-error syndrome with a particular synchronization error. Comparing the calculated synchronization-error syndrome with the three proper synchronization syndromes, it is seen that the calculated syndrome corresponds to

\[ [c(t_{r+1}) - c(t_r)] H'_2\]

for \(i = 1\). Thus the receiver interprets that a left-shift error of order 1 has occurred, and moves the word marks one place to the left.

Thus the correctly synchronized received would be

\[\hat{u}_c = (01100001100011001)\]

Since the receiver has already calculated the additive-error vector, he knows the corrected received word to be

\[\hat{u}_c = (01100001100011001) - (00100000000100000)\]

\[= (0100000110011001).\]
Subtracting $c_A$, this becomes
\[ u_c^c - c_A = (010000011100111001) - (10001011100000010) \]
\[ = (11001010000111111) . \]

Dropping the first and last symbols, we have
\[ u_c^c = (10010100001111) . \]

As a check, the receiver may note that the synchronization-error syndrome
\[ X_c^c \cdot H_2^t \]

is indeed now zero. The receiver hence interprets that
\[ x = X_c^c (n - t_r) \]
\[ = X_c^c (15 - 1) \]
\[ = X_c^c (-1) \]
\[ = (1100101000011110) \]

and so
\[ s = (101110), \]

which is correct.

5.4. Application of the Technique to BCH Binary Codes of Lengths

$n = 63$ and $n = 255$

The examples to be presented now will illustrate in less detail the application of the technique in more realistic situations than that of the above subsection, where $n = 15$. That is, the length and redundancy will be such that the codes are practically useful.

Example 5.2. The Case $n = 63$.

Consider the Galois field GF($2^6$) of polynomials of degree less than six, with coefficients over GF(2). Consider a $t_a$-additive-error-
correcting BCH code having the roots 
\[ \alpha, \alpha^3, \alpha^5, \ldots, \alpha^{2^{t_a}-1}, \]
where \( \alpha \) is a primitive element of \( GF(2^6) \). The length of the code is 
\[ n = 2^6 - 1 = 63. \] Let us assume that the \( GF(2^6) \) is based on the primitive polynomial 
\[ f(x) = x^6 + x + 1. \]
Using Table 5.1, with \( m = 6 \), we have the minimum functions of powers of the primitive element \( \alpha \), where \( \alpha \) is a root of \( f(x) \). The polynomial \( x^{63} - 1 \) can be factored into \( x - 1 \) times the product of all the minimum functions listed in the table.

We shall now consider some specific illustrations of the application of the synchronization technique to the BCH codes of length 63.
We wish to choose for \( m(x) \) the polynomial \( m_1(x) \) of lowest degree such that \( e_1 \geq t_s + 1 \), where \( e_1 \) is the exponent to which \( m_1(x) \) belongs (i.e., \( e_1 \) is the order of the roots of \( m_1(x) \)). Suppose that we wish to correct \( t_a \) additive errors, where \( t_a \leq 4 \). Then the generator of \( C \) is 
\[ g(x) = m_1(x) m_3(x) \ldots m_{2^{t_a}-1}. \]

If we wish to correct \( t_s \) synchronization errors, where \( t_s \leq 2 \), we should choose 
\[ g_s(x) = g(x) m_{21}(x) \]
\[ = g(x) (x^2 + x + 1). \]
The order of the root \( \alpha^{21} \) of \( m_{21}(x) \) is \( e = 3 \), and the condition of Theorem 3.1 that \( t_s \leq e - 1 \) is thus satisfied.

For \( 2 < t_s \leq 6 \), we can choose 
\[ g_s(x) = g(x) m_9(x) \]
\[ = g(x) (x^3 + x^2 + 1). \]
or
\[ g_S(x) = g(x) \cdot m_{27}(x) \]
\[ = g(x) (x^3 + x + 1) , \]
since the order of \( \alpha^9 \) and \( \alpha^{27} \) is 7, and \( t_s \leq 7 - 1 = 6 \).

For \( 6 < t_s \leq 62 \), we can choose
\[ g_S(x) = g(x) \cdot m_1(x) \]
for any \( m_1(x) \) listed in the table that is not already a factor of \( g(x) \), and is such that the exponent \( e_i \) to which it belongs satisfies \( e_i \geq t_s + 1 \).

**Example 5.3. The Case \( n = 255 \).**

Consider now the BCH code of length \( n = 2^8 - 1 = 255 \) that has roots
\[ \alpha, \alpha^3, \alpha^5, \ldots, \alpha^{25} \]
where \( \alpha \), a primitive element of \( GF(2^8) \), is a root of the primitive polynomial
\[ f(x) = x^8 + x^4 + x^2 + 1 . \]
The minimum functions of powers of \( \alpha \) are shown in Table 5.1 with \( m = 8 \), and the product of these minimum functions, times \( x - 1 \), is equal to \( x^{255} - 1 \).

If we wish to correct \( t_s \) synchronization errors for \( t_s \leq 2 \), then
\[ m_{65}(x) = x^2 + x + 1 \]
can be taken as \( m(x) \). For \( 2 < t_s \leq 4 \), we can use
\[ m_{17}(x) = x^4 + x + 1, \]
\[ m_{51}(x) = x^4 + x^3 + x^2 + x + 1, \]
or
\[ m_{119}(x) = x^4 + x^3 + 1. \]
For \( 4 < t_s \leq 14 \), \( m_{17}(x) \) or \( m_{119}(x) \) can be used. For \( 14 < t_s \leq 254 \), we must use an eighth degree minimum function satisfying \( e_i \geq t_s + 1 \).

Of course, no minimum function \( m(x) \) can be used which is already a factor of \( g(x) \).
5.5. The Use of a Reducible Factor $m(x)$ for Some Ranges of $t_s$

In Theorem 3.1, it was assumed that the factor $m(x)$ was a minimum function, and therefore irreducible. This assumption can be relaxed, so that $m(x)$ can in fact be the product of any minimum functions that are not factors of $g(x)$. Of course, $m(x)$ would never be chosen with degree greater than $m$, since there are primitive polynomials of degree $m$, and $t_s$ in the case of a primitive polynomial can be as large as $n - 1$. Corresponding to each minimum function of degree less than $m$, there is a maximum possible value of $t_s$, namely $t_s^* = e - 1$, where $e$ is the exponent to which the minimum function belongs (i.e., $e$ is the order of $z^S$). If we write down the set of values $t_s^*$ corresponding to all possible minimum functions, we obtain a set

$$t_{s1}^*, t_{s2}^*, \ldots, t_{sv}^*.$$ 

Suppose that we wish to use a value of $t_s$ satisfying

$$t_{s1}^* < t_s \leq t_{s(i+1)}^*$$

for some $i$. It may be possible to use for $m(x)$ a product polynomial of degree less than the minimum degree possible for a polynomial of exponent $t_{s(i+1)}^* + 1$. This is illustrated in the following example.

Consider a BCH code of length $n = 2^6 - 1 = 63$. As shown in Table 5.1, if we choose for $m(x)$ a minimum function, we have for possible values of $t_s^*$ the values

$$2, 6, 8, 20, 62,$$

corresponding to minimum functions having exponents

$$3, 7, 9, 21, 63,$$

and (least) degrees

$$2, 3, 6, 6, 6.$$
Suppose that we wish to have $6 < t_s < 20$. If we restrict ourselves to using a minimum function for $m(x)$, then the degree of $m(x)$ must be six, since that is the least degree for which the exponent is greater than seven. We shall now show how we can choose for $m(x)$ a polynomial of degree five which is a product of two polynomials.

Consider the function

$$m(x) = m_9(x) m_{21}(x)$$

$$= (x^3 + x^2 + 1)(x^2 + x + 1)$$

$$= x^5 + x + 1,$$

and let us denote a root of $m_9(x)$ by $u$ and a root of $m_{21}(x)$ by $v$. We assume that $m(x)$ is not a factor of $g(x)$. Let us define the quantity

$$z = (\frac{u}{v}),$$

where we define

$$z^i = (\frac{u^i}{v^i})$$

and

$$z_1 + z_2 = (\frac{u_1 + u_2}{v_1 + v_2}),$$

if

$$z_1 = (\frac{u_1}{v_1}) \text{ and } z_2 = (\frac{u_2}{v_2}).$$

As usual, the order of $z$ is defined to be the least power, $e$, such that $z^e = (\frac{1}{1})$. Now $z^i$ is equal to

$$(\frac{1}{1})^{\frac{i}{1}}$$

only when both $u^i = 1$ and $v^i = 1$, and this will not happen for any $i$ less than the value $e = LCM(e_u, e_v)$, where $e_u$ and $e_v$ denote the orders of $u$ and $v$, respectively. Hence if we define

$$H_2^* = \begin{bmatrix} 1 & u & u^2 & \cdots & u^{n-1} \\ 1 & v & v^2 & \cdots & v^{n-1} \end{bmatrix},$$
and define $H'_2$ as the matrix obtained from $H'_2^*$ by deleting dependent rows, then the vectors

$$c(t_r + 1) H'_2 = z^{u-t_r-1} c H'_2$$

are distinct for $i = 0, 1, 2, \ldots, t_r, n-t_r, \ldots, n-1$ and $t_r + t_j = t_s \leq e - 1$. In the preceding example, we have $e = \text{LCM}(e_u, e_v) = \text{LCM}(7, 3) = 21$. Thus $t_s$ can be as large as 20, and the degree of $m(x) = x^5 + x + 1$ is 5, which is < 6, as was to be shown. (Note that for the example, $H'_2^*$ has twelve rows, and $H'_2$ has five rows.)

