748201

UILU-ENG 72-2244

ERROR-CORRECTING-CODES IN COMPUTER ARITHMETIC

by

Chao-kai Liu

This work was supported in part by the Joint Services Electronics Program (U. S. Army, U. S. Navy and U. S. Air Force) under Contract DAAB-07-67-C-0199, and in part by the National Science Foundation under Grant NSF GK-24879.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Coordinated Science Laboratory University of Illinois Urbana, Illinois, 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP

3 REPORT TITLE

ERROR-CORRECTING-CODES IN COMPUTER ARITHMETIC

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5 AUTHOR(S) *(First name, middle initial, last name)*

Chao-kai Liu

6. REPORT DATE September, 1972	7a. TOTAL NO. OF PAGES 102	7b. NO. OF REFS 49

8a. CONTRACT OR GRANT NO. DAAB-07-67-C-0199; NSF GK-24879	9a. ORIGINATOR'S REPORT NUMBER(S) R-583
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* UILU-ENG 72-2244
d.	

10 DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11 SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Joint Services Electronics Program through U. S. Army Electronics Program, Fort Monmouth, New Jersey

13 ABSTRACT

Arithmetic codes are useful for error-control in digital computation as well as in data transmission. These codes are especially suitable for checking or correcting errors in arithmetic processors due to carry propagation. Two known classes of arithmetic codes are the small-distance high-rate perfect single-error correcting codes and the large-distance low-rate Mandelbaum-Barrows codes. These codes are analogous to the Hamming codes and the maximum-length sequence codes in parity-check block codes respectively. Most other arithmetic codes known have been obtained by computer-search. The discovery for a systematic way of constructing arithmetic codes with intermediate-rate and intermediate-distance has been the subject of research for many years. Finding simpler decoding algorithms is another major unsolved problem in arithmetic codes. Decoding for arithmetic codes by matching the orbits or permuting the residues associated with the codes is straightforward but largely impractical. A particularly interesting question is the possibility of decoding arithmetic codes by majority-logic. In ~~this thesis~~, ~~we have~~ constructed a class of intermediate-rate intermediate-distance binary cyclic arithmetic codes. A majority-logic decoding scheme is developed for the code constructed. This majority-logic decoding scheme is also applicable to a large class of cyclic AN-codes generated by the primitive cyclotomic factors. A new checking technique for the binary adders has been developed. This separate checking technique uses less redundancy than that required for a triplicated system, and the decoding procedure is simple. The application of majority-logic decodable arithmetic codes to the error control in high speed multiplier has been examined. By using the concepts of number theory, the upper and lower bounds on any cyclic AN-code have been formulated.

DD FORM 1 NOV 65 1473

ia

Security Classification

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Cyclic arithmetic codes						
Computer reliability						
Majority-logic decoding						
Minimum distance						
Separate checking						
Partial errors						

Security Classification

ERROR-CORRECTING-CODES IN COMPUTER ARITHMETIC

Chao-kai Liu, Ph.D.
Coordinated Science Laboratory and
Department of Electrical Engineering
University of Illinois at Urbana-Champaign, 1972

Arithmetic codes are useful for error-control in digital computation as well as in data transmission. These codes are especially suitable for checking or correcting errors in arithmetic processors due to carry propagation.

Two known classes of arithmetic codes are the small-distance high-rate perfect single-error correcting codes and the large-distance low-rate Mandelbaum-Barrows codes. These codes are analogous to the Hamming codes and the maximum-length sequence codes in parity-check block codes respectively. Most other arithmetic codes known have been obtained by computer-search. The discovery for a systematic way of constructing arithmetic codes with intermediate-rate and intermediate-distance has been the subject of research for many years.

Finding simpler decoding algorithms is another major unsolved problem in arithmetic codes. Decoding for arithmetic codes by matching the orbits or permuting the residues associated with the codes is straightforward but largely impractical. A particularly interesting question is the possibility of decoding arithmetic codes by majority-logic.

In this thesis, we have constructed a class of intermediate-rate intermediate-distance binary cyclic arithmetic codes. A majority-logic decoding scheme is developed for the code constructed. This majority-logic decoding scheme is also applicable to a large class of cyclic AN-codes generated by the primitive cyclotomic factors. A new checking technique for the binary adders has been developed. This separate checking technique uses less redundancy than that required for a triplicated system, and the decoding procedure is simple. The application of majority-logic decodable arithmetic codes to the error control in high speed multiplier has been examined. Furthermore, some theoretical results about the structure of arithmetic codes have been obtained. By using the concepts of number theory, the upper and lower bounds on any cyclic AN-code have been formulated.

In summary, the results presented in this thesis give a better belief that the use of arithmetic coding for the error control in digital computing systems is indeed promising.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author wishes to express his deepest gratitude to his thesis advisor, Professor Robert T. Chien, for his inspiring guidance, encouragement, advice, and suggestions throughout the period of research for this thesis.

The author wishes to extend his thanks to Professor C. L. Chen for his many helpful suggestions. Thanks are also due to the members of the Coding Group at the Coordinated Science Laboratory.

1. INTRODUCTION

1.1 Error Control in Digital Computing Systems

The rapid growth in the size and speed of modern day digital computing systems has placed stringent reliability demands on the central processors, especially those concerned with space missions, and real time computer applications. In a space mission, the computers require high reliability because of the long mission time during which manual repair of a failure is impossible; those real time computer applications require high reliability because of the serious consequences error might cause in terms of inconvenience or costly mistakes. Consequently, the error control techniques that can be used to improve the reliability of a digital computing system are very important.

There are two major techniques to deal with the faults in a digital system, one is the use of "software diagnosis," and the other is the design of "hardware checking circuits." In the software fault diagnosis approach, a specially organized checking algorithm is incorporated into the program and the arithmetic results are verified at periodic intervals by means of additional instructions which are redundant in a normally functioning computer. Correction procedures in case of an error also must be programmed. The cost of this approach consists of the additional programming effort, the execution time and storage requirements for the instructions. The logic design of the arithmetic processor remains unchanged. An alternative to the software fault diagnosis is the design of hardware checking circuits. In this approach, the results and/or the operands of each arithmetic operation are automatically tested for acceptability without programmed commands. The indication of an unacceptable result initiates the error correction. The cost of the hardware checking approach consists of the additional logic circuits and the increased execution time of the operations.

Most of the past emphasis to improve the reliability of digital systems has been on making more careful designs and using more reliable parts. People have realized that it is also possible to design the system that it operates correctly even when some of its parts fail. Such approaches invariably use redundancy. The above mentioned "software diagnosis" and "hardware checking" techniques are both the approaches that use redundancy. In general, by utilizing different types of redundancy or inserting the redundancy in different parts of a system, we can have different error control schemes to improve the reliability of digital computing systems.

The most commonly used scheme for hardware checking is the well-known replication of processors. Triplication of the processor with majority output voting gives error correction; and duplication of the processor with output comparison, followed by further diagnosis in the case of disagreement gives error detection. Both methods achieve checking at the cost of complete replication of the processor. An alternative to the replication of processors is the use of coding. The encoding of the operands followed by the application of decoding algorithm to the results requires only a fractional increase in cost, but does not guarantee complete error control.

The coding approach is quite different from the technique of replication of processors. In a hardware replication scheme, we simply organize a number of less reliable components to form a reliable computing device;

while in a coding scheme the computing device is fixed. We incorporate redundancy into data themselves which are being processed; in other words the input data are encoded by some coding technique so that the computational errors in the output can be detected and/or corrected. For example, if we use an AN-code, the operand N is encoded by multiplication with a fixed integer A before being represented in the digital computer. Thus, only a fraction 1/A of the possible representations are ever intentionally used in the computer, the resultant redundancy can then be used for error correction and detection.

If coding technique is applied to improve the reliability of a computing system, there is no need to supply any software diagnostic routines, and the decoding circuits can detect and/or correct error immediately after it occurs so that any unnecessary propagation of errors can be avoided. Also, the verification of the acceptability of input operands supplied by another processor provides an approach to error-free operation of systems employing intercommunicating processors. In fact, some of the computing systems designed for the purpose of being used in spacecraft on long missions to the outer planets of the solar system (e.g. the JPL Self-Testing-and-Repair Computer) have employed the coding scheme to fulfill ultra-reliable demands.

Since the appearance of Shannon's theory on data transmission, the problem of how to design an efficient scheme by which the information can be transmitted reliably across noisy channels has become a subject of continuing importance. The invention of many intelligent codes by using some algebraic structures has highlighted the practical aspects of the coding theory. However, the application of error-detecting and error-correcting codes to a digital computing system is not as straightforward as it has been to a data communication system such as space communication or long-distance telephone transmission. Errors in a digital computing system are of different nature from the errors in a communication channel. Those codes which perform powerfully in data transmission cannot be used to control errors caused by logic faults in arithmetic operations. In general, there are two types of errors that occur in a digital computing system, transmission errors and arithmetic errors. Transmission errors usually occur in a large digital system when many intercommunicating data processors are employed, the data flow across different units may be affected by noise. These errors are of the same nature as errors in a communication system. Arithmetic errors occur in the operation of an arithmetic processor whenever its actual behavior deviates from the expected behavior. Those errors are caused by faults of logic circuits within the processor. The main difference between transmission errors and arithmetic errors is that the latter may cause carry or borrow propagations so that a single fault in logic may cause several output bits to be in error. Therefore, an efficient error-detecting or error-correcting code used in a digital computing system must be able to deal with both types of errors. The codes that are specially designed for the use of detecting and/or correcting errors in digital computation as well as data transmission are called arithmetic codes. It is the purpose of this thesis to study the error control in digital computing systems by using arithmetic coding.

The theory of arithmetic coding has drawn the attention of many researchers since Diamond [20] first proposed his paper in 1955. Much research has been done on detecting and correcting errors in arithmetic operations. Most of this work has been concerned with the arithmetic AN-code, (we shall make somewhat detailed discussion on the fundamental concepts and definitions of arithmetic AN-codes in the following chapters.) Massey [36] presented an excellent survey of the early work on these codes. The popularity of these codes is due to the fact that they have a nice algebraic structure and can be used in a computer with little or no change in the circuit design. Although many important results have been obtained in the theory of arithmetic codes, there are still two major unsolved problems:

(1) The problem of how to find a systematic way to construct a multiple error-correcting arithmetic code. The existing codes are either of high-rate, low error-correcting capability or of high error control power but low information rate. The codes with intermediate rate and error control capability are usually obtained by computer search. Therefore, a systematic way of constructing good arithmetic AN-code is much in need.

(2) A more important problem is the implementation (or decoding) of the arithmetic codes. The practicality of using coding in a digital system depends heavily on how complex the decoders are. Unfortunately, the decoding problem of arithmetic codes is quite complex in general because of the carry propagation caused by errors. The only known decoding method is the "permutation of residues" [29,30], which is basically a table-look-up approach and impractical. How to find a

decoding scheme that can be easily implemented is then a problem of particular significance. One approach is suggested by the following consideration: since the majority-logic-decoding (MLD) scheme [6] is by far the easiest one that can be applied in a variety of ways to communication channels, it would certainly be of interest to see if the decoding of arithmetic codes could also be handled by using majority logic.

The solutions of these two problems will greatly increase the feasibility and effectiveness of using arithmetic coding scheme in digital computing systems. These are two of the topics that are investigated in this thesis.

1.2 Scope of the Thesis

The main objectives of this thesis are to construct new arithmetic codes with better performance, to devise decoding schemes for arithmetic codes that can be practically implemented, to develop efficient checking techniques and checking model for arithmetic operations in computer, and to study the theoretical structures of the arithmetic codes.

Chapter 2 is a general review of the background, definitions and concepts that are helpful to the discussion of the material presented in later chapters. Concepts of error weight and code distance, used for determining the codes' error correcting capability are discussed. The cyclicity of the codes and the analogies between the AN-codes and parity-check codes are briefly examined.

Chapter 3 considers the construction of new arithmetic codes with multiple-error-correcting capability. A fast algorithm for finding the binary representation forms of certain integers is derived. By using the number theoretical properties of these integers, a class of new codes is constructed. The error control power of these codes can be determined analytically. The root-distance relation of a class of cyclic AN-code is investigated, the result is completely analogous to the BCH bound for the cyclic parity-check codes. In addition, the bounds on the error-correcting capability of any cyclic AN-codes are determined.

Chapter 4 describes a majority-logic decoding scheme for some cyclic AN-codes. This decoding scheme is completely different from the permutation of residue approach, the implementation of the codes can be achieved by using majority logic alone. The application of the majority-logic decoding algorithm to the cyclic AN-codes constructed in Chapter 3 is examined, and a multi-step majority decoding scheme that can be applied to larger classes of cyclic AN-codes is obtained.

Chapter 5 suggests a new spearate concurrent checking technique for the arithmetic operations in a digital computer. The application of error-correcting AN-codes shows the effectiveness of the separate checking technique. Finally, a study of the error control in high speed arithmetic is included.

Some conclusions on the results presented in this thesis and suggestions on possible future work are also discussed in the last chapter.

2. PRELIMINARIES AND BACKGROUND

2.1 The Arithmetic AN-Code

The arithmetic AN-code is specially designed for the use of correcting and/or detecting errors in digital computation as well as data transmission. The code is a set of integers of the form AN, where A is a fixed integer called the generator of the code. For each integer N from the information set $Z_I = \{0,1,2,\ldots, B-1\}$, there is a corresponding code number AN, hence the integer N is "encoded" by multiplication with the fixed generator A before being represented in the computer arithmetic. Thus only a fraction 1/A of the possible representations are actually used in the computer, the resultant redundancy can then be used for error detection and correction.

The AN-code possesses linear property, for

$$AN_1 + AN_2 = A(N_1+N_2)$$

so that the coded form for the sum of two information integers is the sum of the coded integers. Therefore, two coded numbers can be added in an ordinary adder; if the sum is not a properly coded number error is checked.

As mentioned in the introductory chapter, the errors that occur in a digital computing system are more complicated than the transmission type of errors. Consequently, the identification of the likely modes of failure in a computing system and the formulation of a convenient description of the resultant errors in its output must be considered differently from that of a data transmission system.

The measure of the weight of an arithmetic error and the error-correcting capability of an AN-code have been developed by several authors [35,37], a brief summary of these results will be made in the following sections.

2.2 Arithmetic Weight and Distance of the AN-Code

Because of its dominating practical importance, only binary arithmetic will be considered in this section. We should remark, however, that all the results can be extended to an arbitrary radix system.