Note that the redundancy for synchronization purposes is $t_s + 5$ ($t_s \leq 20$), which is the minimum possible redundancy using the new synchronization technique (see Table 4.1).

5.6 Use of the Synchronization-Error-Correcting Capability of the Code to Enable Additional Additive-Error Correction

5.6.1. Introduction

The code $C_S$ is a $t_a$-additive-error-correcting BCH code of redundancy $r_S > r$, and if used as an additive-error-correcting code it may be capable of correcting more than $t_a$ additive errors. For example, $\alpha^S = \alpha^{2t_a + 1}$, the code $C_S$ is a BCH code capable of correcting at least $t_a + 1$ errors. ($C_S$ in this case may be in fact able to correct $t_a + 1 + a$ additive errors, if $\alpha^{2t_a + 3}, \alpha^{2t_a + 5}, \ldots, \alpha^{2(t_a + a) + 1}$ are also roots of $g_S(x)$.) Ordinarily, the additional redundancy of $C_S$ is used to enable synchronization-error correction. This section will discuss situations under which this additional redundancy can usefully be employed for increased additive-error correction instead of synchronization-error correction.
Because of the seriousness of synchronization errors, the receiver should not take corrective action upon observing the first indication (from the synchronization-error syndrome) that a synchronization error has occurred. For proper use of the technique, the truncated received word must not itself contain the synchronization error. Thus the first word after the word containing the synchronization error gives the first reliable indication of the occurrence of the synchronization error. Words actually containing symbol gains or losses are severely altered and may result in nonzero, but false, synchronization error syndromes. Also, if more than \( t_a \) additive errors occur, a nonzero synchronization-error syndrome may result even though there has been no synchronization error. The importance of making correct decisions regarding synchronization errors warrants the observation by the receiver of the same synchronization-error syndrome for several successive words before correcting the apparent synchronization error. Of course, if the receiver destroys synchronization by taking corrective action corresponding to a spurious nonzero synchronization-error syndrome resulting from the occurrence of more than \( t_a \) additive errors, then this mistake will be rectified with the next received word containing not more than \( t_a \) additive errors.

It is noted that of the possible different synchronization-error syndromes,

\[
\mathcal{Y}_c H_2
\]

which could result from more than \( t_a \) additive errors even if no synchronization error occurred, only \( t_s + 1 \) of them correspond to the proper synchronization-error syndromes.

\[
\left[ c(t_r + 1) - c(t_r) \right] H_2
\]
(since there are only $t_a + 1$ of these proper synchronization-error syndromes). Suppose that $z = \alpha^2$ is of order $e$. Then the matrix $H^*_2$ consists of $n/e$ cycles $1, z, z^2, \ldots, z^{e-1}$, as discussed in Section 3.4, and there are $q^e - 1$ possible nonzero values for $e H^*_2$, for varying $e$. There are thus $q^e - 1 - t_a$ nonzero syndromes $\gamma_c H^*_2$ that are different from the proper synchronization-error syndromes. If we assume that all possible nonzero synchronization-error syndromes occur with equal frequency if more than $t_a$ additive errors occur (corresponding to the case of randomly-occurring additive errors), then the probability is only $t_a/(q^e - 1)$ that one of these syndromes would correspond to a proper synchronization-error syndrome. If an improper syndrome (such as the vector $(01)$ in Example 5.1) occurred, it would indicate that more than $t_a$ additive errors had occurred (ignoring for the moment the possibilities that (1) the received word contains bit losses or gains (either a net bit loss or gain, or a compensating synchronization error), or (2) that an uncorrectable synchronization error had occurred). If $z = \alpha^{2t_a + 1}$, then $C_S$ is a BCH code capable of correcting, say, $t_a + 1 + a$ additive errors, and the receiver would use the code $C_S$ as such in an attempt to correct the additive errors. If not more than $t_a + 1 + a$ additive errors had in fact occurred, then the additive errors would be corrected.

In summary, if the receiver observes a nonzero synchronization-error syndrome for a particular word, but on no successive words, then it is possible that either a compensating synchronization error or an additive error of weight greater than $t_a$ has occurred in that word. The remaining possibility is that a synchronization error of uncorrect-
able order has occurred. This last possibility will be discussed in Section 6.3.

It is reiterated that, as pointed out in the first paragraph of this section, the easiest method by which to use the additional redundancy of the code $C_S$ for additive-error correction is to choose $z = \alpha^{2t_a + l}$. The code $C_S$ is then a BCH code capable of correcting $t_a + l + 1$ additive errors. We can summarize the procedure as follows.

If we take $z = \alpha^g = \alpha^{2t_a + l}$ in Theorem 3.1, then the code $C_T$ can be used to correct either

1. up to $t_a + l + 1$ additive errors if no synchronization error occurs; or
2. up to $t_a$ additive errors and synchronization errors of orders $t_l$ and $t_r$, where $t_l + t_r = t_S < e - 1$, and $e$ is the order of $\alpha^{2t_a + l}$.

We note that it is not possible to accomplish both 1 and 2 at the same time for a given received word. The receiver must decide which of 1 and 2 he wishes to do. This situation is analogous to using a minimum distance $d$ code for either additive-error correction alone or for some combination of additive-error correction and detection.

5.6.2. Examples

The use of the code $C_T$ in accordance with the procedure outlined above will now be illustrated by examples, which are continuations of Examples 5.2 and 5.1.

Example 5.4. Continuation of Example 5.2.

In order to use the $n = 63$ code of Example 5.2 as a BCH code for
correcting \( t_s + 1 \) additive errors, it is necessary that the polynomial \( m_i(x) \) in the expression

\[
g_S(x) = g(x) \cdot m_i(x)
\]

be the minimum function listed in the table corresponding to the least value of \( i \) for which \( m_i(x) \) is not a factor of \( g(x) \). Thus, for example, if

\[
g(x) = m_1(x) \cdot m_3(x)
\]

we would take

\[
g_S(x) = g(x) \cdot m_5(x)
\]

The code with generator \( g(x) \) is a 2-additive-error-correcting code.

The code with generator \( g_S(x) \) is a \((t_a, t_s)\)-error-correcting code, with \( t_a = 2 \), and \( t_s \) any value \( \leq 62 \) (since \( m_5(x) \) is primitive). It is capable of correcting either (1) 2 additive errors and up to \( t_s \) synchronization errors, or (2) 3 additive errors with no synchronization errors. Since \( m_5(x) \) is of degree 6, six information symbols are sacrificed for the synchronization-error correcting capability. If \( t_s \) is less than or equal to 20, then we are not fully utilizing all six redundant symbols for synchronization-error correction, since an adequate \( m(x) \) would have been \( m_{21}(x), m_{27}(x), \) or \( m_{21}(x)m_{27}(x) \), yielding 2, 3, or 5 redundant symbols, respectively, for \( t_s \leq 2, 2 < t_s \leq 6, \) or \( 6 < t_s \leq 20 \). For \( t_s > 20 \), a primitive polynomial \( m_i(x) \) of degree 6 must be used, and hence all 6 redundant symbols corresponding to the primitive polynomial are used for synchronization, in the sense that it is not possible to use an \( m(x) \) of lower degree and have \( t_s > 20 \).

**Example 5.5. Continuation of Example 5.1.**

We shall now provide an illustration of an actual situation of
the sort in which the additional additive-error correction could be accomplished. The code of Example 5.1 will be used.