Any positive integer I can be expressed in binary form or binary sequence as:

$$I = a_n 2^n + a_{n-1} 2^{n-1} + \dots + a_1 2^1 + a_0$$

$$= (a_n\, a_{n-1}\, a_{n-2} \cdots a_1 a_0)$$

where

$$a_i = 0 \text{ or } 1$$

The circuitry for adding two integers in radix-2 form can be considered as a sequence of elemental adding units. Suppose that two binary integers $I_1 = (a_n\, a_{n-1} \dots a_1 a_0)$ and $I_2 = (b_n b_{n-1} \dots b_1 b_0)$ are added, then the i-th elemental adding unit performs the addition of a_{i+1}, b_{i+1} and the carry bit from the (i-1)-th unit. Each unit forms a sum bit and a carry bit to the next unit. A "single failure" in the adder is then considered as an incorrect sum formed by one adding unit or the generation of an incorrect-carry bit by one adding unit; hence a single failure at the i-th unit may cause an error of value $\pm 2^i$ (due to incorrect sum) or $\pm 2^{i+1}$ (due to incorrect carry). If the erroneous sum differs from the correct sum by an amount E, it is then quite natural to think of the weight of E, i.e. the number of errors, as the least number of terms of the form $+2^i$ or -2^i whose sum is E. This measure of the weight of an error is called "arithmetic weight" and can be formalized as follows:

Definition 2.2.1: The arithmetic weight of an integer E (which may be positive, negative or zero), denoted W(E), is the minimum number of non-zero coefficients in the modified binary form of E:

$$E = e_0 + e_1 2^1 + e_2 2^2 + \ldots + e_i 2^i + \ldots$$

where

$$e_i = 0, 1, \text{ or } -1 \tag{2.2.1}$$

Although the modified binary form of an integer is not unique in general, there is a particular modified binary form introduced by Reitwiesner [43] which is unique and which has been proved of much importance in the theory of arithmetic code, called the non-adjacent-form (or NAF for short). A modified binary form is said to be a NAF if the coefficients e_i's in (2.2.1) satisfy $e_i e_{i+1} = 0$ for $i = 0, 1, 2, \ldots$, in other words if there are no two adjacent non-zero coefficients in the form.

The most important property of the NAF of an integer I is that no modified binary form of I has fewer non-zero coefficients than the NAF of I [43]. Therefore, the arithmetic weight of any integer equals the number of non-zero coefficients in its NAF.

Definition 2.2.2: The arithmetic distance between the integers I_1 and I_2, denoted $D(I_1,I_2)$, is the arithmetic weight of their difference, i.e. $D(I_1,I_2) = W(I_1-I_2)$.

It can be shown [35] that the arithmetic distance satisfies the following three properties:

$$D(I_1,I_2) = D(I_2,I_1) \qquad \text{(symmetry)}$$
$$D(I_1,I_2) \geq 0 \quad \text{with equaltiy iff } I_1 = I_2 \qquad \text{(positive definite)}$$
$$D(I_1,I_2) \leq D(I_1,I_3) + D(I_3,I_2) \qquad \text{(triangle inequality)}$$

Hence, any set of integers with arithmetic distance taken as the measure of "distance" form a metric space.

Definition 2.2.3: The minimum arithmetic distance, D_{min}, of an arithmetic AN-code is the minimum of the arithmetic distances between all pairs of distinct code words in the code.

Since the difference of two code words is another code word, the D_{min} of an arithmetic AN-code is equal to the minimum arithmetic weight, W_{min}, of the B-1 non-zero code words in the code.

The error-correcting capability of an AN-code is completely determined by the minimum distance of the code as Massey [35] has shown that an AN-code with $D_{min} = 2t + 1$ can correct all arithmetic errors of weight t or less, and an AN-code with $D_{min} = t + 1$ can detect all arithmetic errors of weight t or less. Therefore, the knowledge of minimum distance of a given arithmetic AN-code has become a very important subject.

2.3 Cyclic AN-Code and Modular Distance

Suppose that an integer I has a binary n-tuple form ($a_{n-1}\ a_{n-2} \cdots a_1\ a_0$). We denote by P(I) the integer whose binary n-tuple form is the left cyclic shift of that for I, namely ($a_{n-2}\ a_{n-3} \cdots a_0\ a_{n-1}$). If the largest code word A(B-1) in an AN-code requires n bits for its radix-2 form, then we define:

Definition 2.3.1: An AN-code is cyclic if its set of codewords is closed under cyclic shifting, i.e. if for every code word AN, the integer P(AN) is another code word.

It has been shown [11,36] that the generator A and the number of codewords B in a cyclic AN-code satisfy $AB = 2^{e(A)} - 1$, where e(A) denotes the least positive integer such that A divides $2^{e(A)} -1$, and is called the exponent of 2 modulo A [49]. Therefore, the block length of a cyclic AN-code generated by A is e(A). Mathematically, all the codewords in a cyclic AN-code form a principal ideal in the ring of integers modulo $2^{e(A)}-1$, with A as the ideal generator.

In a cyclic AN-code, the addition of codewords is the modulo $2^{e(A)}-1$ arithmetic, a negative integer -I is represented by its one's complement, i.e. $2^{e(A)}-1-I$. The arithmetic weight of negative integers then is defined for their one's complements. This leads naturally to a modified measure of the weight of the integers in the ring of integers modulo 2^n-1:

Definition 2.3.1: The "modular weight" of an integer F in the ring of integers modulo 2^n-1 is the minimum of W(F) and $W(2^n-1-F)$, and is denoted $W_m(F)$.

Definition 2.3.2: The "modular distance" between the integers I_1 and I_2 in the ring of integers modulo 2^n-1 is the modular weight of their difference and is denoted $D_m(I_1,I_2)$, i.e. $W_m(I_1-I_2)$.

With the definitions of modular weight and modular distance, we can show that the error correcting capability of a cyclic AN-code in modulo arithmetic is completely determined by the modular minimum distance of the code, a result which is analogous to the fundamental result stated in the previous section.

2.4 Arithmetic Codes for Single Error

Single error detecting and single error correcting arithmetic AN-codes have been thoroughly investigated by Brown [8] and Peterson [37]. Some well-krown results will be summarized in the following theorems.

Theorem 2.4.1: An AN-code with generator $A > 1$ and odd can detect any single error in radix-2 arithmetic.

Theorem 2 4.2: Let $M_2(A,d)$ be the smallest positive integer whose product with A has arithmetic weight less than d, then the arithmetic AN-code with $0 \leq N < M_2(A,d)$, has $D_{min} \geq d$.

Theorem 2.4.3: Let $A > 1$ be an odd integer, e be the smallest integer that satisfies $2^e \equiv 1 \bmod A$, then $M_2(A,3) = (2^e-1)/A$. Similarly, if $2^e \equiv -1 \bmod A$, then $M_2\ (A,3) = (2^e+1)/A$.

Theorem 2.4.4: If A is an odd prime and if 2 is a primitive root of A, then $M_2(A,3) = (2^{\frac{e}{2}} +1)/A$. Similarly, if -2 is the primitive root of A, then $M_2(A,3) = (2^{\frac{e}{2}} -1)/A$, where $e = A-1$.

The above results yield a way of constructing any single error-detecting and single error-correcting arithmetic AN-codes. More importantly, the single error-correcting codes mentioned above are perfect codes or sphere-packed codes, i.e., any integer in the ring of integers modulo $2^e -1$ is at most distance-one away from some code word. These codes are analogous to the well-known Hamming codes in parity-check block codes.

2.5 Mandelbaum-Barrows Equidistant Codes

The Mandelbaum-Barrows codes [5,32] are the first systematically constructed class of AN-codes with $D_{min} > 3$. They are also the first AN-codes to be recognized as cyclic. The structure of these codes is simple:

Theorem 2.5.1: Let B be an odd prime with 2 as its primitive root, let the generator A of the cyclic AN-code be:

$$A = (2^{e(B)}-1)/B = (2^{B-1}-1)/B$$

Then the minimum distance of the code is $\lceil (B+1)/3 \rceil$, where $\lceil x \rceil$ denotes the integral part of x.

For a properly chosen B, these codes can correct any multiple arithmetic errors. Moreover, these codes also possess an interesting property that all the nonzero code words are some cyclic shifts of the generator A, hence they all have the same arithmetic weight $\lceil (B+1)/3 \rceil$. These codes are analogous to the maximum-length sequence codes in parity-check block codes, they are of large-distance but low-rate, and stand at the opposite end of the coding spectrum from the perfect single-error-correcting codes of the previous section.

2.6 Analogy Between Arithmetic AN-Codes and Parity-Check Codes

We begin with a brief review of the theory of cyclic parity-check codes. Extensive treatments of this subject can be found in Peterson [37] and Berlekamp [6].

With a polynominal $f(x) = \sum_{i=o}^{n-1} f_i x^i$ of degree less than n with coefficients in a field F, we associate the vector $f = [f_{n-1}, f_{n-2}, \ldots, f_1, f_0]$ in the vector space F^n. A parity-check code is simply a set of such vectors which form a subspace of the linear vector space F^n. The Hamming weight of a vector (or a code word) is defined as the number of non-zero components in the vector, and the Hamming distance of two vectors is the number of positions in which the two vectors differ. Since the vectors are closed under addition, the minimum Hamming distance, d_{min}, of such a code equals the minimum Hamming weight of the non-zero vectors in the code.

A parity-check code is cyclic if the cyclic shift of every codeword is also a code word. If we denote the cyclic shift of $\underline{f}$ by $\underline{f}'$, then the corresponding polynomial $f'(x)$ of $\underline{f}'$ is obtained by multiplying $f(x)$ by x and take the residue modulo x^n-1. It can be shown [37] that the code words of a cyclic parity-check code form a principal ideal in the polynomial ring modulo x^n-1 over some field F. Therefore, each code word has a corresponding polynomial which is a multiple of some fixed polynomial $g(x)$, called the generator polynomial of the ideal. In addition, $g(x)$ is monic and divides x^n-1. The polynomial

$h(x) = (x^n-1)/g(x)$ is called the parity-check polynomial of the code, and n, the length of the code is chosen as the least positive integer such that $g(x)$ divides x^n-1.

In light of the above, the analogy of cyclic parity-check codes with the cyclic arithmetic AN-codes is evident. The analogous quantities are:

Cyclic parity-check codes	Cyclic arithmetic AN-codes
x	2
$g(x)$	A
$h(x)$	B
x^n-1	2^n-1
period of $g(x)$	exponent of 2 mod A
d_{min}	D_{min}

The closeness of these analogies between cyclic parity-check codes and cyclic AN-codes strongly implies that there exist cyclic AN-codes analogous to many important cyclic parity-check codes, in particular, the Bose-Chandhuri-Hocquenghem codes and the majority-logic decodable geometry codes. Many investigations have been made on this problem, but to date very little is known.

2.7 Remarks

There are other important results and concepts in the theory of arithmetic codes that have not been mentioned. It should be remarked that the purpose of this chapter is to provide some of the terminology and concepts which will be used with regard to the following discussion. The material presented here is just a brief review of the arithmetic coding theory, more detailed treatments of the related subjects can be found in [35,36,37] etc.

3. CONSTRUCTION OF MULTIPLE-ERROR-CORRECTING ARITHMETIC AN-CODES

3.1. Introduction

The theory of arithmetic coding has been developed since Diamond [20] first proposed his paper in 1955. Single error-correcting arithmetic codes have been investigated by Brown [8], Peterson [38], and Bernstein [7]. Some results of multiple error-correcting arithmetic codes have been reported by Barrows [5], Mandelbaum [32], Chang and Tsao-Wu [11], Chien, Hong and Preparata [18].

Since the error-correcting capability of arithmetic codes is directly related to the minimum distance of the codes, an analytical way of calculating the distance for arithmetic codes is important. Chien, Hong and Preparata [18], Tsao-Wu and Chang [11] independently discovered a computational algorithm for minimum distance of cyclic AN-codes. By using number theoretic concepts, they divide the code words of a cyclic AN-code into a number of disjoint sets, each set of code words is called an orbit. The code words in the same orbit are the AN's such that $N \equiv (k \cdot 2^i) \bmod B$, for some fixed k, and $i = 0,1,2,\ldots,n$, $AB = 2^n-1$. Therefore, each k defines an orbit. It can be seen that the code words in the same orbit are of equal weights, hence the minimum weight of the code is the minimum of all the weights of different orbits.

Perhaps the most interesting property of this orbital theory is the following theorem:

Theorem: The weight of a code word AN, represented in binary form

$$AN = a_{n-1}a_{n-2}\cdots a_1a_0$$

where $a_{n-i} = ((N\cdot 2^i) \bmod B) \bmod 2$, $1 \le i \le n$, is given by the number of residues $N\cdot 2^i \bmod B \in M_B$; where M_B, the middle-third region of B is defined

$$M_B = \{x | B \le 3x < 2B\}.$$

The result of this theorem enables us to calculate the arithmetic weight of each orbit by counting the number of residues in M_B. Computationally, this orbital algorithm might still be difficult especially when the code contains a large number of orbits. An easier algorithm to find the minimum distance, or perhaps some closed-form formula, is necessary. Furthermore, in practical use the discovery of systematic way of synthesizing a code which corrects a specified number of errors is more important. The problem that how to construct a code, rather than analyze a given code, is therefore of both theoretical and practical significance.

In this chapter, we shall construct a class of cyclic AN-codes whose generators are the products of cyclotomic factors. The multiple error-correcting capability of these codes can be determined exactly. These codes fill in the gap between the single-error-correcting codes and the Mandelbaum-Barrows codes as far as the distance and rate are concerned. The construction of these codes is based on some interesting properties possessed by the binary forms of certain integers which are used as the generators of the codes.

We shall first establish a fast algorithm for finding the binary forms of certain integers, by using the complementary, symmetric properties we shall construct a class of cyclic AN-codes. Also, the analogy between cyclic AN-codes and cyclic polynomial codes will be studied, we shall demonstrate a root-distance relationship of a class of AN-codes which is shown to be exactly the same as the well-known BCH bound on the distance of cyclic polynomial codes. Furthermore, by using number theoretic concepts, the strict upper and lower bounds on the error-control capability of any cyclic AN-codes are derived. In general, the results obtained in this chapter are all theoretical, the possible practical applications and implementations of the codes constructed will be discussed in the following chapters.

3.2. Binary Representation Forms of Certain Integers

The representation of integers in binary form has been under investigation for various applications. In particular, the binary representation is of importance in fast computer arithmetics, number systems, algebraic coding theory and arithmetic codes. In this section, the binary representation of the integer A of the form $(2^n-1)/(2^{n_1}-1)\cdot(2^{n_2}-1)$, with $n_1\cdot n_2 = n$ and n_1,n_2 being two relatively prime integers, is considered. Firstly, some interesting properties of the binary representation form will be presented. It will be shown that the binary form of A has 1's and 0's at certain fixed positions with complementary symmetrical properties. Secondly, a simple and fast algorithm for finding the binary form will be introduced. From this algorithm, the binary form of a large integer can be easily obtained without any calculation.

Formulations and Notations

Let A be an integer of the form

$$A = (2^n-1)/(2^{n_1}-1)(2^{n_2}-1)$$

where n_1 and n_2 are two relatively prime positive integers with $n_2 > n_1$ and $n = n_1 n_2$. The integer A can be uniquely expressed in the following binary form

$$A = \sum_{i=0}^{n-1} a_i 2^i \quad \text{with } a_i = 0 \text{ or } 1 \text{ for } 0 \leq i \leq n-1$$

Let $B = (2^{n_1}-1)(2^{n_2}-1)$, then $AB = 2^n-1$. The coefficients a_i in the binary representation form of A can be determined [12] by

$$a_i = (2^{n-i} \bmod B) \bmod 2 \tag{3.2.1}$$

where $(2^{n-i} \bmod B)$ is the residue of 2^{n-i} modulo B. The above equation states that $a_i = 0$ if and only if $(2^{n-i} \bmod B)$ is an even integer.