Suppose that four successive information vectors are

\[ s_1 = (10100), \]
\[ s_2 = (00110), \]
\[ s_3 = (00010), \]
\[ s_4 = (11101). \]

These are encoded as

\[ x_1 = (110101111100100), \]
\[ x_2 = (001001101011110), \]
\[ x_3 = (000111011001010), \]
\[ x_4 = (101011110001001), \]

and then augmented and translated to form

\[ x_{1T} = (01011100000010001), \]
\[ x_{2T} = (10101101001111010), \]
\[ x_{3T} = (10010110000101010), \]
\[ x_{4T} = (00100100100100100). \]

Then the sequence

\[ \ldots 0101110000001000110111100101100100101100001000100100100100 \ldots \]

is sent over the channel. Now suppose that no synchronization error occurs, but that a triple additive error occurs in \( x_{2T} \), so that the sixth, tenth, and thirteenth positions are received in error. It is recalled that the code \( C_T \) is capable of correcting only two additive errors together with synchronization errors of order one in either direction. We shall now see how this triple error causes a spurious
nonzero synchronization-error syndrome (which is in fact one of the proper synchronization-error syndromes).

The sequence

\[ \ldots 0101110000010001101010010111010101001001100 \]

\[ 00101010001001000100100100 \ldots \]

is received, and after subtracting \( c_A \) and dropping the first and last symbols of each seventeen-symbol received word, the receiver has

\[ y_1 = (1010111110001001), \]
\[ y_2 = (0100010111110100), \]
\[ y_3 = (0011101100010100), \]
\[ y_4 = (0101111000100111). \]

Note that the additive errors are in the fifth, ninth, and twelfth places of \( y_2 \). The additive-error syndrome \( y_4 H' \) is equal to \( 0 = (00) \) for \( i = 1, 3, \) and \( 4 \), but for \( i = 2 \) we have

\[ y_2 H' = (010001011110100) H' \]
\[ = (00011101). \]

We observe that

\[ (0000000000010100) H' = (00011101), \]

so that when the receiver uses \( C \) as a double-error-correcting code, he will interpret that

\[ e = (0000000000010100) \]

is the error vector. Of course, this is not the correct error vector, which is (000010001001000). Calculating \( y_{2c} \), we obtain

\[ y_{2c} = y_2 - e \]
\[ = (010001011110100) - (0000000000010100) \]
\[ = (010001011100000). \]
The synchronization error syndrome $y_{1c}^i H_2^t$ is 0 for $i = 1, 3, \text{ and } 4$, but for $i = 2$ it is

$$y_{2c}^i H_2^t = (010001011100000) H_2^t = (10).$$

Since this is the synchronization-error syndrome corresponding to $i = 14 = 15 - 1$, the above syndrome indicates that a right-shift error of order 1 has occurred.

Thus the occurrence of $t_a + 1 = 3$ additive errors has resulted in the false indication that a right-shift synchronization error of order one has occurred. If the receiver takes corrective action, synchronization is destroyed, but it will be regained at the next word, $y_3^1(1)$, which would in that case indicate that a left-shift error of order 1 had occurred. Alternatively, the receiver could defer action until he observed that $y_3^i$ and $y_4^i$ did not indicate synchronization errors. He would then decide to use $C_S$ as a 3-additive-error-correcting code, on the assumption that the lone nonzero synchronization-error syndrome is the result of 3 additive errors. Calculating the syndrome

$$y_2^i H_S^t = (010001011110100) H_S^t$$

$$= (0001101101)$$

and employing the correction procedure for the BCH code $C_S$, he obtains

$$e^c = (000010001001000),$$

since

$$(000010001001000) H_S^t = (0001101101).$$

Calculating

$$y_{2c} = y_2 - e^c$$

$$= (01001101011110),$$
the receiver reverses the augmentation-truncation process and interprets the word as

\[ x_{2c}(n - t_r) = x_{2c}(-1) \]
\[ = (001001101011110), \]

which is precisely \( x_2 \). The receiver now resumes reception.

Note that the synchronization-error syndrome \( x_{2c} H_2^t \) is now zero. This must be the case, of course, since the equations

\[ x_2 H_S^t = e H_S \]

resulted in a solution \( e^c \) such that

\[ x_{2c} H_S^t = (x_2 - e^c) H_S = 0. \]

Hence

\[ x_{2c} H_2^t = 0. \]

The use of \( C_S \) as a 3-additive-error-correcting code assumed that no synchronization errors had occurred, and so the syndrome \( x_{2c} H_2^t \) has no significance in this case with regard to synchronization.

It is noted that in the above example, the triple additive error was such that a proper nonzero synchronization-error syndrome was observed (i.e., \( [q(t_r, t_i) - q(i)] H_2^t \) for some permissible value of \( i \), in this case, the syndrome \( (10) \)). As indicated previously, excess additive errors can be such that the spurious synchronization-error syndrome is nonzero but not equal to a proper nonzero synchronization-error syndrome.

5.6.3. Use of the Augmented Code to Detect Additive Errors

Section 5.6.1 indicated how to make use of the additional redundancy that is in \( C_S \) but not in \( C \) to correct additional additive errors. Recall that the code \( C_A \) has \( t_s \) more redundant symbols than the
code $C_S$, since each codeword of $C_A$ is formed by repeating the first $t_s$ symbols of a word in $C_S$.

These repeated symbols can be used to detect the occurrence of additive errors in the first $t_s$ or last $t_s$ coordinates of the received word $\mathbf{y}$, simply by comparing the first $t_s$ and last $t_s$ coordinates of $\mathbf{y}$ and noting the coordinates that disagree. If the members of a pair of corresponding coordinates disagree (and no synchronization error has occurred), then at least one of them is in error. If the additive-error-correction procedure of the code $C_S$ is applied to the first $n$ and last $n$ coordinates of a received word $\mathbf{y}$, then the additive errors found should be consistent with the error indications as noted above. Such an error detection procedure is a weak one, and not much additional information is obtained by employing it.

Finally, it is noted that by just comparing the first $t_s$ and last $t_s$ symbols of the received word $\mathbf{y}$, the receiver could perhaps maintain synchronization with fair reliability, particularly if $t_s$ were large. The new technique has the advantage that it can resynchronize immediately, with $t_s$ as small as desired, and in the presence of additive errors.
CHAPTER VI

COMPARISON OF THE NEW SYNCHRONIZATION TECHNIQUE
WITH PREVIOUS TECHNIQUES

6.1. Introduction

Chapter IV provided a comparison of the new synchronization technique with an asymptotic form of an extension of the Hamming bound. This comparison indicated that in a limiting sense, the new technique appears to be efficient for $t_s$ small compared with $N$. It is desirable to compare the new technique with other techniques, for parameter ranges of practical interest; such a comparison will provide a useful indication of the efficiency of the new technique relative to actual alternative methods, rather than to an asymptotic bound. The new technique will be compared with the fixed synchronization sequence procedure and Levy’s "altered codes" method. It is noted that the various techniques for synchronization are fundamentally different, and that they possess a variety of different properties with respect to which comparisons could be made. For this reason, there is no "natural" basis on which to compare the new technique with previous techniques. The bases selected for comparison in this chapter represent particular choices of a variety of criteria, and are not represented to constitute an "absolute" comparison in any sense. It is hoped, however, that the comparisons presented constitute a useful indication of the relative capabilities of the methods considered over a practical range of parameter values.
6.2. \textbf{Comparison With Previous Techniques}

6.2.1. \textbf{Consideration of the New Technique as a Generalization of the Synchronization Sequence Approach}

Before comparing the new technique with the fixed synchronization sequence procedure, we will first show that the new technique can be considered to be a generalization of the synchronization sequence approach. We shall in fact show how the new synchronization technique can be considered as a method of adding to each word of $C_S$ a synchronization sequence which depends on the codeword. To see that this is true, let us suppose that the canonical form of the parity check equations specified by $H_S$ are used for encoding. If $x$ is in the sub-code $C_S$, then $x H_S^t = 0$. Now $H_S$ is of rank $n - k_S$, so that these equations can be solved for $x_{k_S+1}, x_{k_S+2}, \ldots, x_n$ in terms of $x_1, x_2, \ldots, x_{k_S}$. Thus, if $x_1, x_2, \ldots, x_{k_S}$ are information symbols, then $x_{k_S+1}, x_{k_S+2}, \ldots, x_n$ are the corresponding parity checks. Thus, using the new synchronization technique, a sequence $x_1, x_2, \ldots, x_{k_S}$ of information symbols is encoded into

$$x_A = (x_1, x_2, \ldots, x_{k_S}, x_{k_S+1}, \ldots, x_n, x_{n+1}, \ldots, x_N)$$

where $x_{k_S+1}, x_{k_S+2}, \ldots, x_N$ are parity checks, $N = n + t_s$, and $k_S = k - \deg[m(x)]$. Now $r = n - k$ of the parity checks are used for additive-error correction, so that

$$r_s - r = (N - k_S) - (n - k)$$

$$= (n + t_s) - (k - \deg[m(x)]) - n + k$$

$$= t_s + \deg[m(x)]$$

parity check symbols are used in $x_A$ for synchronization purposes. Further, let us suppose that $H_S$ is arranged (as usual)
\[ H_S = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}, \]

so that the parity checks \( x_{n - \deg[m(x)] + 1}, \ldots, x_n \) are associated with synchronization-error correction. Thus, in all, the last \( t_s + \deg[m(x)] \) symbols of \( x_A \), namely

\[ x_{n - \deg[m(x)] + 1}, \ldots, x_n, x_{n + 1}, \ldots, x_{n + t_s}, \quad n + t_s = N, \]

are for synchronization-error correction.