Consider the binary form of the integer $A \cdot 2^{n_1}$,

$$C = A \cdot 2^{n_1}$$
$$= \sum_{i=0}^{n-1} c_i 2^i \qquad c_i = 0 \text{ or } 1 \text{ for } 0 \leq i \leq n-1$$

By (3.2.1),

$$c_i = [2^{n-i} \cdot 2^{n_1} \bmod B] \bmod 2 \tag{3.2.2}$$

Similarly, let

$$D = A \cdot (2^{n_1}-1)$$
$$= \sum_{i=0}^{n-1} d_i 2^i \qquad d_i = 0 \text{ or } 1 \text{ for } 0 \leq i \leq n-1$$

then,

$$d_i = [2^{n-i}(2^{n_1}-1) \bmod B] \bmod 2 \tag{3.2.3}$$

Since

$$D = (2^n-1)/(2^{n_2}-1) = 2^{(n_1-1)n_2} + 2^{(n_1-2)n_2} + \dots + 2^{n_2}+1, \tag{3.2.4}$$

$$d_i = 1 \quad \text{if } i = tn_2 \text{ for } 0 \leq t \leq n_1-1$$

$$= 0 \quad \text{otherwise}$$

Let us denote

$$[2^{n-i} \cdot 2^{n_1} \bmod B] = x_i$$

$$[2^{n-i} \quad \bmod B] = y_i$$

$$[2^{n-i}(2^{n_1}-1) \bmod B] = z_i$$

where $0 < x_i, y_i, z_i < B$. Note that $c_i \equiv x_i \bmod 2$, $a_i \equiv y_i \bmod 2$ and $d_i \equiv z_i \bmod 2$.

Properties

Property 1: $C_o = 0$ and $a_o = 1$.

Proof: Since B divides 2^n-1, $2^n \equiv 1 \bmod B$ and $y_o = 1$. In addition, since $2^n \cdot 2^{n_1} \equiv 2^{n_1} \bmod B$, $x_o = 2^{n_1}$. Thus $c_o = 0$ and $a_o = 1$.

Q.E.D.

Suppose that $i > 0$. Let $n-i = k$. By Euclidean division algorithm we have $k = qn_1+r$, where $0 \leq r < n_1$. Notice that

$B = (2^{n_1}-1)(2^{n_2}-1)$ implies:

$$2^{n_2}(2^{n_1}-1) \equiv 2^{n_1}-1 \mod B$$

Thus

$$2^{jn_2}(2^{n_1}-1) \equiv 2^{(j-1)n_2}(2^{n_1}-1)$$

$$\equiv 2^{n_1}-1 \quad \text{mod } B \text{ for } j \geq 0 \qquad (3.2.5)$$

Now, equation (3.2.5) can be used successively to obtain

$$2^{n-i} = 2^k = 2^{k-n_1}(2^{n_1}-1) + 2^{k-n_1}$$

$$\equiv 2^{(k-n_1) \bmod n_2}(2^{n_1}-1) + 2^{k-2n_1}(2^{n_1}-1) + 2^{k-2n_1}$$

$$\equiv (2^{n_1}-1)[2^{(k-n_1) \bmod n_2} + 2^{(k-2n_1) \bmod n_2}] + 2^{k-3n_1}(2^{n_1}-1)+2^{k-3n_1}$$

$$\equiv (2^{n_1}-1)[2^{(k-n_1) \bmod n_2} + 2^{(k-2n_1) \bmod n_2} + 2^{(k-3n_1) \bmod n_2}] +$$
$$2^{k-4n_1}(2^{n_1}-1) + 2^{k-4n_1}$$

$$\vdots$$

$$\equiv (2^{n_1}-1)[2^{(k-n_1) \bmod n_2} + 2^{(k-2n_1) \bmod n_2} + \dots + 2^{(k-qn_1) \bmod n_2}] +$$
$$2^{k-qn_1} \pmod B$$

Let

$$S = [2^{(k-n_1) \bmod n_2} + 2^{(k-2n_1) \bmod n_2} + \dots + 2^{(k-qn_1) \bmod n_2}]$$

Then

$$2^{n-i} = 2^k \equiv (2^{n_1}-1)\, S + 2^r \quad \bmod\ B \tag{3.2.6}$$

Similarly,

$$2^{n-i} \cdot 2^{n_1} \equiv (2^{n_1}-1)\, S+2^r + (2^{n_1}-1)2^{k \bmod n_2} \quad \bmod B \tag{3.2.7}$$

Notice that there are q terms in S, each term is a power of 2. In addition, these terms are distinct, for otherwise,

$$k-mn_1 \equiv k-m'n_1 \bmod n_2 \text{ for some } 0 \leq m,\ m' \leq q < n_2$$

This would imply $m-m' \equiv 0 \bmod n_2$. Since $(n_1,n_2)=1$ and $0 \leq m,m' < n_2$, the congruence is impossible unless $m = m'$.

For $i > 0$, $q \leq n_2-1$. Since all the q terms in S are distinct, $q < n_2$, and $2^{n_2}-1 = 1+2^1+2^2+\dots+2^{n_2-1}$, we have $S \leq 2^{n_2}-2$. Also, $2^r < 2^{n_1}-1$; thus:

$$(2^{n_1}-1)\, S+2^r < (2^{n_1}-1)(2^{n_2}-2) + (2^{n_1}-1) = B$$

By (3.2.6), the residue of 2^{n-i} modulo B is equal to $(2^{n_1}-1)\, S+2^r$. Thus we have

Property 2: $y_i = (2^{n-i} \bmod B) = (2^{n_1}-1)\, S+2^r$, where $i \neq 0$

$$S = 2^{(k-n_1) \bmod n_2} + 2^{(k-2n_1) \bmod n_2} +\dots+ 2^{(k-qn_1) \bmod n_2}$$

$$k = n-i = qn_1 + r, \qquad 0 \leq r < n_1 \tag{3.2.8}$$

Similarly, the following property can be obtained:

Property 3: $y_i = (2^{n_2}-1)\,T + 2^t$, where $i \neq 0$

$$T = 2^{(k-n_2) \bmod n_1} + 2^{(k-2n_2) \bmod n_1} + \dots + 2^{(k-pn_2) \bmod n_1}$$

$$k = n-i = p \cdot n_2 + t, \quad 0 \leq t < n_2 \tag{3.2.9}$$

Theorem 3.2.1: $x_i < y_i$ for $0 < i \leq n_1$.

Proof: For $0 < i \leq n_1$, $n - i = n_1 n_2 - i = (n_2-1)n_1+(n_1-i)$, we have

$$q = n_2-1,\ r = n_1-i$$

Since $2^{n-i} \cdot 2^{n_1} = 2^{n_1n_2+n_1-i} = 2^{n_1n_2} \cdot 2^r \equiv 2^r \bmod B$,

$$x_i = (2^{n-i} \cdot 2^{n_1} \bmod B) = 2^r \tag{3.2.10}$$

By (3.2.8), we have $y_i > x_i$.

Q.E.D.

Property 4: $c_i = 0$, $a_i = 1$ for $0 < i < n_1$; and $c_{n_1}=1$, $a_{n_1} = 0$.

Proof: By (3.2.10), x_i is even if $0 < i < n_1$ and x_{n_1} is odd.

Therefore, $c_i = 0$ for $0 < i < n_1$ and $c_{n_1}=1$. From theorem 3.2.1,

$x_i - y_i < 0$. Note that $z_i \equiv x_i-y_i \bmod B$, we have

$$z_i = B + x_i - y_i$$

From (3.2.4), we see that z_i is even. Since B is odd, $y_i = B + x_i-z_i$ is odd if $0 < i < n_1$; and y_{n_1} is even. Hence, $a_i = 1$ for $0 < i < n_1$

and $a_{n_1}=0$.

Q.E.D.

Theorem 3.2.2: $x_i > y_i$ for $n > i > n_1$.

Proof: Let $S' = S + 2^{k \bmod n_2}$. Then there are $q + 1 \leq n_2 - 1$ distinct terms in S'. It is easy to see that $S' \leq 2^{n_2}-2$, and $(2^{n_1}-1)\, S' + 2^r < B$. Thus, by (3.2.8)

$$x_i = (2^{n_1}-1)S' + 2^r$$

$$= y_i + (2^{n_1}-1) \cdot 2^{k \bmod n_2} \qquad (3.2.11)$$

and

$$x_i > y_i \quad \text{for} \quad n > i > n_1.$$

Q.E.D.

Property 5: For $i \neq 0$, $a_i = 0$ if i is a multiple of n_1 or n_2.

Proof: Case 1: $i = sn_1$ for $0 < s \leq n_2-1$. Since $k = n-i = (n_2-s) \cdot n_1$, $r = 0$. Equation (3.2.8) can be written as:

$$y_i = (2^{n_1}-1)[2^{(k-n_1) \bmod n_2} + \ldots + 2^0] + 2^0$$

Therefore, y_i is even and $a_i = 0$.

Case 2: $i = pn_2$ for $0 < p \leq n_1-1$. By (3.2.9),

$$y_i = (2^{n_2}-1)[2^{(k-n_2) \bmod n_1} + \ldots + 2^0] + 2^0$$

Thus, y_i is even and $a_i = 0$.

Q.E.D.

Property 6: For $i \neq 0$, $c_i=1$ and $a_i=0$ if i is a multiple of n_2.

Proof: By theorem 3.2.2, we have $x_i-y_i > 0$ if i is a multiple of n_2. Thus $z_i = x_i - y_i$.

By equation (3.2.4) and property 5, z_i is odd and y_i is even. Therefore, x_i is odd and $c_i = 1$.

Q.E.D.

Property 7: If i is not a multiple of n_2, then $a_i = c_i$ for $i > n_1$.

Proof: By theorem 3.2.2, we have $z_i = x_i - y_i > 0$. Since z_i is even if i is not a multiple of n_2, x_i and y_i must be either both even or both odd. Thus $a_i = c_i$.

Q.E.D.

Fast Algorithm for the Binary Form of A

The results presented above enable us to devise a simple algorithm for finding the binary representation form of A.

Let us denote the binary form of A by the n-tuple

$$A = (a_{n-1}\ a_{n-2}\ \cdots\ a_1\ a_0)$$

Then $a_{n-1} = a_{n-2} = \dots\dots = a_{n-n_1-n_2+1} = 0$.

Notice that the binary form of $C = 2^{n_1} \cdot A$ is just the binary form of A shifted n_1 places to the left, i.e. $c_{i+n_1} = a_i$. Let us denote C by

$$C = (c_{n-1}\ c_{n-2}\ \cdots\cdots\ c_1 c_0)$$

By properties 1 and 4,

$$(a_{n_1}, a_{n_1-1}, \dots, a_1, a_0) = (0, 1, 1, \dots, 1, 1)$$

Thus $(c_{2n_1}, c_{2n_1-1}, \ldots, c_{n_1+1}, c_{n_1}) = (0, 1, 1, \ldots, 1, 1)$ (3.2.12)

Now the values of the n_1-tuple $(a_{2n_1}, a_{2n_1-1}, \ldots, a_{n_1+1})$ can be determined from equation (3.2.12) and properties 6, 7. Next, the n-tuple $(c_{3n_1}, c_{3n_1-1}, \ldots, c_{2n_1+1})$ can be set equal to $(a_{2n_1}, \ldots, a_{n_1+1})$. Again, the n_1-tuple $(a_{3n_1}, \ldots, a_{2n_1+1})$ can be determined from $(c_{3n_1}, c_{3n_1-1}, \ldots, c_{2n_1+1})$ and properties 6 and 7. This process can be repeated until all the n-tuple $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ are determined. Therefore, we have the following algorithm:

```
1. Set $a_i = 1$ for $0 \le i \le n_1-1$, $a_{n_1} = 0$ and $j = 0$.

2. If $(j+1)\, n_1 < n-n_1-n_2$, set
   a. $c_{i+n_1} = a_i$ for $jn_1+1 \le i \le (j+1)\, n_1$
   b. $a_i = 0$ if $n_2$ divides $i$
      $a_i = c_i$ if $n_2$ does not divide $i$
      for $(j+1)\, n_1+1 \le i \le (j+2)\, n_1$
   c. $j = j+1$ and repeat step 2.

3. If $(j+1)\, n_1 \ge n-n_1-n_2$, set $a_i = 0$ for $n-n_1-n_2+1 \le i \le n-1$, Stop
```

The algorithm provides a fast way of finding the binary representation form of A, even for very large integers. For example, let $n_1=4$, $n_2=9$, $A=(2^{36}-1)/(2^4-1)(2^9-1)$. Follow the above algorithm, the binary form of A is readily obtained as:

$$C \qquad \overbrace{0100}^{6}\overbrace{0100}^{5}\overbrace{0110}^{4}\overbrace{0110}^{3}\overbrace{0111}^{2}\overbrace{0111}^{1}1$$

$$A \qquad 000000\underbrace{0000}_{7}\underbrace{0100}_{6}\underbrace{0100}_{5}\underbrace{0110}_{4}\underbrace{0110\,0111}_{3\quad 2}\underbrace{0111}_{1}1$$

The Complementary Symmetrical Property

We shall show that the binary form of A is complementary symmetrical. Specifically,

$$a_{n-(n_1+n_2)-i}+a_i = 1 \quad \text{for } 0<i<n-(n_1+n_2),$$

Let us consider the binary forms of A and D. Since,

$$D = A(2^{n_1}-1)=2^{(n_1-1)n_2}+2^{(n_1-2)n_2}+\dots+2^{n_2}+1$$

$$D = 1\,\underbrace{000\dots 0\,1}_{n_2\text{ bits}}\,\underbrace{00\dots 0\,1}_{n_2\text{ bits}}\,0\dots 01\,\underbrace{0\dots 0\,1}_{n_2-n_1}\,\underbrace{\bar{1}\,\bar{1}\dots\bar{1}}_{n_1} \tag{3.2.13}$$

where $\bar{x}$ denotes $-x$. Let D* be the integer obtained from D by reversing the order of the sequences in (3.2.13), i.e.,

$$D^* = \bar{1}\,\underbrace{\bar{1}\dots\bar{1}}_{n_1-1}\,\underbrace{10\dots01}_{n_2-n_1}\,\underbrace{00\dots01}_{n_2}\,0\dots01\,\underbrace{0\dots01}_{n_2} \tag{3.2.14}$$

Also let A* be the integer whose binary form is the binary form of A in reverse order. Then it is easy to check that $D^* = A^*(1-2^{n_1})$. Now, let us consider the integer D-D*. By (3.2.13) and (3.2.14),

$$D-D^* = 2\,\underbrace{11\dots1\bar{1}}_{n_1}\,00\dots01\,\underbrace{\bar{1}\bar{1}\dots\bar{2}}_{n_1} = 3\,\underbrace{00\dots0\bar{1}\bar{1}}_{n_1}\,\underbrace{00\dots0}_{n_2-n_1}\,\underbrace{00\dots0}_{(n_1-2)n_2}$$

$$= 3\,\underbrace{00\dots0\bar{3}}_{n_1}\,\underbrace{00\dots\dots\dots0}_{(n_1-1)n_2-n_1}$$

Thus, $D-D^* = 3\cdot 2^{(n_1-1)n_2}-3\cdot 2^{(n_1-1)n_2-n_1} = 3\cdot 2^{(n_1-1)n_2-n_1}(2^{n_1}-1)$.

Since $D = A(2^{n_1}-1)$ and $D^* = A^*(1-2^{n_1})$, we have

$$A + A^* = (D-D^*)/(2^{n_1}-1)$$
$$= 3\cdot 2^{n-n_1-n_2}$$
$$= 2^{n-n_1-n_2+1} \qquad (3.2.14)$$

Recall that

$$A = (1, a_{n-n_1-n_2-1}, \ldots, a_1, 1)$$

and

$$A^* = (1, a_1, \ldots., a_{n-n_1-n_2-1}, 1)$$

Starting from the lower order bits, by (3.2.14) we have,

$$a_1 + a_{n-n_1-n_2-1} + 1 \equiv 0 \mod 2$$

or in general

$$a_i + a_{n-n_1-n_2+i} + 1 \equiv 0 \mod 2 \quad \text{for } 0 < i < n-n_1-n_2$$

Therefore, a_i is the binary complement of $a_{n-n_1-n_2-i}$, and we have:

Property 8: The binary form of A is complementary symmetrical, i.e., $a_i + a_{n-n_1-n_2-i} = 1$ for $0 < i < n-n_1-n_2$.

Property 8 enables us to save half of the effort finding the binary form of A . The procedure for finding the binary form of A stops at the place when a_i, $i = \frac{1}{2}(n-n_1-n_2)$ is found. The high order bits are then obtained by reversing the order of the complement of the lower order bits. The complementary symmetrical property can be seen from the binary form of $A = (2^{36}-1)/(2^4-1)(2^9-1)$ in the previous example.

3.3 Construction of Multiple-Error-Correcting AN-Codes

Let A be the generator of a binary cyclic arithmetic (AN) code of length n. Then A is a positive integer that divides 2^n-1. Let $AB = 2^n-1$, for each N, $0 \leq N \leq B-1$, AN is a code word of the code. In this section, we shall consider the class of cyclic AN-codes whose generator A is of the same form as in Section 3.2:

$$A = (2^n-1)/(2^{n_1}-1)(2^{n_2}-1) \tag{3.3.1}$$

where n_1, n_2 are relatively prime integers and $n = n_1 n_2$. In the following discussion we shall assume that $n_2 > n_1$.

Any code word AN can be expressed in the following radix-2 or binary form:

$$\begin{aligned} AN &= a_{n-1}\, 2^{n-1} + a_{n-2}\, 2^{n-2} + \ldots + a_1\, 2^1 + a_0 \\ &= (a_{n-1}, a_{n-2}, \ldots, a_1, a_0) \end{aligned} \tag{3.3.2}$$

where $a_i = 0$ or 1 for $0 \leq i \leq n-1$. From the definition of arithmetic weight of an integer, one can easily derive the following result.

Lemma 3.3.1: If the binary sequence of AN in (3.3.2) can be divided into d disjoint subsequences (each subsequence consists of a certain number of consecutive digits), such that each subsequence contains at least one 1 and one 0, then the arithmetic weight of AN is at least equal to d.

Given A and B such that $AB = 2^n-1$, it can be shown [12] that the coefficients a_i's in (3.3.2) are given by

$$a_i \equiv (N\cdot 2^{n-i} \bmod B) \bmod 2 \qquad (3.3.3)$$

Thus, $a_i = 1$ if $N2^{n-i} \bmod B$ is odd and $a_i = 0$ if $N2^{n-i} \bmod B$ is even.

For the code considered in (3.3.1), $B = (2^{n_1}-1)(2^{n_2}-1)$. Thus,

$$2^{n_2}(2^{n_1}-1) \equiv 2^{n_1}-1 \bmod B$$

$$2^{in_2}(2^{n_1}-1) \equiv 2^{n_1}-1 \bmod B, \text{ for any positive integer } i \qquad (3.3.4)$$

Lemma 3.3.2: If $N \equiv 2^s \bmod (2^{n_2}-1)$ for $0 \le N \le B-1$ and $0 \le s < n_2$, then the arithmetic weight of the code word AN generated by A in (3.3.1) is at least equal to n_1.

Proof: Let a_i's be the binary coefficients of AN in (3.3.2). By (3.3.3)

$$a_{n-(in_2+n_1-s)} \equiv (N\, 2^{in_2+n_1-s} \bmod B) \bmod 2$$

and

$$a_{n-(in_2-s)} \equiv (N\cdot 2^{in_2-s} \bmod B) \bmod 2$$

where $0 \le i \le n_1-1$, and the subscripts i of a_i are taken to be i modulo n.

Let $x_i = (N2^{in_2+n_1-s} \bmod B)$ and $y_i = (N2^{in_2-s} \bmod B)$, then $0 < x_i, y_i < B$. Since $B = (2^{n_1}-1)(2^{n_2}-1)$ and $N \equiv 2^s \bmod (2^{n_2}-1)$,

$$x_i \equiv N2^{in_2+n_1-s} \bmod (2^{n_2}-1)$$

$$\equiv 2^{n_1} \bmod (2^{n_2}-1)$$

or

$$x_i = q_{1i}(2^{n_2}-1) + 2^{n_1}$$

Similarly
$$y_i \equiv N2^{in_2-s} \bmod (2^{n_2}-1)$$
$$\equiv 1 \bmod (2^{n_2}-1)$$
or
$$y_i = q_{2i}(2^{n_2}-1) + 1$$
Thus
$$x_i - y_i = (q_{1i} - q_{2i})(2^{n_2}-1) + (2^{n_1}-1) \qquad \text{(a)}$$
where $0 < q_{1i}, q_{2i} < 2^{n_1}-1$, for $0 < x_i, y_i < B = (2^{n_1}-1)(2^{n_2}-1)$. On the other hand,
$$N2^{in_2+n_1-s} - N2^{in_2-s} = N2^{in_2-s}(2^{n_1}-1)$$
$$\equiv N2^{-s}(2^{n_1}-1) \bmod B \qquad \text{by (3.3.4)}$$
$$\equiv 2^{n_1}-1 \bmod B$$
Thus
$$x_i - y_i \equiv 2^{n_1}-1 \bmod B$$
or
$$x_i - y_i = q(2^{n_1}-1)(2^{n_2}-1) + (2^{n_1}-1);\ q = 0, \text{ or } -1 \qquad \text{(b)}$$
From (a) and (b) we conclude that $q(2^{n_1}-1) = q_{1i} - q_{2i}$. But $|q_{1i} - q_{2i}| < 2^{n_1}-1$, q must be 0. Thus $x_i - y_i = 2^{n_1}-1$. This implies either x_i = odd and y_i = even or x_i = even and y_i = odd. In other words, for each i, one and only one of the pair of binary digits $a_{n-(in_2+n_1-s)}$ and $a_{n-(in_2-s)}$ is equal to 1 (or 0).

Now, the sequence of n digits of AN in (3.3.2) can be divided into n_1 subsequences, each subsequence contains n_2 consecutive digits. Therefore; the digits $a_{n-(in_2+n_1-s)}$ and $a_{n-(in_2-s)}$ are in the same subsequence for a fixed i.

Each subsequence, then, contains at least one 1 and one 0. By Lemma 1 the arithmetic weight of AN is at least n_1.

Q.E.D.

Lemma 3.3.3: If $N \equiv -2^s \bmod (2^{n_2}-1)$ for $0 \leq N \leq B-1$ and $0 \leq s \leq n_2$, the arithmetic weight of the code word AN generated by A in (3.3.1) is at least equal to n_1.

Proof: This lemma can be proved in a similar way as in Lemma 3.3.2.

Q.E.D.

Lemma 3.3.4: If $N \not\equiv 0 \bmod (2^{n_2}-1)$ and $N \not\equiv \pm 2^s \bmod (2^{n_2}-1)$ for $0 < N \leq B-1$, the arithmetic weight of AN generated by A in (3.3.1) is at least equal to n_1.

Proof: Let $V = AN$, $C = V(2^{n_1}-1)$ and $W(x)$ denote the arithmetic weight of x. Since the arithmetic weight of integers satisfies triangular inequality,

$$W(C) = W[V(2^{n_1}-1)] \leq W(V2^{n_1}) + W(V) = 2W(V)$$

$$W(C) \leq 2W(V)$$

Let $N_1 \equiv N \bmod (2^{n_2}-1)$, $C_1 \equiv C \bmod (2^n-1)$. Then, $C_1 = N_1 2^{(n_1-1)n_2} + N_1 2^{(n_1-2)n_2} + \ldots + N_1 2^{n_2} + N_1$. Thus the binary form of C_1 consists of n_1 replicas of the binary form of N_1. Since N_1 is nonzero and $N_1 \neq \pm 2^s$, the binary form of N_1 contains at least two 1's and two 0's. Therefore, the modular weight [36] of C_1, $W_m(C_1) \geq 2n_1$. Hence,

$$2W(V) \geq W(C) \geq W_m(C_1) \geq 2n_1$$

$$W(V) \geq n_1$$

Q.E.D.

Theorem 3.3.1: The minimum arithmetic distance of the code generated by A in (3.3.1) is equal to n_1 for $n_1 < n_2$.

Proof: Let us assume first that $N \equiv 0 \bmod (2^{n_2}-1)$, $0 \leq N \leq B-1$. Then $N = q(2^{n_2}-1)$, $0 < q \leq 2^{n_1}-1$, and

$$\begin{aligned} AN &= \frac{2^{n_1 n_2}-1}{2^{n_1}-1} \cdot q \\ &= q\,2^{(n_2-1)n_1} + q\,2^{(n_2-2)n_1} + \ldots + q\cdot 2^{n_1} + q \end{aligned}$$

By lemma 3.3.1, $W(AN) \geq n_2 > n_1$.

For other nonzero values of N, lemmas 3.3.2, 3.3.3, and 3.3.4 assure that the code word AN has an arithmetic weight of n_1 or greater. Therefore, the arithmetic distance of the code is at least equal to n_1. However, the weight of the code word $A(2^{n_2}-1)$ is exactly equal to n_1. Thus the arithmetic distance of the code is equal to n_1.

Q.E.D.

The code we constructed above is a multiple-error-correcting cyclic AN-code. The synthesis of this class of codes is straightforward, we merely choose two proper relatively prime integers n_1 and n_2, the integer $(2^{n_1 n_2}-1)/(2^{n_1}-1)(2^{n_2}-1)$ generates a code with minimum distance n_1.

3.4 Root-Distance Relation of p_1p_2-Codes

If the block length n is a product of two distinct primes p_1 and p_2, then the code is called a p_1p_2-code. A p_1p_2-code is an ideal in the ring of integers modulo $2^{p_1p_2}-1$.

Since the polynomial $x^{p_1p_2}-1$ can be factored into cyclotomic polynomials over the rational field as:

$$x^{p_1p_2}-1 = Q^1(x)Q^{p_1}(x)Q^{p_2}(x)Q^{p_1p_2}(x) \qquad (3.4.1)$$

We can substitute 2 for x in (3.4.1) and get

$$2^{p_1p_2}-1 = Q^1(2)Q^{p_1}(2)Q^{p_2}(2)Q^{p_1p_2}(2) \qquad (3.4.2)$$

where $Q^d(2)$'s are called the cyclotomic factors of 2^n-1.

We define a "cyclotomic generator" of arithmetic codes as:

Definition 3.4.1: A cyclotomic generator of an arithmetic code is a product of some cyclotomic factors of 2^n-1.

By this definition, the generator of the codes considered in Section 3.3 is a cyclotomic generator.

The only possible cyclotomic generators of a p_1p_2-code are:

$A_1 = Q^1(2)$, $A_2 = Q^{p_1}(2)$, $A_3 = Q^{p_2}(2)$, $A_4 = Q^{p_1}(2)Q^{p_2}(2)$, $A_5 = Q^{p_1}(2)Q^{p_1p_2}(2)$, $A_6 = Q^{p_2}(2)Q^{p_1p_2}(2)$, $A_7 = Q^{p_1p_2}(2)$ and $A_8 = 2^{p_1p_2}-1$. Both A_1 and A_8 are trivial generators.

Since $Q^{p_1}(2) = (2^{p_1}-1)/(2-1) = 2^{p_1}-1$, the minimum distance of the code generated by $A_2 = Q^{p_1}(2)$ is 2. Similarly, the minimum distance of the code generated by A_3 is also 2. These are the single error-detecting codes. For $A = A_4 = (2^{p_1}-1)(2^{p_2}-1)$, the minimum distance of the code is given by the following theorem:

Theorem 3.4.1: The minimum distance of p_1p_2-code generated by A_4 is

(a) 3 if $p_1p_2 = 6$;

(b) 4 if $p_1p_2 \neq 6$.

Proof: With no loss of generality we may assume $p_2 > p_1$, hence $p_2 \geq 3$. Furthermore, $p_1p_2 = 6$ if and only if $p_1 = 2$ and $p_2 = 3$. It is obvious that A_4 cannot divide 2^i hence there are no codewords of weight 1. Codewords of weight 2 will be of the form $2^k(2^m \pm 1)$, where $m < p_1p_2$. For $A_4N = 2^k(2^m-1)$ we have $(2^{p_1}-1)|(2^m-1)$, hence $p_1|m$. Similarly, $p_2|m$ hence $p_1p_2|m$. But p_1p_2 is greater than m, a contradiction. If $A_4N = 2^k(2^m+1)$, then we may write $m = q_1p_1 + r_1$, $r_1<p_1$; $(2^{p_1}-1)|(2^m+1)$ then implies $2^{p_1}-1|2^{r_1}+1$, an impossibility. Hence all nonzero codewords of the code generated by $A_4 = 2^{p_1+p_2}-2^{p_2}-2^{p_1}+1$ are at least of weight 3.

For the case $p_1p_2 = 6$, $p_1 = 2$ and $p_2 = 3$. Hence $A_4 = 2^5-2^3-2^2+1 = 2^4+2^2+1$. Consequently A_4 generates a code of minimum distance 3.

For the case $p_1p_2 \neq 6$, it follows that $p_2 \geq p_1 + 2$, hence $A_4 = 2^{p_1+p_2}-2^{p_2}-2^{p_1}+1$ is of weight 4. The codewords of weight 3 will take the form $2^k(2^{m_1}\pm 2^{m_2}\pm 1)$, where $m_2 \leq m_1-2$. Writing:

$m_1 = q_{11}p_1+r_{11}$	where $0 \leq r_{11} < p_1$
$m_1 = q_{12}p_2+r_{12}$	where $0 \leq r_{12} < p_2$
$m_2 = q_{21}p_1+r_{21}$	where $0 \leq r_{21} < p_1$
$m_2 = q_{22}p_2+r_{22}$	where $0 \leq r_{22} < p_2$

we may deduce that

$$2^{p_i}-1 \,|2^{m_1}\pm 2^{m_2} \pm 1 \text{ implies } 2^{p_1}-1|2^{r_{11}}\pm 2^{r_{21}} \pm 1$$

and

$$2^{p_2}-1 \mid 2^{m_1} \pm 2^{m_2} \pm 1 \text{ implies } 2^{p_2}-1 \mid 2^{r_{12}} \pm 2^{r_{22}} \pm 1$$

The only possibilities are: $A_4N = 2^k(2^{m_1}-2^{m_2}-1)$ where $r_{21} = r_{22} = 0$; $r_{12} = r_{11} = 1$ and $A_4N' = 2^k(2^{m_1}+2^{m_2}-1)$ where $r_{21} = r_{11} = p_1-1$; $r_{22} = r_{12} = p_2-1$.