We can summarize the situation as follows. Let

\[ x_1, x_2, \ldots, x_{k_S} \]

be a sequence of information symbols, and let

\[ x_A = (x_1, x_2, \ldots, x_{k_S}, x_{k_S + 1}, \ldots, x_N) \]

be the corresponding codeword of \( C_A \). Let \( P = n - \deg[m(x)] \) and \( L = t_s + \deg[m(x)] \) so that \( N = P + L \). Then

\[ x_1, x_2, \ldots, x_{k_S}, x_{k_S + 1}, \ldots, x_{n - \deg[m(x)]} \]

or (since \( P = n - \deg[m(x)] \))

\[ x_1, x_2, \ldots, x_P \]

is the part of the codeword \( x_A \) comprised of the information symbols and the redundant symbols associated with additive-error correction, and

\[ x_{n - \deg[m(x)] + 1}, \ldots, x_{n + t_s} \]

or (since \( L = t_s + \deg[m(x)] \))

\[ x_{P + 1}, x_{P + 2}, \ldots, x_{P + L} \]

are the redundant symbols associated with synchronization-error correction. If we call the sequence \( x_1, x_2, \ldots, x_P \) the base word, then to each base word (of length \( P \)) there is added a tailormade synchro-
nization sequence of length L, where \( N = P + L \).

The essential difference between previous synchronization sequence techniques (such as the use of Barker sequences) and the new technique is that the previous techniques employ the same sequence every time whereas the new technique employs a different sequence for each different base word. In either case, let us denote the length of the synchronization sequence by \( L \), and the length of the base word (i.e., the sequence of symbols that are not associated with synchronization-error correction) by \( P \).

Having shown the new technique to be a generalization of the synchronization sequence procedure, we shall now compare the synchronization protection afforded by the two methods.

6.2.2. Comparison With the Interlaced Synchronization Symbols Method

6.2.2.1. Introduction

For proper comparison of two synchronization techniques, the number and type of additive and synchronization errors which each method can correct should be identified. Unfortunately, this cannot always be determined. For example, with the interlaced synchronization symbols method (in which \( L \) arbitrary synchronization symbols occur at intermittent locations in the codeword) synchronization can be established at positions for which received subsequence matches the synchronization sequence in at least \( L - K \) symbols (i.e., up to \( K \) additive errors are allowed in the sequence). Synchronization errors of every order are corrected with a probability that is an increasing function of the number, \( L \), of interlaced symbols. If additive errors can occur, they have the effect of reducing this probability. We shall
now present a comparison of the interlaced synchronization symbols technique with the new technique, assuming that additive errors do not occur in the interlaced pattern of symbols. This condition will result in a conservative (i.e., pessimistic) comparison of the capability of the new technique with respect to the interlaced synchronization symbols technique.

6.2.2.2. The Probability of Error for the New Technique

Using the new synchronization technique, the probability $P^{(1)}$ that a correctable synchronization error is not corrected is upper-bounded by the probability that more than $t_a$ additive errors occur in the truncated word, $y$. This probability is

$$P = 1 - \sum_{i=0}^{t_a} \binom{n}{i} p^i (1-p)^{n-i}$$

where $p$ is the probability\(^1\) that a symbol is received in (additive) error. Approximately (for $n$ large) this is

$$1 - \sum_{i=0}^{t_a} \frac{e^{-np}(np)^i}{i!}.$$  

This approximate upper bound for $P^{(1)}$ is shown in Table 6.1 for a range of values of $np$. The approximation is adequate for the purposes of this chapter. For exact values of $P$, tables of the binomial probability distribution (e.g., [26]) should be consulted.

\(^1\)In this chapter, the symbol $p$ is used, as is customary, for the probability of additive error in a symbol. No confusion should arise between this use of the letter $p$ and the previous use of $p$ for the characteristic of the Galois field $GF(q=p^m)$. 
TABLE 6.1.

APPROXIMATE UPPER BOUND FOR THE PROBABILITY, $p^{(1)}$, OF FALSE SYNCHRONIZATION USING THE NEW SYNCHRONIZATION TECHNIQUE (APPROXIMATE PROBABILITY OF UNCORRECTABLE ADDITIVE ERRORS)

<table>
<thead>
<tr>
<th>$\eta \rho$</th>
<th>.05</th>
<th>.1</th>
<th>.25</th>
<th>.5</th>
<th>1</th>
<th>2.5</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.04877</td>
<td>.09516</td>
<td>.22120</td>
<td>.39347</td>
<td>.63212</td>
<td>.90791</td>
<td>.99326</td>
<td>.99995</td>
</tr>
<tr>
<td>1</td>
<td>.00121</td>
<td>.00468</td>
<td>.02650</td>
<td>.09020</td>
<td>.26424</td>
<td>.71270</td>
<td>.95957</td>
<td>.99950</td>
</tr>
<tr>
<td>2</td>
<td>.00002</td>
<td>.00015</td>
<td>.00216</td>
<td>.01439</td>
<td>.08030</td>
<td>.45619</td>
<td>.87535</td>
<td>.99723</td>
</tr>
<tr>
<td>3</td>
<td>.00013</td>
<td>.00175</td>
<td>.01899</td>
<td>.24242</td>
<td>.73497</td>
<td>.98966</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.00001</td>
<td>.00017</td>
<td>.00366</td>
<td>.10882</td>
<td>.55951</td>
<td>.97075</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.0001</td>
<td>.00059</td>
<td>.04202</td>
<td>.38404</td>
<td>.93291</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.0008</td>
<td>.01419</td>
<td>.23782</td>
<td>.86986</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.00001</td>
<td>.00425</td>
<td>.13337</td>
<td>.77978</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.00114</td>
<td>.06809</td>
<td>.66718</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>.00028</td>
<td>.05183</td>
<td>.54207</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.00006</td>
<td>.01370</td>
<td>.41696</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.00000</td>
<td>.00545</td>
<td>.30322</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>.00002</td>
<td>.20844</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>.00070</td>
<td>.13554</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>.00023</td>
<td>.08346</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>.00007</td>
<td>.04874</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>.00002</td>
<td>.02704</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>.00071</td>
<td>.00159</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>.00003</td>
<td>.0005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>.00000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>.00001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>.00000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 Table entry is $\text{Prob}[t > t_a]$, where $t$ is a random variable having a Poisson distribution with parameter $\lambda = \eta \rho$. (This is approximately equal to $\text{Prob}[\text{more than } t \text{ additive errors }]$.) Where no table entry occurs, probability is $< 5 \times 10^{-5}$. 
6.2.2.3. The Probability of Error for the Interlaced Synchronization Symbols Method

Barker has presented an analysis of the efficiency of arbitrary synchronization sequences which includes the interlaced synchronization symbols method as a special case. The results that he derived will briefly be described here.

The receiver decides that the synchronization sequence has been sent if a sequence is received that matches the synchronization sequence in at least $L - K$ places. (Barker uses $n$ and $k$ for $L$ and $K$, respectively.) To describe the capability of a synchronization sequence (or synchronizing pattern, in Barker's terminology), Barker considers the probabilities $P_i$ ($i = 1,2,3,4$) of four (exhaustive and mutually exclusive) events:

- $E_1$: the synchronization sequence is sent and observed (i.e., recognized);
- $E_2$: the synchronization sequence is sent but not observed;
- $E_3$: the synchronization sequence is not sent but is observed by chance; and
- $E_4$: the synchronization sequence is not sent and is not observed.

Barker provides curves showing (1) the approximate probability $F(K)$ (as a function of $K$, $L$, and the prior probability $S = P_1 + P_2$ that the synchronization sequence was sent) that the synchronization sequence was not sent, given that it was observed; and (2) the approximate probability $M(K)$ (as a function of $K$ and $\lambda = \lambda_p$) that the synchronization sequence was not observed, given that it was sent.

We now calculate the probability $P^{(2)}$ of incorrect synchroniza-
tion using an arbitrary but fixed pattern of synchronization symbols placed throughout a word. In order to do so, it is necessary to specify the prior probability, $S$, that the synchronization sequence occurs at a given position. Since we are considering only $t_s + 1$ positions (corresponding to the errors correctable by the new technique), we can take $S = 1/(t_s+1)$. We can then lower-bound $P^{(2)}$ by the quantity

$$F(0) = 1/2^L s,$$

which Barker shows to be the approximate probability that the synchronization sequence was not sent, given that it appears in a random sequence of bits. That is, $F(0)$ is the probability of false synchronization at any position, using $K = 0$. $F(0)$ is a lower bound for $P^{(2)}$, since the receiver may synchronize incorrectly because of either (1) false synchronization action due to observation of the synchronization sequence at an incorrect position, or (2) failure to take necessary corrective synchronization action due to failure to observe the synchronization sequence anywhere. Values of $L$ and corresponding values of the lower bound $F(0)$ for $P^{(2)}$ are shown in Table 6.2, for the case $n = 255$ (assuming binary digits).

6.2.2.4. Comparison of the Two Techniques

The situation is illustrated by the case considered in Table 6.2. Whenever, for a given $L$, the probability $P$ of more than $t_a$ additive errors in a word is less than the tabled value (lower bound for $P^{(2)}$) corresponding to $L$, the new technique is more efficient, with respect to correctable synchronization errors, than the arbitrary-fixed-synchronization-symbols technique, since $P$ is an upper bound on the
<table>
<thead>
<tr>
<th>$t_s^3$</th>
<th>$L^4$</th>
<th>Lower bound for probability $P(2)$ of false synchronization $= (t_s + 1)/2^L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>.25</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>.19</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$3 \times 10^{-2}$</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>$2 \times 10^{-2}$</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>.10</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>$7 \times 10^{-3}$</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>$4 \times 10^{-3}$</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>$2 \times 10^{-3}$</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>.10</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>$7 \times 10^{-4}$</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>$4 \times 10^{-4}$</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>$2 \times 10^{-4}$</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>.10</td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td>$6 \times 10^{-5}$</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>$2 \times 10^{-5}$</td>
</tr>
<tr>
<td>16</td>
<td>21</td>
<td>$8 \times 10^{-6}$</td>
</tr>
<tr>
<td>17</td>
<td>22</td>
<td>$4 \times 10^{-6}$</td>
</tr>
<tr>
<td>18</td>
<td>23</td>
<td>$2 \times 10^{-6}$</td>
</tr>
<tr>
<td>19</td>
<td>24</td>
<td>.10</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>$5 \times 10^{-7}$</td>
</tr>
<tr>
<td>21</td>
<td>29</td>
<td>$4 \times 10^{-8}$</td>
</tr>
<tr>
<td>22</td>
<td>30</td>
<td>$2 \times 10^{-8}$</td>
</tr>
</tbody>
</table>

$t_s^3$ is the maximum number of correctable synchronization errors if the new technique were used with an $n = 255$ binary BCH code with the given value of $L$ (see Table 4.1).

$L^4$ is the number of synchronization symbols per word.
probability $P^{(1)}$ of synchronization failure with the new technique while the value in Table 6.2 is a lower bound on the probability $P^{(2)}$ of synchronization failure with the interlaced technique. For example, if $P < 10^{-6}$, then for all values of $L$ not exceeding $2^4$ the new technique is more efficient, since $P^{(2)} \geq 10^{-6}$ for $L \leq 2^4$. The comparison is very conservative since it is based just on an upper bound for $P^{(1)}$ and a lower bound for $P^{(2)}$. If actual values for $P^{(1)}$ and $P^{(2)}$ were calculated in specific instances, given values for $p$ and values for $t_a$ which are comparable to the value of $K$, the breakpoint would be greater than $2^4$.

The fixed synchronization pattern of $L$ symbols located at arbitrary (fixed) positions in the codewords has the advantage that, if a synchronization error of very large order occurs, then even for a moderate value of $L$ the receiver is not likely to observe the pattern of synchronization symbols other than where it occurs, and he will therefore be able to reestablish synchronization. Furthermore, the "pattern recognition" feature of the fixed technique allows for correction of synchronization errors of any order. With the technique introduced in this dissertation, however, the synchronization sequence depends on each particular codeword, and if there occurs a synchronization error that is not correctable, the receiver must employ a somewhat involved procedure to reestablish synchronization (as discussed in Section 6.3). The new technique is thus best suited for situations in which the synchronization errors are correctable. It is noted, of course, that the new technique was designed for correction of synchronization errors of specified orders.
6.2.3. Barker Sequences

6.2.3.1. Introduction

If a synchronization sequence technique is such that synchronization symbols in a word occur together rather than intermittently, then the probability of observing the synchronization sequence at an incorrect position within \( L - 1 \) positions of the correct position (i.e., the probability that an overlap of the synchronization sequence is identical with the synchronization sequence) depends not only on the probability, \( p \), of additive errors, but on the nature of the sequence itself. Because of this, if synchronization is known to within \( L - 1 \) symbols of the correct position, the receiver can have a better chance of correct synchronization if he properly chooses the synchronization sequence than if he uses an arbitrary pattern of synchronization symbols. As discussed in Chapter I, Barker [1] determined sequences having the property that the correlation of a sequence and an overlap comprised of part of the sequence and part of the null sequence (consisting of \( L \) 0's) was close to zero. By using Barker sequences, it is quite unlikely that the receiver will make a synchronization error within \( L - 1 \) positions of the correct position, even if a number of additive errors occur.

6.2.3.2. Some Properties of Barker Sequences

Before comparison of the Barker sequence synchronization technique with the new technique, it is convenient to first derive certain properties of Barker sequences. Let us denote a Barker sequence composed of \( L \) 0's and 1's by

\[
\mathbf{b}_L = (b_1, b_2, \ldots, b_L).
\]
Let us change the 0's to -1's by applying the transformation
\[ b_i^* = 2b_i - 1. \]
Then, by the definition of a Barker sequence, the sequence
\[ b_L^* = (b_1^*, b_2^*, \ldots, b_L^*) \]
must satisfy
\[ \sum_{j=1}^{L-1} |b_j^* b_{j+1}^*| \leq 1 \]
for \( i = 1, 2, \ldots, L-1 \).

Let us define \( b_i \) as the vector formed by changing the last \( i \) coordinates of \( \underline{b}(i) \) to zero; i.e.,
\[ b_i = (b_{i+1}, b_{i+2}, \ldots, b_L, 0, 0, \ldots, 0). \]
Similarly, define
\[ b_i^* = (b_{i+1}^*, b_{i+2}^*, \ldots, b_L^*, 0, 0, \ldots, 0). \]
Now the number of mismatching symbols in the first \( L - i \) positions of \( b^* \) and \( b_i^* \) is given by
\[ m_2(i) = \frac{1}{2} \sum_{j=1}^{L-1} |b_j^* - b_{j+1}^*| \]
\[ = \sum_{j=1}^{L-1} |b_j - b_{j+1}|, \]
and the number of matches in the first \( L - i \) positions is thus
\[ m_1(i) = L - i - m_2(i). \] (Note that \( m_1(i) \) and \( m_2(i) \) are also the number of matches and mismatches, respectively, in \( \underline{b} \) and \( b_i \).) For (6.1) to be satisfied, the number of matches, \( m_1(i) \), and the number of mismatches, \( m_2(i) \), in the vectors \( b^* \) and \( b_i^* \) must satisfy
\[ |m_1(i) - m_2(i)| \leq 1. \]
Thus if \( L - i \) is even, we have
\[ m_1(i) = m_2(i) = \frac{L-i}{2}, \]
and if \( L-i \) is odd, we have either
\[ m_1(i) = \frac{L-i-1}{2} \quad \text{and} \quad m_2(i) = \frac{L-i+1}{2} \]
or
\[ m_1(i) = \frac{L-i+1}{2} \quad \text{and} \quad m_2(i) = \frac{L-i-1}{2}. \]

Let us now consider the vector
\[ \mathbf{b}^{(i)} = (b_1, b_2, \ldots, b_i, b_1, b_2, \ldots, b_{L-i}). \]
The vector \( \mathbf{b}^{(i)} \) differs from \( \mathbf{b} \) in \( m_2(i) \) of the last \( L-i \) positions, and of course is identical to \( \mathbf{b} \) in the first \( i \) positions. Let us denote by \( \mathbf{b}^{(i)}_{\perp} \) and \( \mathbf{b}_1 \), respectively, the vectors obtained by changing any
\[ \left[ \frac{m_2(i)}{2} \right] + 1 \]
of the symbols in the mismatched positions of \( \mathbf{b}^{(i)} \) and \( \mathbf{b} \) (where \( \left[ x \right] \) denotes the greatest integer \( \leq x \)). Then we have
\[ d(\mathbf{b}^{(i)}_{\perp}, \mathbf{b}) < d(\mathbf{b}_1, \mathbf{b}), \]
i.e., \( \mathbf{b}^{(i)}_{\perp} \) is nearer \( \mathbf{b} \) than is \( \mathbf{b}_1 \).