When $A_4N = 2^k(2^{m_1}-2^{m_2}-1)$, we have:

$$A_4N = 2^k(2^{t\cdot p_1p_2+1} - 2^{s\cdot p_1p_2} -1)$$

This is clearly impossible as $s \geq 1$ and $m_2 \leq m_1-2$ would imply $t \geq 2$. When $A_4N' = 2^k(2^{m_1}+2^{m_2}-1)$, we have:

$$A_4N' = 2^k(2^{t\cdot p_1p_2-1} + 2^{s\cdot p_1p_2-1} -1)$$

Again $s \geq 1$ and $m_2 \leq m_1-2$ would imply $t \geq 2$, a contradiction. This completes the proof of theorem 3.4.1.

Q.E.D.

For $A = A_5 = Q^{p_1}(2)Q^{p_1p_2}(2)$, we have:

$$A_5 = (2^{p_1p_2}-1)/Q^{p_2}(2) = (2^{p_1p_2}-1)/(2^{p_2}-1)$$
$$= 2^{(p_1-1)p_2} + 2^{(p_1-2)p_2} + \ldots + 2^{p_2} + 1$$

As indicated by Erosh [23], the minimum distance of a code generated by an integer of the form $(2^{k_1k_2}-1)/(2^{k_1}-1)$ is k_2. Therefore, the p_1p_2-code generated by A_5 is of distance p_1. Similarly, the code generated by A_6 is of distance p_2.

For $A = A_7 = Q^{p_1p_2}(2) = (2^{p_1p_2}-1)/(2^{p_1}-1)(2^{p_2}-1)$, the code is a special case of the code mentioned in Section 3.3. Since $(p_1,p_2) = 1$ and $p_1 < p_2$, by theorem 3.3.1, the minimum distance is p_1.

From the above results, some interesting properties about the root-distance relation of arithmetic codes can be brought to light.

Let us consider the polynomial x^n-1 over the rational field. If u is a primitive n-th root of unity [48], then all roots of x^n-1 are powers of u and we have $x^n-1 = \prod_{i=1}^{n}(x-u^i)$. Also, x^n-1 can be factorized into cyclotomic polynominals over the rational field as:

$$x^n-1 = \prod_{d/n} Q^d(x) \tag{3.4.3}$$

Hence, the roots contained in each $Q^d(x)$ are powers of u. Due to the property of cyclotomic polynomials [6], the roots in $Q^d(x)$ are the u^i's where i's satisfy: $(n,i) = n/d$.

Substituting 2 for x in eq. (3.4.3), we get $2^n-1 = \prod_{d/n} Q^d(2)$. Let A be a cyclotomic generator, then A equals a product of some cyclotomic factors, say:

$$A = Q^{d_1}(2)Q^{d_2}(2)^{d_3}(2)\dots Q^{d_s}(2)$$

We define the corresponding polynomial of the generator A as:

$$A(x) = Q^{d_1}(x)Q^{d_2}(x)\dots Q^{d_s}(x)$$

Then the roots of the integral generator A can be defined as follows:

Definition 3.4.2: The roots of a cyclotomic generator A are defined as the roots of its corresponding polynomial A(x).

This definition of the roots of A was first proposed by Chien and Hong [15]. Since all the roots of A are powers of u, the number of consecutive roots in A is defined as the largest number of consecutive powers of u contained in A(x).

For the p_1p_2-code, the root-distance relation can be easily obtained. The cyclotomic factors of $2^{p_1p_2}-1$ are $Q^1(2)$, $Q^{p_1}(2)$, $Q^{p_2}(2)$ and $Q^{p_1p_2}(2)$. If u is a p_1p_2-th root of unity, then with a little calculation, we can summarize the root-distance relation as follows:

Generator	No. of consecutive roots	Minimum distance
$A_2 = Q^{p_1}(2)$	1	2
$A_3 = Q^{p_2}(2)$	1	2
$A_4 = Q^{p_1}(2)Q^{p_2}(2)$		
*if $p_1 = 2$, $p_2 \neq 3$	3	4
*if p_1,p_2 are odd	2	4
*if $p_1p_2 = 6$	3	3
$A_5 = Q^{p_1}(2)Q^{p_1p_2}(2)$	p_1-1	p_1
$A_6 = Q^{p_2}(2)Q^{p_1p_2}(2)$	p_2-1	p_2
$A_7 = Q^{p_1p_2}(2)$	p_1-1	p_1

From the above tabulated results, we see that for all possible cyclotomic generators of a p_1p_2-code (except the case $p_1p_2 = 6$), the minimum distance of the code is at least one greater than the number of consecutive roots contained in the generator--the same root-distance relations as the BCH theorem for polynomial codes. As for the case of $p_1p_2 = 6$, we have the following argument.

Since $2^6-1 = Q^1(2)Q^2(2)Q^3(2)Q^6(2)$, where $Q^1(2) = 1$, $Q^2(2) = 3$, $Q^3(2) = 7$ and $Q^6(2) = 3$, the generator A_4 is 21. The code generated by A_4 is of distance 3, which equals the number of consecutive roots contained in $Q^2(x)Q^3(x)$. However, the generator $A_4 = 3.7$ can also be considered as the product of $Q^3(2)$ and $Q^6(2)$ since $Q^6(2) = Q^2(2) = 3$. The number of consecutive roots in $Q^3(x)Q^6(x)$ is only 2, which is indeed one less than the minimum distance of the code. The situation that a cyclotomic generator can be expressed as different products of cyclotomic factors can only happen when $p_1p_2 = 6$, since Dickson [21] has shown that $Q^i(2) \neq Q^j(2)$ for all $i \neq j$, except $i = 2$ and $j = 6$.

The analogies between cyclic parity-check codes and cyclic AN-codes have been investigated by many authors. The root-distance relationship discussed above indicates another analogy. It is our hope that the closeness of all these analogies would lead us to the discovery of a class of cyclic AN-codes which is analogous to the large and powerful Bose-Chaudhuri-Hocquenghem codes. Then, for all practical purposes, the synthesis of multiple-error-correcting code can be achieved through the root conditions of the code generator.

3.5 Bounds on Error Control Capability of Cyclic AN-Codes

Since the generator A of cyclic AN-code of length n is a proper divisor of 2^n-1, we can write $2^n-1 = AB$, with $B > 1$. As briefly mentioned in the introductory section of this chapter, the well-known "middle-third" region of B is defined as the set of consecutive integers x, with $B \leq 3x < 2B$. If we denote the "middle-third" region of B by M_B, then there are $\lceil (B+1)/3 \rceil$ integers in M_B, where $\lceil x \rceil$ denotes the largest integral value of x.

It has been shown [12,17] that the arithmetic weight of any codeword is related to the integers in the region M_B. This result can be briefly summarized in the following:

Theorem: The arithmetic weight of a codeword AN is given by the number of residues $X_i \in M_B$, where, $X_i = (N \cdot 2^i) \bmod B$, for $i = 0,1,2,\ldots, n-1$.

From the above theorem, it is seen that the minimum distance of a code can be evaluated by counting the number of residues X_i in the region M_B.

To calculate the distance, we first divide the codewords into several disjoint orbits [17], for each orbit we count the number of residues in M_B, and get the arithmetic weight of each orbit. Then the distance equals the minimum weight. Clearly, this evaluation will be quite complex for large, and composite B's. Also, since we do not have a closed-form formula for the distance of a code, the above result does not suggest a synthesis procedure for finding any multiple-error correcting arithmetic codes, we can only calculate, through a computer-programmed procedure, the error-correcting capability of a given code. Therefore, the result is a theoretical analysis of the structure of the arithmetic codes rather than a practical formulation.

In this section, we shall apply some number theoretic concepts to obtain the upper bound and the lower bound on the error-control capability of any cyclic AN-codes. These bounds suggest the criteria for us to choose proper parameters in constructing an AN-code.

3.5.1 Lower Bound on the Distance

Let the B integers $\{0, 1, 2, \ldots, B-1\}$ be divided into three regions L_B, M_B, and U_B, where:

$$L_B = \text{"lower third" region} = \{x \mid 0 \leq x < \lceil \tfrac{B}{3} \rceil\}$$

$$M_B = \text{"middle third" region} = \{x \mid \lceil \tfrac{B}{3} \rceil \leq x < \lceil \tfrac{2B}{3} \rceil\}$$

$$U_B = \text{"upper third" region} = \{x \mid \lceil \tfrac{2}{3}B \rceil \leq x < B\}$$

Let us define the following:

Definition 3.5.1: For any integer N, $0 < N < B$, the smallest, non-negative integer i such that $(2^i \cdot N) \bmod B \in M_B$ is defined as the <u>index</u> of N, and the integer N is said to have index i.

Suppose that AN is a codeword, then $0 \leq N < B$. If $N \in M_B$, then the smallest power of 2 that yields $2^i N \bmod B \in M_B$ is clearly zero, hence N has index 0. For $N \in L_B$ or $N \in U_B$, the index is certainly not zero. Let us first consider the integers in L_B. Except the integer 0, such integer in L_B has a nonzero index. Denoting the index of $x \in L_B$ by i_x, we have the following:

Lemma 3.5.1: The index $i_1 = \lceil \log_2 \frac{B}{3} \rceil$.

Proof: Since $2^{i_1} \cdot 1 \bmod B \in M_B$ and $2^{i_1 - 1} \cdot 1 \bmod B \in L_B$, we have:

$$2^{i_1 - 1} < \lceil \tfrac{B}{3} \rceil \leq 2^{i_1}$$

or:

$$i_1 - 1 < \log_2 \lceil \tfrac{B}{3} \rceil \leq i_1$$

thus:

$$i_1 = \lceil \log_2 \tfrac{B}{3} \rceil \ .$$

Q.E.D.

Lemma 3.5.2: For $x;y \epsilon L_B$, $i_x \geq i_y$ if $2x > y > x$. Furthermore, if $y \geq 2x$, then $i_x > i_y$.

Proof: Since $x;y \epsilon L_B$, by the definition of index, the integers $2^{i_x} \cdot x$ and $2^{i_y} \cdot y$ are less than B, thus we have:

$$2^{i_x} \cdot x \epsilon M_B \text{ and } 2^{i_y} \cdot y \epsilon M_B .$$

Also, by the definitions of L_B, M_B and U_B, we have:

$$\lceil \tfrac{1}{3}B \rceil \leq 2^{i_x} \cdot x = 2^{i_x - 1} \cdot (2x) < \lceil \tfrac{2}{3}B \rceil \qquad (3.5.1)$$

$$\lceil \tfrac{1}{3}B \rceil \leq 2^{i_y} \cdot y \lceil \tfrac{2}{3}B \rceil \qquad (3.5.2)$$

Hence, if $2x > y > x$, we have:

$$2^{i_x} \cdot y > 2^{i_x} \cdot x = 2^{i_x - 1}(2x) > 2^{i_x - 1} \cdot y \qquad (3.5.3)$$

From (3.5.1), (3.5.2) and (3.5.3), it is seen that the integer $2^{i_x - 1} \cdot y$ is either in L_B or in M_B.

If $2^{i_x - 1} \cdot y \epsilon M_B$, then $i_y = i_x - 1$; while if $2^{i_x - 1} \cdot y \epsilon L_B$, then $2^{i_x} \cdot y \epsilon M_B$ and $i_x = i_y$, hence $i_x \geq i_y$.

If $y \geq 2x$, then we have:

$$2^{i_y} \cdot y \geq 2^{i_y + 1} \cdot x$$

hence either $2^{i_y + 1} \cdot x \epsilon M_B$ or $2^{i_y + 1} \cdot x \epsilon L_B$.

If $2^{i_y + 1} \cdot x \epsilon M_B$, then $i_y + 1 = i_x$, while if $2^{i_y + 1} \cdot x \epsilon L_B$ then $i_y + 2 \leq i_x$; hence $i_x > i_y$.

Q.E.D.

Lemma 3.5.3: The residue 2^{n-1} mod B is in M_B, if $B \neq 1$.

Proof: Let $2^{n-1} \equiv k \pmod{B}$, then

$$2^{n-1} = m \cdot B + k \qquad \text{for some } m.$$

Since

$$2^n - 1 = AB, \qquad \text{both A and B are odd, and}$$

$$2^n = AB + 1 \tag{3.5.4}$$

hence,

$$2^{n-1} = \frac{AB+1}{2}$$

$$= \frac{B+1}{2} + \frac{(A-1)B}{2} \equiv \frac{1}{2}(B+1) \mod B \tag{3.5.5}$$

But

$$\frac{B+1}{2} - \frac{B}{3} = \frac{1}{6}[3B+3-2B] = \frac{1}{6}[B+3],$$

and

$$\frac{2B}{3} - \frac{B+1}{2} = \frac{1}{6}[4B-3B-3] = \frac{1}{6}[B-3]$$

Hence, for $B \geq 3$, $2^{n-1} \mod B = \frac{1}{2}(B+1) \in M_B$.

Q.E.D.

Since the integers in U_B are congruent to the negatives of the integers in L_B, we can use the same argument as in lemmas 3.5.1 and 3.5.2 to obtain:

Lemma 3.5.4: For $x \in U_B$, the index i_x equals to index i_{B-x}, where $B-x \in L_B$.

Therefore, in the range $\lceil 0, B-1 \rceil$, the integers 1 and B-1 have the largest index $\lceil \log_2 \frac{B}{3} \rceil$. With the above lemmas, we now derive a lower bound on the minimum distance of the cyclic AN-codes.

Let us consider an orbit of n residues:

$$(N \bmod B), (2N \bmod B), (2^2N \bmod B), \ldots\ldots, (2^{n-1}N \bmod B) \quad (3.5.6)$$

Suppose that the residue $(2^kN \bmod B) \varepsilon M_B$. By lemma 3.5.1, there exists another residue $(2^{k+m}N \bmod B) \varepsilon M_B$, with $1 < m \leq i_1+1$, where $i_1 = \lceil \log_2 \frac{B}{3} \rceil$. The rest of the residues are:

$$(2^{k+m+1}N \bmod B), (2^{k+m+2}N \bmod B), \ldots, (2^{n-1}N \bmod B), (N \bmod B),$$
$$(2^{k-1}N \bmod B) \quad (3.5.7)$$

By lemmas 3.5.2 and 3.5.3, we see that there exists at least one residue which is in M_B for every i_1+1 consecutive residues in the set (3.5.7). Since the arithmetic weight of the code word AN equals the number of residues of the orbit (3.5.6) which are in M_B, we have proved the following theorem:

Theorem 3.5.1: The minimum distance, D_m, of a cyclic AN-code of length n is at least as large as

$$D_m \geq \left\lceil \frac{n-1-(i_1+1)}{i_1+1} \right\rceil + 2 \quad (3.5.8)$$

The tightness of the bound in theorem 3.5.1 can be improved if we consider the following:

If B=3, then $i_1=0$. In this case there are only two codewords in the code, the minimum distance of the code is trivially seen to be n/2. In the following, we consider the case that $B \neq 3$:

Lemma 3.5.4: Let the exponent of 2 modulo B be denoted by e(B). If e(B)=n, and if 1 and B-1 are not contained in the same residue set $\{2^i \bmod B;\ 0 \leq i < n\}$, then

$$D_m \geq \left\lceil \frac{n-i_1-2}{i_1} \right\rceil + 2$$

Proof: Since 1 and B-1 are in distinct residue sets, for every i_1 consecutive residues in the residue set (2) there must exist at least one residue in M_B.