Finally, we have that
\[ m_2(i) \geq \left[ \frac{L-i}{2} \right] \]
and
\[ \left[ \frac{m_2(i)}{2} \right] \leq \left[ \frac{L-i+1}{4} \right] \]
since \( m_2(i) \) is equal to one of \( (L-i-1)/2 \), \( (L-i)/2 \), or \( (L-i+1)/2 \).

Having established the preceding results, we shall proceed with a comparison of the Barker sequence synchronization technique and the new technique.
6.2.3.3. The Probability of Error for a Suggested Method of Immediate Synchronization-Error Correction in the Presence of Additive Errors, Using Barker Sequences

The new synchronization technique is a procedure for correction of synchronization errors with the word immediately after the synchronization error. In order to consider the Barker-sequence synchronization technique and the new technique on a comparable basis, we shall consider a method by which the Barker sequence synchronization technique can be used to immediately correct synchronization errors. We shall adopt a synchronization rule analogous to the minimum-distance correction method for additive errors. That is, we shall synchronize at the position for which the L-sequence preceding that position differs least from the Barker sequence (i.e., is nearest to the Barker sequence, in the sense of Hamming distance).

Let us now consider conditions under which false synchronization would be caused by the receiver due to too many additive errors. Suppose that the sequence

\[ \ldots, x_1, x_2, \ldots, x_P, b_1, b_2, \ldots, b_L, \ldots \]

is transmitted, where \( b = (b_1, b_2, \ldots, b_L) \) is the Barker sequence. Consider the received sequence

\[ \ldots, x_1 + e_1, x_2 + e_2, \ldots, x_P + e_P, b_1 + e_{P+1}, b_2 + e_{P+2}, \ldots, b_L + e_N, \ldots \]

where \( N = P + L \). Now from the considerations of the preceding section, the received sequence

\[
(x_{P-1+1} + e_{P-1+1}, \ldots, x_P + e_P, b_1 + e_{P+1}, b_2 + e_{P+2}, \ldots, b_{L-1} + e_{N-1})
\]
will differ less from \( b \) than will the sequence
\[
(b_1 + e_{p+1}, b_2 + e_{p+2}, \ldots, b_L + e_N)
\]
if \( (x_{p+1} + e_{p+1}, \ldots, x_p + e_p) = (b_1, b_2, \ldots, b_i) \) and at least
\[
\left[ \frac{m_2(i)}{2} \right] + 1
\]
of the mismatched symbols of \( B^{(i)} \) are in error. But the probability that this happens is greater than the probability \( P_i \) that \( (x_{p+1} + e_{p+1}, \ldots, x_p + e_p) = (b_1, b_2, \ldots, b_i) \) and exactly
\[
\left[ \frac{m_2(i)}{2} \right] + 1
\]
of the mismatched symbols of \( B^{(i)} \) are in error. Assuming that \( x_i \)'s are random bits, and \( p \) is the probability of an additive error in a symbol, we have
\[
P_i = \left( \frac{1}{2} \right)^i \left[ \frac{m_2(i)}{2} \right] + 1 \left[ \frac{m_2(i)}{2} \right] - 1 \left( 1 - p \right)^{L - i - \left[ \frac{m_2(i)}{2} \right] - 1}
\]
where \( i \leq L - 1 \). The inequality (6.2) holds since, if \( b > a/2 \), then
\[
\binom{a}{b} > \binom{a}{b + 1}
\]
and hence
\[
\binom{a}{b} > \binom{a}{b + 1} > \binom{a - 1}{b + 1}
\]
If we assume that \( p \) is small, we may neglect the factor
\[
\frac{m_2(i)}{L - i - \left[ \frac{m_2(i)}{2} \right] - 1}
\]
occurring in (6.2), obtaining an approximate lower bound for \( P_p \), namely

\[
(6.2.1) \quad P_p \geq \left( \frac{1}{2} \right)^i \left( \frac{m_2(i)}{\left[ \frac{L-1}{2} \right]} + 1 \right) \left( \left[ \frac{L-1}{2} \right] + 1 \right)
\]

\[
\geq \left( \frac{1}{2} \right)^i \left( \frac{\left[ \frac{L-1}{2} \right]}{\min\left( \frac{L-1}{2}, \left[ \frac{L-1}{4} \right] + 1 \right)} \right) \]

\[= P(i), \text{ say.} \]

Thus, \( P(i) \) is an approximate lower bound for the probability that synchronization is incorrectly placed \( i \) positions from the correct position because of an excess of additive errors in the Barker sequence. (The advantage in using the right-hand side of (6.2.1) rather than the left-hand side lies with the fact that the right-hand side depends on the length, \( L \), of the sequence, rather than on \( m_2(i) \).)

Now let us consider the probability \( P^{(3)} \) that the suggested Barker technique falsely synchronizes at positions which are correctable with the new technique. An approximate lower bound to this probability is given by the sum of the probabilities \( P(i) \) for \( i = 1, 2, \ldots, t_1 \) and \( i = 1, 2, \ldots, t_r \). Thus \( P^{(3)} \) is approximately lower-bounded by

\[
(6.2.2) \quad \left( \sum_{i=1}^{t_1} P(i) \right) + \left( \sum_{i=1}^{t_r} P(i) \right).
\]

Values of this approximate lower bound for \( P^{(3)} \) are shown in Table 6.3 for various values of the length, \( L \), of the Barker sequence. The value of \( t_s \) shown in the table is that which corresponds to \( L \) if
### TABLE 6.3.

**AN APPROXIMATE LOWER BOUND FOR THE PROBABILITY OF FALSE SYNCHRONIZATION USING THE SUGGESTED BARKER-SEQUENCE METHOD FOR SYNCHRONIZATION**

<table>
<thead>
<tr>
<th>$t_{s_{\text{max}}}$</th>
<th>$t_{s_{255}}$</th>
<th>$L$</th>
<th>Lower bound for probability $P(3)$ of false synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>$.5P^2$</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
<td>$.4P^2 + .15P^2</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>11</td>
<td>$.825P^2 + 1.05P^3 + .10P^4</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>15</td>
<td>$.625P^6 + .16P^5 + .14P^4</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>23</td>
<td>$.69P^6 + .16P^5 + .04P^4</td>
</tr>
</tbody>
</table>

5. $t_{s_{\text{max}}}$ is the maximum number of synchronization errors that can be corrected using the new technique, given that there are $L$ redundant symbols for synchronization, for a binary BCH code such that $m(x)$ is of minimum possible degree. That is, $t_{s_{\text{max}}}$ is the maximum $t$ such that $t = L - u$, where $2^{u-1} - 2 < t < 2^u - 2$. (See Table 4.1.)

6. $t_{s_{255}}$ is the maximum number of synchronization errors that can be corrected using a BCH binary code of length $n = 255$, given that there are $L$ redundant symbols used for synchronization. (See Table 4.1.)

7. $L$ is the length of the Barker sequence.
the new method is used. In the table, we have considered the situations

\[ t_r = \left[ \frac{S}{2} \right], \quad t_s = \left[ \frac{S + 1}{2} \right]; \]

\[ t_r = 0, \quad t_s = S \quad \text{(or} \quad t_s = 0, \quad t_r = S). \]

For \( L = 3, 7, \) and \( 11 \), Barker sequences are 110, 111010, and 1110011010, and the more precise expression on the left-hand side of (6.2.1) is used to calculate the lower bound for \( P^{(3)} \). For values of \( L > 11 \), probabilities may be calculated even though Barker sequences of the given length may not exist; the result in this case would correspond to values for "near-Barker" sequences.

6.2.3.4. Comparison of the Two Techniques

Using Table 6.3, the new synchronization technique and the suggested Barker-sequence synchronization technique may be compared. For given \( L \), whenever the probability, \( P \), of more than \( t_s \) additive errors, is less than the approximate lower bound for \( P^{(3)} \) as calculated from Table 6.3, then the new synchronization technique is (approximately) more efficient (with regard to correctable synchronization errors), than the suggested Barker-sequence procedure. The reason for this is that \( p \) is an upper bound for the probability \( P^{(1)} \) of synchronization failure with the new technique, and the value calculated from Table 6.3 is a lower bound for the probability \( P^{(3)} \) of synchronization for the suggested Barker-sequence procedure.

To illustrate such a comparison, let us suppose that \( p = .01 \), and that we are using the new synchronization method with a \( t_a \)-additive-error-correcting code such that the probability of more than
the additive errors is $\leq 10^{-6}$. Then an upper bound to the probability of false synchronization by the new technique due to an excess of additive errors is $10^{-6}$.

Under these assumptions, we observe that, for the choices of $t_r$ and $t_s$ in Table 6.3, the use of the Barker sequences is (approximately) less efficient than the use of the new technique for $L \leq 15$, i.e., for $t_s \leq 11$. For $L > 15$, however, Barker sequences (i.e., near-Barker sequences) may enable more accurate synchronization. (The situation is not certain since it is an upper bound for the probability of synchronization failure using the new technique that is greater than a lower bound for the probability of synchronization failure using the suggested Barker-sequence procedure.) Note that if $t_r = t_s$ (corresponding to the situation where bit gains only are protected against), or if $t_r = t_s$ (corresponding to the situation where bit losses only are protected against), then for $L = 15$ the new technique is considerably more efficient than the suggested Barker technique.

One reason why the Barker sequence technique is very efficient for large values of $L$ is the following. With Barker sequences there is a very small number of error patterns that can cause false synchronization. Further, the number of additive errors required in any pattern to cause the Barker sequence to appear in the wrong position becomes quite large as the sequence length, $L$, increases. The probability of a large number of additive errors is, of course, extremely small if $p$ is reasonably small.

Observe that if we set $t_r = t_s$, then 1 in (6.2.2) can be as large as $[t_s/2]$, which approaches $[L/2]$ as $t_s$ becomes large. Thus the exponent $[(L-1+1)/4] + 1$ in (6.2.1) approaches $L/8$ as $t_s$ (and hence
L) becomes large. Hence, for L large, an approximate lower bound to the probability of false synchronization with a Barker sequence (at positions which would be correctable using the new method with the same value of L) is given by (6.2.1) with \( i = \frac{L}{2} \), which is approximately

\[
\left( \frac{1}{2} \right)^{\frac{L}{L/8}} \frac{L/8}{p} \approx \frac{2^{-t(L-6)}}{\sqrt{\pi L}} \frac{L/8}{p}
\]

(6.3)

(using Stirling's formula).

With the new method, however, large \( t_s \) would typically correspond to large code length, N. By the coding theorem of information theory (see [14]), the average probability of an uncorrectable additive error of a randomly chosen group code decreases exponentially with the code length, n; i.e.,

\[
\text{Prob[uncorrectable error]} = \text{Prob[more than } t_s \text{ additive errors]}
\]

\[
\leq A_2^{-nK(R)}
\]

where \( K(R) \) is a constant depending on the channel but not on n, and A is a constant, which is equal to 1 under certain conditions, and proportional to \( 1/\sqrt{n} \) otherwise. \( K(R) = 0 \) when the transmission rate, R, is equal to C, the channel capacity, \( K(R) > 0 \) for \( R = 0 \), and \( K(R) \) decreases as R increases from 0 to C.) Now \( n = N - L \), so that an upper bound to the average probability of error using the new technique is

\[
A_2^{-((N-L)K(R))}
\]

(6.4)

The expressions (6.3) and (6.4) provide an asymptotic comparison of the probability of false synchronization of the suggested Barker sequence method and the new method, with L the same for both methods. For L small compared to N, say \( L = 0.05N \), we have \( n = N - L \approx N \), so that
(6.4) is approximately

\[ A_2^{\cdot N K(R)} \]

and (6.3) is approximately

\[ \frac{7}{\sqrt{N}} 2^{-0.0125N} p^{0.006N} \]

For \( p = 0.01 \), reasonable values for the constant \( K(R) \) would lie between .01 and .2, and so (6.5) becomes

\[ A_2^{\cdot 0.01N} \text{ to } A_2^{\cdot 2N} \]

while (6.6) becomes

\[ \frac{7}{\sqrt{N}} 2^{-0.0125N} (.01)^{0.006N} \approx \frac{7}{\sqrt{N}} 2^{-0.05N} \]

The comparison between (6.7) and (6.8) indicates that there can be a marked superiority of the new method over the suggested Barker sequence method, provided \( N \) is very large, \( L \) is small compared with \( N \), and provided we can find "good" additive-error-correcting group codes (i.e.,

codes with transmission rate close to capacity, and small probability of uncorrectable errors). (It should be borne in mind that (6.7)

provides upper bounds, and that (6.8) is an approximate lower bound;

the actual situation would be more favorable for the new technique

than these bounds indicate. For a given value of \( N \) (even if \( L \) is small

relative to \( N \)), it is possible that the Barker sequence technique

could be better than the new technique with the best known additive-
error-correcting code. Furthermore, for \( N \) of moderate size (say \( N \approx 

255 \)), we may not wish to add very many redundant symbols for additive

error correction. In this case, the probability of uncorrectable

additive errors, and hence the upper bound to the probability of false

synchronization using the new technique could be much greater than the

probability of false synchronization using a Barker sequence of comparable length.

The preceding observations indicate that in spite of the apparent asymptotic efficiency of the new technique in the class of synchronization-error-correcting procedures which drop $t_s$ symbols, it is possible for the Barker sequence method to be more efficient for moderate values of $N$. For small values of $L$, however, the new technique can be much more efficient than the use of a Barker sequence of length $L$. It would appear that a combination of both techniques, using relatively infrequently occurring Barker sequences to guard against gross synchronization errors, and application of the new technique to each word, using a small value of $t_s$, would result in highly efficient synchronization protection.

It is noted that if additive errors occur so as to render the Barker sequence unrecognizable, false synchronization need not necessarily occur, since corrective action may be taken only when the Barker sequence is observed, and no action taken if it is not observed.

6.2.4. Levy's Altered Codes

6.2.4.1. A Formulation of Levy's Altered Codes

Levy's altered codes can be described as follows. If $C$ is a $t$-error-correcting BCH code, and if $\underline{x} = (x_1, x_2, \ldots, x_n)$ is a word of $C$, then the transmitter adds $g = (c_1, c_2, \ldots, c_n)$ to each codeword, and the receiver subtracts $g$ from each received word, where $g$ is a suitably chosen fixed word not in the code $C$. The mechanics associated with Levy's altered codes are the same as those associated with the technique presented in this dissertation, except for the fact that no
augmentation of words is done, and no subcode is involved. In the manner of Section 3.3, we can write the vector obtained after subtracting \( \mathbf{c} \) from the received word or
\[
\mathbf{y} = \mathbf{x}(i) + \mathbf{c}(i) - \mathbf{c} + \mathbf{e}_n
\]
if a left-shift error of order \( i \) has occurred. If the vector
\[
\mathbf{x}(i) + \mathbf{c}(i) - \mathbf{c}
\]
differs from every vector in \( \mathbf{C} \) in at least \( \delta \) positions, for \( i = 1, 2, \ldots, s, n-s, n-s+1, \ldots, n-1 \), then the code \( \mathbf{C} \), used with the vector \( \mathbf{c} \), has slip-detecting characteristic \([s, \delta]\). (Note that this \( s \) is not the \( s \) used in the expression \( z = \alpha^s \).) Levy's approach to synchronization is to find vectors \( \mathbf{c} \) corresponding to a variety of pairs \([s, \delta]\). Levy considers only vectors \( \mathbf{c} \) such that
\[
\mathbf{c}(x) = c_1 x + c_2 x^2 + \cdots + c_n x^{n-1}
\]
is of degree \( \leq r \). There is no need for such a restriction, and in fact this restriction increases the difficulty of finding desirable vectors \( \mathbf{c} \).

6.2.4.2. A Method of Immediate Synchronization-Error-Correction

Using Levy's Altered Codes

In order to compare the synchronization ability of altered codes with the synchronization ability of the new technique, we shall consider how Levy's altered codes can be used to correct additive and synchronization errors in the word immediately following a synchronization error. For each method, we will then be able to compare the number of redundant symbols used for synchronization purposes, corresponding to given values of \( t_s = 2s \).
To correct synchronization and additive errors simultaneously, we shall proceed as follows. We assume that when the receiver receives \( \chi \), he calculates the associated error vector \( \varepsilon_y \), using \( C \) as a \( t \)-additive-error-correcting code. Since the vector \( c \) is fixed, \( c(i) - c \) is fixed, so that the pattern of additive errors that \( c(i) - c \) represents will be fixed for given \( i \). Let us assume that these patterns differ considerably for different values of \( i \). Suppose that when the receiver calculates the error vector \( \varepsilon_y \) corresponding to the received word \( \chi \), this error vector is near (in the Hamming-distance sense) to \( c(i) - c \) for some \( i \). The error vector is probably not identical to \( c(i) - c \) since it may contain true additive errors, and the last \( i \) symbols are assumed random, being part of the following word. (Since in an overlap of order \( i \) the first or last symbols are assumed random, being the first or last \( i \) digits of the next word, it is assumed that the vector \( c(i) - c \) will have nonzero symbols primarily in the positions from \( s + 1 \) through \( n - s \).) Then the receiver will be relatively sure that a left synchronization error of order \( i \) has occurred. (In practice, the receiver would check to see if the calculated error pattern persisted for several successive received words, as might be done using the new technique.) Let us adopt the rule that the receiver decides that synchronization error has occurred from whose error pattern the calculated error pattern differs least (i.e., is nearest, in the Hamming distance sense). In order for the error pattern to be correctly calculated, it is necessary that

\[
t_a + w(c(i) - c) + s \leq t
\]

where \( t_a \) is the number of additive errors allowed to occur, and \( t \), it is recalled, is the number of additive errors that the code \( C \) can
correct. The term \( s \) is included to allow for any number of indicated additive errors in the overlapped portion (of up to \( s \) symbols) of the received word.