Lemma 3.5.5: If $e(B)=n$ and $2^k \equiv -1 \bmod B$ for some $0<k<n$, then $n=2k$.

Proof: We have

$$2^n = 2^{e(B)} \equiv 1 \quad (\text{mod } B)$$
$$2^k \equiv -1 \quad (\text{mod } B)$$
$$2^{2k} \equiv 1 \quad (\text{mod } B)$$

By definition of e(B), n is a divisor of 2k. If n is odd, then $n|k$; this is impossible since $k<n$. If n is even, then $(n/2)|k$ or $k=t\cdot(n/2)$ for some integer t. Since $0<k<n$, $t=1$. Therefore, $n=2k$.

Lemma 3.5.5 states that if 1 and B-1 are contained in the same residue set $\{2^i \bmod B;\ 0\leq i<n\}$, the length of the code must be even. Any residue set $\{N\cdot 2^i \bmod B;\ 0\leq i<n\}$ can then be divided into two subsets,

$$S_1 = \{N \bmod B,\ N\cdot 2 \bmod B,\ \ldots\ldots\ldots,\ N\cdot 2^{(n/2)-1} \bmod B\}$$
$$S_2 = \{N\cdot 2^{(n/2)} \bmod B\ \ldots\ldots\ldots\ldots,\ N\cdot 2^{n-1} \bmod B\}$$

The elements in S_2 are the additive inverses in the ring of integers modulo B of the elements in S_1. These two sets have the same number of residues that are in the region M_B. Therefore, we have

Lemma 3.5.6: If $e(B)=n=$even, and $2^{(n/2)} \equiv -1 \bmod B$, then

$$D_m \geq \left\lceil \frac{(n/2)-i_1-2}{i_1} + 2 \right\rceil \times 2$$

It should be mentioned that $(n/2)-i_1-2$ is always non-negative as long as the codes have error correcting capability. This is true because that $(n/2)-i_1 \geq 1$ and the equality holds only when the code is of distance 2.

If the exponent of 2 modulo B is not equal to n, then e(B) is a factor of n,i,e., $n=e(B)\cdot q$. In this case any residue set $\{N\cdot 2^i \bmod B,\ 0\leq i<n\}$ can be divided into q identical subsets.

Lemma 3.5.7: If $n=e(B)\cdot q$, 1 and B-1 are not contained in the same residue set $\{2^i \bmod B,\ 0\leq i<e(B)\}$, then

$$\text{(a)}\quad D_m \geq \left\lceil \frac{e(B)-i_1-2}{i_1} +2\right\rceil xq \qquad \text{for } e(B)-i_1>1$$

$$\text{(b)}\quad D_m = q \qquad \text{for } e(B)-i_1=1$$

Proof:

(a) Since $e(B)-i_1\geq 1$, we have $e(B)-i_1-2\geq -1$. If $e(B)-i_1>1$, then $e(B)-i_1-2\geq 0$. From lemma 3.5.4 and the fact that any residue set contains q identical subsets, part (a) of the lemma follows clearly.

(b) If $e(B)-i_1=1$, then $e(B)-i_1-2=-1$. This implies that among the e(B) residues 2N mod B, 2^2N mod B,..., $2^{e(B)-1}N$ mod B, there is only one residue, namely, $2^{e(B)-1}N$ mod B, which is in M_B. The NAF of the generator A is seen to be

$$\overbrace{00\ldots 01}^{e(B)\text{ bits}},\overbrace{00\ldots 01}^{e(B)\text{ bits}},\ldots\ldots,\ldots\ldots,\overbrace{00\ldots 01}^{e(B)\text{ bits}}$$

and the minimum distance of the code is q. Q.E.D.

If $n=e(B)\cdot q$, and if 1, B-1 are both contained in the same residue set $\{2^i \bmod B,\ 0\leq i<e(B)\}$, then by a similar argument as in lemmas 3.5.5 and 3.5.6 we have:

Lemma 3.5.8: If $n=e(B)\cdot q$ and $2^{e(B)/2}\equiv -1 \bmod B$, then

$$D_m \geq \left\lceil \frac{e(B)/2-i_1-2}{i_1} +2\right\rceil \cdot 2\cdot q$$

Again, by using the same argument as in lemma 3.5.7, we have

Lemma 3.5.9: If $n=e(B)\cdot q$, $2^{e(B)/2}\equiv -1 \bmod B$ and $e(B)/2-i_1=1$, then

$$D_m = 2q$$

In light of the above lemmas, the determination of the lower bound on the minimum distance of any cyclic AN-code can be summarized as follows:

Theorem 3.5.2: For a cyclic AN-code of length n,

(a) $n=e(B)$

$$D_m \geq \{\lceil((n/2)-i_1-2)/i_1\rceil+2\}\cdot 2$$

if $n=$even and $2^{(n/2)}\equiv -1 \bmod B$

$$D_m \geq \lceil(n-i_1-2)/i_1\rceil+2$$

otherwise

(b) $B=3$

$$D_m = n/2$$

(c) $n=e(B)\cdot q$

$$D_m \geq \{\lceil(e(B)/2-i_1-2)/i_1\rceil+2\}\cdot 2q$$

;if $e(B)=$even and $2^{e(B)/2}\equiv -1 \bmod B$

** $D_m = 2q$;if $e(B)=$even, $2^{e(B)/2}\equiv -1 \bmod B$, and $e(B)/2-i_1=1$

$$D_m \geq \{\lceil(e(B)-i_1-2)/i_1\rceil+2\}\cdot q$$

;if $2^{e(B)/2}\not\equiv -1 \bmod B$

** $D_m = q$;if $2^{e(B)/2}\not\equiv -1 \bmod B$ and $e(B)-i_1=1$

For any given cyclic AN-code, the above theorem can be used to determine the lower bound on the error correcting capability of the code. The calculation is very simple. We present in the following the lower bounds (calculated by the CDC 1604 computer) of all possible cyclic AN-codes of length up to 36:

n(code length)	B	e(B)	i_1	D_m(lower bound)	D_m(actual)
8	3	2	0	4	4
8	5	4	1	4	4
9	7	3	2	3	3
10	11	10	2	4	4
10	3	2	0	5	5
12	7·13	12	5	3	3
12	3·3·13	12	6	3	3
12	3·3·5	12	4	3	3
12	5·7	12	4	3	4
12	13	12	3	4	4
12	3·3	6	2	4	4
14	43	14	4	4	4
14	3	2	0	7	7
15	7·31	15	7	3	3
15	151	15	6	3	3
15	7	3	2	5	5
16	5·17	8	5	4	4
16	51	8	5	4	4
16	3·5	4	3	4	4
16	5	4	1	8	8
16	3	2	0	8	8
18	3·3·7	6	5	3	3
18	3·3·73	18	8	3	4
18	7	3	2	6	6
18	3·3	6	2	6	6
18	7·3·19	18	8	3	4
18	73	9	5	4	4
20	3·11·5·41	20	12	3	3
20	3·31·5·41	20	14	3	3
20	5·3·31	20	8	3	4
20	5·3·11	20	6	4	4
20	5·11·31	20	10	3	4
20	11	10	2	8	8
20	3·5	4	3	5	5
20	5·11	20	5	4	4
20	5	4	1	10	10
20	3	2	0	10	10
21	7·127	21	9	3	3
21	7·337	21	10	3	4
21	7	3	2	7	7
22	683	22	8	4	4
22	3	2	0	11	11
24	3·3·5·7·17	24	11	3	3
24	3·3·5·7·241	24	15	3	3
24	5·7·31·241	24	16	3	3
24	3·3·5·13·241	24	12	3	3
24	3·3·13·7·17	24	12	3	4

n(code length)	B	e(B)	i_1	D_m(lower bound)	D_m(actual)
24	3·3·5·7	12	7	4	4
24	3·3·5·13	12	8	4	4
24	3·3·13·17	24	10	3	4
24	7·13·17	24	9	3	5
24	5·7·17	24	8	3	4
24	3·3·13	12	6	4	4
24	3·3·5	12	4	6	6
24	5·7	12	4	6	8
24	13	12	3	8	8
24	9	6	2	8	8
24	5	4	1	12	12
24	3	2	0	12	12
25	31	5	4	5	5
26	2731	26	10	4	4
27	73	9	5	6	6
27	7	3	2	9	9
28	3·5·43	28	8	4	4
28	3·5·127	28	10	4	4
28	3·29·113	28	12	3	4
28	3·127·29·113	28	18	3	3
28	3·43·29·113	28	17	3	3
28	3·29·113	28	12	3	4
28	3·5	4	3	7	7
28	5	4	1	14	14
28	3	2	0	14	14
30	3·3·11·7·31	30	13	3	3
30	7·151·11·331	30	21	3	3
30	9·31·151·331	30	22	3	3
30	9·11·31·151	30	18	3	3
30	7·11·31·331	30	19	3	4
30	7·9·31·331	30	18	3	4
30	7·9·11·151	30	14	4	4
30	9·31·331	30	12	3	4
30	9·31·151	30	14	3	4
30	9·11	30	6	6	6
30	9·7·11	30	8	4	5
30	7·9·31	30	10	4	5
30	9·31	30	7	5	6
30	9·11	30	6	6	6
30	7·11·31	30	10	4	5
30	3·3	6	2	10	10
30	11	10	2	12	12
30	7	3	2	10	10
30	3	2	0	15	15
32	3·5·17	8	7	4	4
32	3·5·257	16	11	4	4
32	3·17·257	16	13	4	4
32	5·17·257	16	13	4	4

n(code length)	B	e(B)	i_1	D_m(lower bound)	D_m(actual)
32	3·5	4	3	8	8
32	3·17	8	5	8	8
32	5·17	8	5	8	8
32	3	2	0	16	16
32	5	4	1	16	16
33	7·23·89	33	13	3	3
33	599479	33	18	3	3
33	7	3	2	11	11
34	43691	34	14	4	4
34	3	2	0	17	17
35	71·122921	35	22	3	4
35	127	7	6	5	5
35	31	5	4	7	7
35	31·127	35	11	5	5
36	5·7·3·3·13·73	36	17	3	3
36	9·7·5·73·37·109	36	26	3	3
36	5·7·27·19·37·109	36	25	3	3
36	5·7·27·19·13	36	17	3	3
36	9·7·73·13·109·37	36	27	3	3
36	3·3·7·5·73	36	13	3	4
36	9·5·7·37·109	36	19	3	4
36	9·7·13·37·109	36	20	3	4
36	9·7·13·73	36	14	3	4
36	3·5·7·13·73·19	36	19	3	4
36	3·3·3·5·7	36	9	4	4
36	9·7·19·13	36	14	3	4
36	9·5·13·19·73	36	20	3	4
36	3·3·5·7	12	7	6	6
36	3·3·7·13	12	9	6	6
36	5·9·13·73	36	15	6	6
36	3·7·13·19	36	11	4	5
36	3·3·5·73	36	11	4	8
36	3·3·73	18	8	6	8
36	3·5·7·19	36	10	6	8
36	7·13	12	5	9	9
36	9·13	12	6	9	9
36	3·3·7	6	5	6	6
36	9·73	18	8	6	8
36	73	9	5	8	8
36	3·7·19	18	8	6	8
36	5·3·3	12	4	9	9
36	13	12	3	12	12
36	3·3	6	2	12	12
36	7	3	2	12	12
36	5	4	1	18	18
36	3	2	0	18	18

From the above tabulated results, it is seen that the lower bound developed in this section is very tight. This lower bound also suggests a criteria for one to choose proper B's in constructing any cyclic AN-code. Suppose that we want to construct a cyclic AN-code with a specified error correcting capability. First we let the lower bound on the code distance to be D_1, if the code length n is odd and the code is so constructed that $e(B)=n$, (other cases can be considered in a similar manner), then by theorem 3.5.2, we have

$$D_1 = \left\lceil \frac{n-i_1-2}{i_1} \right\rceil +2 = \left\lceil \frac{n-2}{i_1} \right\rceil +1$$

$$D_1-1 = (n-2)/i_1-u \qquad \text{where } 0\leq u<1$$

$$D_1-1 \leq (n-2)/i_1$$

$$i_1 \leq (n-2)/(D_1-1)$$

$$i_1 \leq \lceil (n-2)/(D_1-1) \rceil$$

$$\log_2(B/3)-u' \leq \lceil (n-2)/(D_1-1) \rceil \qquad \text{where } 0\leq u'<1$$

the smallest integer that is greater than or equal to $\log_2(B/3)$ is thus,

$$\log_2(B/3) \leq \lceil (n-2)/(D_1-1) \rceil +1$$

hence,

$$B/3 \leq 2^{\lceil (n-2)/(D_1-1) \rceil +1}$$

or,

$$B \leq 3\cdot 2^{\lceil (n-2)/(D_1-1) \rceil +1} \triangleq B_1$$

From the above expression, we see that for a fixed code length n, the number of code words should be smaller than B_1 so that the code constructed guarantees to possess a minimum distance at least as great as D_1. Let us consider the following example:

Suppose that we want to construct a single error correcting cyclic AN-code of length 15. The code must have distance at least 3, and B_1 can be calculated as

$$B_1 = 3 \cdot 2^{\lceil (n-2)/(D_1-1) \rceil} + 1$$
$$= 3 \cdot 2^{\lceil (15-2)/(3-1) \rceil} + 1$$
$$= 3 \cdot 2^7 = 384$$

We choose B to be smaller than 384, (e.g., $B=31 \cdot 7=217$) and $e(B)=15$, the code is of minimum distance at least 3.

By using the lower bound on the minimum distance derived in theorem 3.5.2, we can synthesize any cyclic AN-code for the correction of a specified number of errors. The following codes are obtained:

n	=	length of the code
B	=	number of code words
k	=	number of information bits
D_{ml}	=	lower bound of the minimum distance
D_{ma}	=	actual minimum distance
r	=	rate of the code

n	B	k	D_{ml}	D_{ma}	r
42	5419	13	6	6	0.31
42	16257	14	6	6	0.33
44	10565	14	6	6	0.32
48	24929	15	6	6	0.314
50	8283	14	8	10	0.28
50	44561	16	8	8	0.32
52	85489	17	8	8	0.33
52	253241	18	5	6	0.346
52	1266205	21	6	6	0.404
54	784899	20	6	8	0.37
54	87211	17	6	6	0.315
54	261633	18	8	8	0.33
58	3033169	22	5	6	0.38

n	B	k	D_{ml}	D_{ma}	r
60	429325	19	6	6	0.315
60	3520465	22	6	6	0.365
60	162565	18	8	8	0.30
60	402905	19	6	6	0.317
60	812825	20	6	6	0.33
60	270805	19	6	8	0.317
64	6700417	23	6	6	0.36
66	1397419	21	6	6	0.318
66	4192257	22	6	6	0.33
66	411849	19	6	6	0.288
66	62571	16	8	10	0.244
66	45761	16	10	12	0.244
68	652805	20	8	10	0.295
68	130561	17	10	12	0.25
68	3505429	22	8	8	0.325
68	25080101	25	5	6	0.368
70	12083	14	12	14	0.20
70	36249	16	8	10	0.228
70	398739	19	10	12	0.271
70	3705353	22	6	8	0.315
70	11116059	24	5	6	0.342
72	16773121	24	6	8	0.33
72	1774001	21	6	8	0.292
72	9335617	24	6	6	0.33
72	104323	17	10	14	0.237
74	25781083	25	6	8	0.338
74	77343249	27	6	8	0.365

3.5.2 Upper Bound on the Distance

To find an upper bound on the distance of a cyclic AN-code, we first consider the congruence:

$$2^i N \equiv x \mod B \tag{3.5.9}$$

where $0 \leq i \leq n-1$ is given, and x is any integer. We want to solve for the integer N.