6.2.4.3. The Redundancy of the Suggested Correction Procedure Using Altered Codes

The probability that an incorrect decision (about a synchronization error of order \( i \)) is made is lower-bounded by the probability that additive errors occur in places such that the observed error pattern differs more from \( c(i) - c \) than from \( c(j) - c \) for some \( j \neq i \). The easiest way for this to occur is for additive errors to erase \( c(i) - c \). If we assume that \( w(c(i) - c) \) is approximately equal to \( \delta \), then the probability of this is approximately \( p^\delta \) (disregarding the possibility of errors elsewhere in the received word). For \( p \) small (e.g., .01), this is quite small, even for \( \delta \) relatively small (e.g., 3).

The probability of false synchronization by the receiver due to an excess of additive errors can thus be quite small (as for the new technique). We are therefore justified in comparing the above suggested technique with the new technique solely on the basis of how much redundancy each method uses for synchronization-error correction. The number of redundant symbols used for synchronization in the method suggested above is the number of redundant symbols associated with correction of \( w(c(i) - c) + s \) additive errors. Assuming \( m \) redundant symbols per additive error, this number is

\[
m \cdot \{w(c(i) - c) + s\}.
\]

This is approximately equal to \( m(\delta + s) \) if we assume that \( w(c(i) - c) \approx \delta \).
6.2.4.4. Comparison of the Two Techniques

We can now construct Table 6.4, showing the redundancy for the new technique and for the above suggested method, using altered codes. For the table, we have arbitrarily set \( \delta = 3 \); for \( p = .01 \), for example we would thus have \( p^5 = 10^{-6} \). The probability of synchronization error would be negligible for both methods assuming that with the new technique we have chosen \( t_a \) comparably large.

It can be seen from Table 6.4 that the method using altered codes is much less efficient than the new technique, even though we have assumed optimal conditions (i.e., that the \( c(i) - c \) are quite different for different \( i \)).

It is remarked that there may well be a more efficient procedure than the one suggested above by which altered codes can be used for immediate correction of synchronization errors. For a more efficient procedure than the one suggested above by which altered codes can be used for immediate correction of synchronization errors. For a more efficient procedure, of course, the use of altered codes would compare more favorably with the new technique.

6.2.5. Tong's Development of Altered Codes

Tong's work on altered codes came to the attention of the author after completion of the present research, and a detailed comparison of his procedure is not included here for that reason. His results provide means for immediate correction of synchronization errors, in many cases in the presence of additive errors, and would allow a more interesting comparison with the present technique than the procedure suggested in the preceding section. As noted in Chapter I, however,
<table>
<thead>
<tr>
<th>( m )</th>
<th>( t_s = 2s )</th>
<th>( L_n = \text{redundancy corresponding to } t_s \text{ for the new technique} )</th>
<th>( L_n = \text{redundancy corresponding to } t_s \text{ for the suggested altered codes procedure} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>5</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>64</td>
</tr>
</tbody>
</table>

\( (s = 3, \text{ codeword length } = n = 2^m - 1) \)
a considerable amount of redundancy is required for each additional
synchronization error is desired to be able to correct. For immediate
correction of synchronization errors of order up to i in either
direction the receiver must generally use the redundancy corresponding
to the capability for correction of two additive errors. As noted at
the end of Chapter IV, considerably less redundancy than this is re-
quired by the new technique (approximately on the order of one redu-
dant symbol per correctable synchronization error.)

6.3. Some Remarks Concerning the Occurrence of Uncorrectable
Synchronization Errors Using the New Technique

So long as only correctable synchronization errors occur, with
at least one complete word between synchronization errors, the new
synchronization technique is reliable, even if uncorrectable additive
errors occur occasionally. It is important to note what happens if a
gross (i.e., uncorrectable) synchronization error occurs. Suppose,
for definiteness, that a left-shift error of order i, where i > t',
occurs. Let us write i = t' + u, where u > 0. We consider three
cases:

1. u is small compared to t', i.e., u < t';
2. u is large compared to t', i.e., u > t'; and
3. u is comparable in size to t'.

In all three cases, the truncated word \( \overline{x} \) is of the form

\[
\overline{x} = (y_1, y_2, \ldots, y_{n-u}, a_1, a_2, \ldots, a_u)
\]

\[
= (x_{t_r+i+1}, x_{t_r+i+2}, x_{t_r+i+n-u}, a_1, a_2, \ldots, a_u) + (t_r+i-q) + (t_r) +
(e_1, e_2, \ldots, e_{n-u}, 0, 0, \ldots, 0)
\]
\[(x_{r_1+1}, x_{r_1+2}, \ldots, x_{r_1+n}) + c(t_1) + c(t_2) + c(t_3) + \ldots + c(t_r) + c(t_{r+1}) + c(t_{r+2}) + \ldots + c(t_{r+u})\]

where the last \(u\) positions of \(y\) contain symbols (assumed random) from the following word.

In Case 1, it will generally be the case that there are no more than \(t_a - u\) additive errors in the symbols \(x_{r_1+1}, x_{r_1+2}, \ldots, x_{r_1+n-u}\), and hence the number of nonzeros in \(e\) will generally be \(\leq (t_a - u) + u = t_a\). Thus \(e\) generally is a correctable error pattern (although only the first \(n-u\) coordinates are true additive errors), and the receiver will calculate it correctly. The receiver then calculates \(\hat{y}_c = y - e\). When the receiver calculates the synchronization error syndrome \(\hat{y}_c H_2^t\), he obtains

\[(6.9) \quad \hat{y}_c H_2^t = [c(t_1) + c(t_2) + \ldots + c(t_{r+u})] H_2^t.

This may or may not result in one of the \(t_s+1\) proper synchronization-error syndromes, but it will be the same for successive words (so long as another synchronization error does not occur). If (6.9) does not result in a proper synchronization error syndrome, the receiver will interpret that, because of the consistent occurrence of (6.9) for the synchronization-error syndrome, a gross synchronization error has occurred, and he will employ the following procedure to reestablish synchronization. First, calculate synchronization-error syndromes for the \(n\)-vectors \(y_c\) resulting from placing word marks successively after each of \(N\) successive received symbols. Now \(t_s\) successive of
these will indicate that synchronization should be at the same position. That position is the one chosen for resynchronization. If (6.9) does result in a proper synchronization-error syndrome, the only indication that a gross synchronization error has occurred is in the nature of the error pattern, e. For in Case 1, the last u symbols of e will form patterns according to the binomial distribution (assuming that the a's are random) with parameter p = 1/2. By observing this unusual behavior (or simply by observing an increase in the average additive-error rate), the receiver interprets that something is amiss, and will employ the procedure indicated above to reestablish synchroniza-

In Case 2, there are so many random symbols in e that its weight will usually exceed t_a, and the receiver will generally not determine it correctly. In this case, the synchronization-error syndrome will generally vary from word to word. The receiver will interpret that a gross synchronization error has occurred (assuming that the additive error rate is stable), and will proceed as indicated above.

In Case 3, there are sufficient random digits in e so that non-zero synchronization error syndromes will occur relatively frequently. Since the probability of synchronization errors is typically small, this behavior is unusual, and the receiver will proceed as above.

It is acknowledged that the procedures necessary to guard against gross synchronization errors are not, to say the least, well-defined operations. One of the reasons why the new technique is efficient is that it protects against a specified number of synchronization errors. If gross synchronization errors can occur, it is suggested that the receiver either periodically employ the procedure indicated above, or
else periodically employ a Barker sequence.
BIBLIOGRAPHY


[49] Stenbit, J. P., "Table of Generators for Bose-Chaudhuri Codes,"


of Electrical Engineering, March 1965.