Since B is an odd integer, 2^i is relatively prime with respect to B for all i's. By Euclidean lemma, we have:

$$s \cdot 2^i + t \cdot B = 1 \quad \text{for some integers s and t} \tag{3.5.10}$$

Hence, for any integer x, the above congreuence has a unique residue class solution for N modulo B. If the value of N is restricted to be: $0 \leq N \leq B-1$, then the solution of (3.5.9) is the actual value of N.

For a code word AN, $0 \leq N \leq B-1$, the residue $2^i N$ mod B indicates the nonzero terms in its NAF. If we take a fixed integer i, and set x to be an integer in M_B, then the solution of N in the above congruence says that the code word AN has a nonzero term in its NAF at the position 2^{n-i}. Also, for a fixed i, the solution N of the congruence (3.5.9) are all distinct for different x's in M_B; this is because

$$2^i N_1 \equiv x \mod B$$

$$2^i N_2 \equiv y \mod B$$

would imply that $x \equiv y$ mod B and $0 \leq x,y \leq B-1$ then implies $x = y$.

These set of N's obtained by choosing a fixed i and setting x to be all the integers in M_B are simply the codewords that have nonzero term in their NAF's at the position 2^{n-i}. As mentioned above, the congruence (3.5.9) always has unique solutions. That means we are always able to find the code words with nonzero terms at the position 2^{n-i}, for $i = 0,1,2,\ldots, n-1$.

There are $\left\lceil \frac{B+1}{3} \right\rceil$ integers in the region M_B, for each integer in M_B we can solve an N with respect to a fixed i. In other words, in the NAF of all the code words, there are exact $\left\lceil \frac{B+1}{3} \right\rceil$ code words with nonzero terms at the position 2^{n-i}. This argument holds true for all i, $0 \leq i \leq n-1$, hence there are totally $n \cdot \left\lceil \frac{B+1}{3} \right\rceil$ nonzero terms which are distributed among the NAF's of all the nonzero codewords.

The number of the nonzero codewords is B-1, hence if the minimum distance of the code is D_{min}, we have the following theorem:

Theorem 3.5.2: The minimum distance, D_{min}, of a cyclic AN-code of length n is at most as large as:

$$D_{min} \cdot (B-1) \leq \left\lceil \frac{B+1}{3} \right\rceil \cdot n \qquad (3.5.11)$$

where B is the number of code words.

It is quite interesting to note that for large B's, this bound is approximately equal to $\frac{n}{3}$, that means the distance of the code can not exceed one third of the length of the code!

If we fix the length n of the code, then for increasing values of B, the upper bound decreases. This is because the rate of the code increases as the number of code words increases, while the error-correcting capability of the code decreases as the rate gets higher.

4. IMPLEMENTATION OF ARITHMETIC CODES BY USING MAJORITY LOGIC

4.1 Introduction

One of the major unsolved problems in the theory of arithmetic codes is the implementation of decoding circuitry. Because the decoders must be used frequently in systems using coding redundancy, the practicality of coding redundancy depends on how complex the decoders are. Also, unlike the usual situation in a data transmission system, the delay introduced in the decoding circuit is always important to a digital computing system. Consequently, a practical decoder should have simple logic circuitry and the decoding procedure should be fast.

The decoding of arithmetic codes for multiple error correction has been shown to be difficult. Since the operation of the codes is an integer or modulo arithmetic, the only information that decorder receives is one equation (or one congruent relation) in several integral unknowns. In terms of number theory, the decoding equation is usually a Diophantine equation, which is extremely difficult to be solved. The only known decoding scheme for multiple error correction arithmetic codes is the "permutation of residue" technique, obtained by Laste and Tsao-Wu [30] and Hong [29]. In the following, we shall briefly discuss the permutation of residue technique, to see how difficult this approach is.

The first step of the "permutation of residue" decoding scheme is to calculate the decoding index k, which is defined as the smallest positive integer such that $k\,A > \max_{E} |E_{min}|$; where E is the error and E_{min} is some cyclic shift of E. The syndrome $S \equiv E \bmod A$ is then computed; calculate $S_i \equiv 2^i\, S \bmod A$ for each left shift of S mod A. For each i, the arithmetic weight of S_i is computed, to see if $W(S_i)$ is within the error correcting capability of the code. If none of the S_i's satisfy $W(S_i) \leq t$, where t is the error-correcting capability of the code, the $S_i' = A - S_i \equiv 2^i(A-S) \bmod A$ are calculated one by one; for each i the arithmetic weight $W(S_i')$ is counted to see if $W(S_i') \leq t$. The same procedure goes through all $S_i + jA$ and $S_i' + jA$ for all $0 \leq j \leq k-1$, $0 \leq i < n$, where n is the length of the code, until for some i_d, j_d, the inequalities $W(S_{i_d} + j_dA) \leq t$ or $W(S_{i_d}' + j_dA) \leq t$ are satisfied, we obtain the i_d-th cyclic shift of the error pattern E.

It is easy to see that the essence of this approach is a table look-up attempt. Although the amount of search is reduced from that required by a brute force approach, the "permutation of residue" scheme still involves a great deal of searching. Also, a great deal of complicated calculations must be done by the decoder, and the decoding delay would be considerable due to the large number of shifting cycles.

If the "permutation of residue" decoding scheme was implemented in a digital computing system, it would be natural for one to ask "can this decoder be used to provide as good or better use of redundant parts than do other techniques?" Therefore, how to find decoding algorithms that can be easily implemented is a problem of both theoretical importance and practical significance.

Since the majority-logic decoding (MLD) scheme is by far the easiest one that can be applied to many data communication channels, it would certainly be very interesting if the decoding of arithmetic codes in a digital computing system can also be done by using majority logic. This is not clear, however, due to the propagation of arithmetic errors as a result of carry or borrow failure.

In this chapter, we are able to divise a majority-logic decoding scheme for some arithmetic AN-codes. We will show that the cyclic AN-codes constructed in the previous chapter are all majority-logic decodable. Furthermore, this majority-logic decoding scheme can be generalized to a multi-step majority-logic decoding algorithm, the number of steps required for decoding is related to the number of prime factors contained in the length of the code. With this majority-logic decoding scheme, the implementation of the cyclic AN-code in a real digital computing system would be much simpler.

4.2 Errors in Modular Arithmetic

The arithmetic in most digital computers is arithmetic modulo m where either $m = 2^n-1$ (one's complement arithmetic) or $m = 2^n$ (two's complement arithmetic). In an arithmetic modulo m, the integers under consideration are the set of integers $\{0, 1, 2, \ldots, m-1\}$, and the additive operation is defined as:

$$I_1 + I_2 = R_m (I_1+I_2) \qquad (4.2.1)$$

where $0 \leq I_1, I_2 \leq m-1$, and $R_m(x)$ denotes the residue of x modulo m. Similarly the multiplicative operation is defined as,

$$I_1 \cdot I_2 = R_m (I_1 \cdot I_2) \qquad (4.2.2)$$

The set of integers $\{0, 1, 2, \ldots, m-1\}$ under the above two operations forms a ring, called the ring of integers modulo m. For any integer $I \neq 0$ in the ring, its negative (or the additive inverse) is the integer m-I.

In an arithmetic unit which performs arithmetic modulo m, we shall always assume that the result of any operation is represented by some integer R such that $0 \leq R < m$. If the arithmetic is modulo 2^n-1, we have an n-stage one's complement adder. A cyclic AN-code of length n, where $AB = 2^n-1$, can be used for error-detection or correction in this adder. When two code words AN_1 and AN_2 are added, the sum has the form:

$$\text{Sum of } (AN_1+AN_2) = \bar{R}_m(AN_1+AN_2) \qquad \text{where } m = 2^n-1$$

$$= A \cdot R_B\,(N_1+N_2) \tag{4.2.3}$$

Since $0 \leq R_B(N_1+N_2) < B$, it follows that the modulo sum of any two code words is another code word in the cyclic AN-code. Hence, a cyclic AN-code is linear under modulo arithmetic.

If an error has occurred during the additive operation, we first note that the absolute value of the error is less than 2^n-1, for the error is caused by adding logic failure and there are only n adding units in the adder. An error may change the correct sum (which is a code word) into an integer which is greater than 2^n-1, in that case the actual result from the output of the adder is the modulo sum of the correct code word and the error pattern:

$$R = \text{actual sum} = R_m[A \cdot R_B(N_1+N_2) + E\,] \tag{4.2.4}$$

The syndrome S associated with the possibly erroneous result R is defined to be the residue of R modulo A, i.e. $S = R_A(R)$. Since $2^n-1 = m = AB$, we have:

$$S = R_A(R) = R_A[R_{AB}(A \cdot R_B(N_1+N_2)) + E]$$
$$= R_A(E) \qquad (4.2.5)$$

Therefore, the syndrome S is uniquely determined by the residue of E modulo A. There are B distinct error patterns, namely, E, E + A, E + 2A, ... , E + (B-1) A which have the same syndrome S, they form a coset expansion of the principal ideal $\{AN | 0 \leq N \leq B-1\}$ in the ring of integers modulo 2^n-1. The coset leader is the pattern that has a minimum arithmetic weight; it is the error pattern that is most likely to occur. The d ..der then attempts to identify this most likely occurred error pattern from the syndrome by some intellegent scheme.

4.3 Majority Logic Decoding Scheme for Arithmetic Codes

Let us consider a cyclic AN-code with a composite block length $n = n_1 \cdot n_2$. Since $n_1 | n$ implies $(2^{n_1}-1) | (2^n-1)$, we may choose the generator A_o as:

$$A_o = (2^n-1)/(2^{n_1}-1)$$
$$= 2^{n_1(n_2-1)} + 2^{n_1(n_2-2)} + \ldots + 2^{n_1} + 1 \qquad (4.3.1)$$

In this A_oN-code, all codewords are of the form A_oN with N = 0, 1, 2, ... , $2^{n_1}-2$. If we divide the n bits of a codeword into n_2 blocks, then each block is of length n_1. Since N is less than $2^{n_1}-1$, we have enough positions to express N in its binary form at each block. Therefore, the binary form of any codeword A_oN consists of n_2 replicas of the binary form of N. It has been indicated by Erosh [23] that the minimum distance of this code is equal to n_2, hence this code can correct all arithmetic errors of weight up to $\lceil (n_2-1)/2 \rceil$.

Suppose that two coded numbers A_oN_1, A_oN_2 are added in an adder which performs modulo 2^n-1 arithmetic, and an error E occurs, then the incorrect sum R from the output of the adder is:

$$R \equiv A_o(N_1+N_2) + E \quad \text{mod } 2^n-1 \tag{4.3.2}$$

Since R is an integer in the ring of integers modulo 2^n-1, $0 \leq R < 2^n-1$; we have:

$$R \equiv A_o\, N_3 + E \quad \text{mod } 2^n-1 \tag{4.3.3}$$

where

$$N_3 = R_B\ (N_1+N_2)$$

We assume that E is a correctable error pattern, then $W(E) \leq \lceil (n_2-1)/2 \rceil$. To decode, the decoder wants to recover the correct sum A_oN_3 from the res[illegible] R. We now prove the following lemma and theorems which yield the majority decoding scheme:

Lemma 4.3.1: The actual sum A_oN_3 is a zero word if and only if the modular weight of the result R is less than or equal to $[(n_2-1)/2]$.

Proof: Suppose that $A_oN_3 = 0$, the (4.3.3) can be written as:

$$R = E \quad \text{mod } 2^n-1 \tag{4.3.4}$$

which implies that $R = E$ for $E > 0$ or $R = (2^n-1) + E$ for $E < 0$. Hence, $W_m(R) = W_m(E) \leq [(n_2-1)/2]$.

On the other hand, if $W_m(R) \leq [(n_2-1)/2]$, we have:

$$\begin{aligned} W_m(A_oN_3) &= W_m(R-E) \\ &\leq W_m(R) + W_m(E) \quad \text{(by triangle inequality)} \\ &\leq [(n_2-1)2] + [(n_2-1)/2] \\ &\leq n_2-1 \qquad (4.3.5) \end{aligned}$$

which implies that $A_oN_3 = 0$ since all nonzero code words are of weight at least n_2.

Q.E.D.

Lemma 4.3.1: determines the actual sum to be zero or not. For $W_m(R) > [(n_2-1)/2]$, we have the following arguments.

We first consider how the binary bits of a correct sum A_oN_3 are affected by a single arithmetic error. A single arithmetic error may cause several bits to be in error because of the influecne of carry or borrow propagation. Sometimes an error may cause a carrry process which changes the code word into an integer greater than 2^n-1. In a modulo 2^n-1 adder (i.e., one's complement arithmetic), there are only n positions for the binary representation of the result R. Hence, the carry propagation may go beyond the bit with position 2^{n-1} and then change cyclically the bits with positions 2^0, 2^1, 2^2 ... and so on. However, any carry propagation will stop whenever a zero bit is reached. Similarly, any borrow propagation will stop whenever a bit 1 is reached. Since the generator A_o of the code is of the form as shown in (4.3.1), any nonzero code word can have at most n_1-1 consecutive 0's or 1's in its binary form (we note that the bits with positions 2^{n-1} and 2^0 are considered to be consecutive). Starting at the erroneous bit, any carry or borrow propagation caused by a single arithmetic error cannot propagate more than n_1 consecutive positions. In other words, the carry or borrow propagation caused by a single arithmetic error can change at most n_1 consecutive bits of a nonzero code word in the A_oN-code. Thus, we have proved the following theorem:

Theorem 4.3.1: A single arithmetic error at any bit of a nonzero code word in the A_oN-code can change at most n_1 consecutive bits of this code word.

Next, we consider a multiple, but correctable error pattern E. Expressing the error E in its NAF as:

$$E = e_1 2^{i_1} + e_2 2^{i_2} + \ldots + e_t 2^{i_t}$$

$$\text{where } e_j = \pm 1; \quad t \leq [(n_2-1)/2] \qquad (4.3.6)$$

We partition the n binary bits of R into n_1 disjoint sets, each set contains n_2 bits with positions $2^k, 2^{n_1+k}, 2^{2n_1+k}, \ldots 2^{(n_2-1)n_1+k}$ for a fixed k, where $0 \leq k \leq n_1-1$. For any fixed k, these n_2 bits are $n_1 + 1$ positions apart from one another, therefore, by theorem 4.3.1, no two bits of the same set can be changed by a single arithmetic error. Since the arithmetic weight of E is t, at most t of the n_2 bits with positions $2^k, 2^{n_1+k}, 2^{2n_1+k}, \ldots 2^{(n_2-1)n_1+k}$ can be altered by E, the rest remain unchanged as in the binary form of the correct sum A_oN_3. But E is a correctable error, t is less than or equal to $[(n_2-1)/2]$, thus we have proved the following theorem:

Theorem 4.3.2: If the error is of weight less than or equal to $[(n_2-1)/2]$, then in the binary form of R, the majority of the bits with positions:

$$2^k,\ 2^{n_1+k},\ 2^{2n_1+k},\ \dots,\ 2^{(n_2-1)n_1+k}$$

where

$$0 \le k \le n_1-1$$

remain the same as in the binary form of the correct code word.

From the lemma and theorems, we now summarize the decoding algorithm for the A_oN-code generated by $A_o = (2^n-1)/(2^{n_1}-1)$ as follows:

(1) If $W_m(R) \le [(n_2-1)/2]$, decode $A_oN_3 = 0$.

(2) If $W_m(R) > [(n_2-1)/2]$, work on the n_1 sets of bits in the binary form of R with positions 2^k, 2^{n_1+k}, 2^{2n_1+k}, ..., $2^{(n_2-1)n_1+k}$ for $k = 0, 1, 2, \dots, n_1-1$. For each set take the majority value of the bits, then form a block of n_1 bits with the n_1 majority value obtained.

It is straightforward to see that the above majority logic decoding scheme is also applicable to the A'_oN-code generated by $A'_o = (2^n-1)/(2^{n_2}-1)$. The decoding procedure is exactly the same except we divide the n bits of R into n_1 blocks, with each block of length n_2.

Since there is no restriction on the parameters n_1 and n_2, the majority-logic decoding scheme can be applied quite generally to any A_oN or A'_oN-codes of composite length. When n_1 and n_2 are primes, it should be mentioned that the A_oN-code is the p_1p_2-code generated by A_5 and the A'_oN-code is the p_1p_2-code generated by A_6, as discussed in Chapter 3.

4.4 2-Step Majority-Logic Decoding Scheme

It has been shown in the previous section that the cyclic A_0N-code is majority-logic decodable. To decode, we only need one level of majority logic. It is natural to think that by using more than one level of majority logic other cyclic AN-codes might also be decoded easily. In this section, we will derive a 2-step majority-logic decoding scheme which can be applied to the class of multiple error-correcting AN-code described in Chapter 3.

We recall that the codes described in Section 3.3 are generated by the cyclotomic generator $A = (2^n-1)/(2^{n_1}-1)(2^{n_2}-1)$, where $(n_1,n_2) = 1$ and $n = n_1n_2$. If n_1 and n_2 are primes, then the codes are the p_1p_2-codes generated by A_7 in Section 3.4. If we impose one additional condition on the parameters n_1 and n_2 so that $n_2 \geq 2n_1-1$, then the majority-logic decoding can be done as follows:

From theorem 3.3.1, we know that the minimum distance of the cyclic AN-code generated by $A = (2^n-1)/(2^{n_1}-1)(2^{n_2}-1)$ is n_1 provided that $(n_1,n_2) = 1$ and $n_1 < n_2$. Therefore, a correctable error pattern E is of weight $W(E) \leq [(n_1-1)/2]$.

Suppose that two coded numbers AN_1 and AN_2 are added in an adder which performs modulo 2^n-1 arithmetic and a correctable error E occurs, then the incorrect sum R from the output of the adder is:

$$R \equiv AN_3 + E \mod 2^n-1,$$

where

$$N_3 = R_B(N_1+N_2)$$

$$W(E) \leq [(n_1-1)/2] \tag{4.4.1}$$

For simplicity, we denote 2^n-1 by m in the following discussion.

Multiplying both sides of (4.4.1) by $(2^{n_2}-1)$, we have:

$$\begin{aligned} R(2^{n_2}-1) &\equiv A(2^{n_2}-1) \cdot N_3 + E(2^{n_2}-1) \\ &\equiv \frac{(2^n-1)}{(2^{n_1}-1)(2^{n_2}-1)} \cdot (2^{n_2}-1)\, N_3 + E(2^{n_2}-1) \\ &\equiv A_o N_3 + E(2^{n_2}-1) \quad \bmod 2^n-1 \end{aligned} \tag{4.4.2}$$

where $A_o = (2^n-1)/(2^{n_1}-1)$. Denoting the residue of any integer I modulo 2^n-1 by $R_m(I)$, then $R_m[A_oN_3]$ is a code word in the A_oN-code discussed in previous sections. Expressing $E(2^{n_2}-1)$ in NAF and substituting 1 for $2^{n_1n_2}$, we can reduce $E(2^{n_2}-1)$ to an integer E_r with the property that the absolute value of E_r, i.e. $|E_r|$, is less than $2^{n_2n_2}-1$. We note that E_r is not necessarily the same as $R_m(E(2^{n_2}-1))$ since $R_m(E(2^{n_2}-1))$ is always prositive while E_r may be negative.

Equation (4.4.2) can now be rewritten as:

$$R_m[R(2^{n_2}-1)] = R_m[R_m(A_oN_3) + E_r] \tag{4.4.3}$$

The integer $R_m[R(2^{n_2}-1)]$ can be considered as a corrupted code word of the code generated by A_o. Since the cyclic A_oN-code is of distance n_2, and:

$$W(E) \leq [(n_1-1)/2]$$

$$n_2 \geq 2n_1-1$$

$$\begin{aligned} W(E_r) &\leq W(E(2^{n_2}-1)) \\ &\leq 2W(E) \qquad \text{(by triangle inequality)} \\ &\leq n_1-1 \leq [(n_2-1)/2] \end{aligned}$$

the erroı pattern E_r is within the error-correcting capability of the cyclic A_oN-code. Therefore by div ..ı˜ the n bits of $R_m[R(2^{n_2}-1)]$ into n_2 blocks the majority-logic decoding scheme for the A_oN-code will correctly yield the code word $R_m(A_oN_3)$.

To obtain the actual error pattern E, we need the second step. Denoting $\{R_m[R(2^{n_2}-1)] - R_m(A_oN_3)\}$ by E', it can be seen from equations (4.4.2) and (4.4.3) that:

$$E' \equiv E(2^{n_2}-1) \quad \text{mod } 2^n-1 \tag{4.4.4}$$

hence,

$$E'/(2^{n_2}-1) \equiv E \quad \text{mod } (2^n-1)/(2^{n_2}-1) \tag{4.4.5}$$

Recalling that $(2^n-1)/(2^{n_2}-1) = A_o'$, we have:

$$R_m[E'/(2^{n_2}-1)] = R_m[R_m(A_o' \cdot k) + E\,] \text{ for some } k \tag{4.4.6}$$

From (4.4.6), the integer $R_m[E'/(2^{n_2}-1)]$ can be considered as a corrupted code word of the cyclic AN-code generated by A_o'. Since the cyclic $A_o'N$-code is of minimum distance n_1, and since $W(E) \leq [(n_1-1)/2]$, the actual error pattern E can be decoded by the majority-logic scheme for the cyclic $A_o'N$-code.

The 2-step majority-logic decoding scheme derived above can now be summarized as follows:

(1) If $W(R) \leq [(n_1-1)/2]$, decode $AN_3 = 0$.

(2) If $W(R) > [(n_1-1)/2]$, get E' by applying the majority-logic decoding scheme for A_oN-code to $R_m[R(2^{n_2}-1)]$. (first step)

(3) Obtain the actual error pattern E by applying the majority-logic decoding scheme for A'_oN-code to $R_m[E'/(2^{n_2}-1)]$. (second step)

It should be remarked that while the condition $n_2 \geq 2n_1-1$ imposed on the codes generated by $A = (2^{n_1n_2}-1)/(2^{n_1}-1)(2^{n_2}-1)$ guarantees the corection of errors up to minimum distance, the condition is not necessary for the 2-step majority-logic decoding scheme to work. In general the majority-logic decoding scheme will correct up to error patterns of arithmetic weight the smaller of $[(n_1-1)/2]$ and $\frac{1}{2}[(n_2-1)/2]$.

Let us consider the following example:

Example: Suppose that the block length of the code $n = 5.9 = 45$, and the generator $A = (2^{45}-1)/(2^9-1)(2^5-1)$. Then the code is of minimum distance 5, capable of correcting any double arithmetic errors. We have,

$$A = \frac{(2^{45}-1)}{(2^9-1)(2^5-1)} = 2^{31} + 2^{26} + 2^{22} + 2^{21} + 2^{17} + 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2 + 1$$

$$= (000000000000010000100011000110011100111011111)$$

If two coded numbers 37A and 27A are added in a 45-stage one's complement adder, and a double error $E = -2^{38} + 2^8$ occurs, then the incorrect sum R from the output of the adder is, $R = A(37+27) + E = 64A + E$. To decode, we first calculate the residue of $R(2^9-1)$ modulo $2^{45}-1$ as:

00010, 01010, 00010, 00010, 00010, 00110, 00001, 11001, 11110

We divide the above binary form into 9 blocks and check that,

digits with positons		majority value
2^{5k+4}, $0 \leq k \leq 8$	0, 0, 0, 0, 0, 0, 0, 1, 1	0
2^{5k+3}, $0 \leq k \leq 8$	0, 1, 0, 0, 0, 0, 0, 1, 1	0
2^{5k+2}, $0 \leq k \leq 8$	0, 0, 0, 0, 0, 1, 0, 0, 1	0
2^{5k+1}, $0 \leq k \leq 8$	1, 1, 1, 1, 1, 1, 0, 0, 1	1
2^{5k+0}, $0 \leq k \leq 8$	0, 0, 0, 0, 0, 0, 1, 1, 0	0

Hence the majority decision on $R_m(R(2^9-1))$ yeidls a code word:

$R_m(A_oN)$ = 00010, 00010, 00010, 00010, 00010, 00010, 00010, 00010, 00010

The error at this state is $E' = R_m(R(2^9-1)) - R_m(A_oN)$, which has the binary form:

00000, 01000, 00000, 00000, 00000, 00100, 000(-1)1, 110(-1)1, 11100

The actual error pattern E is congruent to $E'/(2^9-1)$ modulo $2^{45}-1$,

$$E(2^9-1) \equiv E' = 2^{38} + 2^{17} - 2^8 - 2^2 \mod 2^{45}-1$$

$$E \equiv 2^{29} + 2^{20} + 2^{11} + 2^8 + 2^2 \mod(2^{45}-1)/(2^9-1)$$

which has the binary form:

000000000, 000000100, 000000100, 000000100, 100000100 (4.4.7)

Again, the majority-logic scheme on (4.4.7) yields a block 000000100. Repeating this block five times, we have:

$$000000100,\ 000000100,\ 000000100,\ 000000100,\ 000000100 \quad (4.4.8)$$

The binary integer (4.4.8) is a code word generated by $A_o' = (2^{45}-1)/(2^9-1)$. Subtracting (4.4.8) from (4.4.7), we get the actual error pattern E:

$$000000(-1)00,\ 000000000,\ 000000000,\ 000000000,\ 100000000$$

Hence, the error pattern E is decoded as $-2^{38}+2^8$.

4.5 L-Step Majority-Logic Decoding Scheme for Arithmetic Codes

In the previous section we have shown that the cyclic AN-code generated by $A = (2^{n_1 n_2}-1)/(2^{n_1}-1)(2^{n_2}-1)$ is 2-step majority-logic decodable provided that $n_2 \geq 2n_1-1$. If n_1 and n_2 are two prime integers, then the code is the p_1p_2-code generated by A_7. The reason that the decoding needs 2-step of majority logic is that the denominator of the generator A contains two factors of the form 2^a-1. We must multiply the form generator A by a factor 2^a-1 so that the modified generator of the code has a form the same as A_o or A_o', then the majority-logic decoding scheme can be applied. After the first step of majority-logic decision, we divide the result by the factor 2^a-1, the generator of the code at this stage is $(2^n-1)/(2^a-1)$, which has the same form as A_o or A_o', hence the second step of majority logic is applied.

It is not difficult for one to realize that the number of factors of the form 2^a-1 contained in the denominator of the generator is related to the number of steps required for majority-logic decoding. As a matter of fact, the 2-step majority-logic decoding scheme developed in the previous section can be generalized in a straightforward manner to the following theorem:

Theorem 4.5.1: Let $n_1, n_2, \ldots, n_L$ be L pairwise relatively prime integers w'th the condition that $n_{i+1} \geq 2n_i$ for $i = 1,2,\ldots, L-1$; then the cyclic AN-code of length $n = \prod_{i=1}^{L} n_i$ generated by:

$$A = (2^n-1)/\prod_{i=1}^{L}(2^{n_i}-1) \tag{4.5.1}$$

can be majority logically decoded in exactly L stops.

Proof: The proof of this theorem is essentially the same as the proof of 2-step majority-logic decoding scheme. We start with the decoding equation:

$$R = AN + E \tag{4.5.2}$$

Multiplying both sides of (4.5.2) by $\prod_{i=2}^{L}(2^{n_i}-1)$, we have:

$$R \cdot \prod_{i=2}^{L}(2^{n_i}-1) = [A\cdot \prod_{i=2}^{L}(2^{n_i}-1)]\, N + E \cdot \prod_{i=2}^{L}(2^{n_i}-1)$$

$$= A_1 N + E \cdot \prod_{i=2}^{L}(2^{n_i}-1)$$

where

$$A_1 = (2^n-1)/(2^{n_1}-1) \tag{4.5.3}$$

Taking the residue of both sides of (4.5.3) modulo 2^n-1. we have:

$$R_m[R\cdot \prod_{i=2}^{L}(2^{n_i}-1)] = R_m[R_m(A_1N) + E_{r1}] \tag{4.5.4}$$

Where E_{r1} is obtained by substituting 1 for 2^n in the NAF of $E\cdot \prod_{i=2}^{L}(2^{n_i}-1)$.

We assume that the error pattern E is of weight less than or equal to $\lceil (\prod_{i=1}^{L-1} n_i -1)/2 \rceil$, where $\lceil x \rceil$ denotes the integeral part of x, then we have:

$$W(E_{r1}) \le W[E\cdot \prod_{i=2}^{L}(2^{n_i}-1)] \le 2^{L-1}\, W(E) \le 2^{L-1} \lceil (\prod_{i=1}^{L-1} n_i -1)/2 \rceil \tag{4.5.5}$$

$$n_{i+1} \ge 2n_i \quad \text{for} \quad i = 1, 2, \ldots, L-1 \tag{4.5.6}$$

Therefore,

$$2^{L-1} \cdot \prod_{i=1}^{L-1} n_i \le \prod_{i=2}^{L} n_i \tag{4.5.7}$$

$$W(E_{r1}) \le \lceil \{ (\prod_{i=2}^{L} n_i) -1 \}/2 \rceil \tag{4.5.8}$$

Since the integer A_1 is of the same form as A_o in section 4.3, it generates a cyclic A_1N-code of minimum distance $\prod_{i=2}^{L} n_i$. Also, $R_m(A_1N)$ is a code word in the A_1N-code, and the integer $R_m[R\cdot \prod_{i=2}^{L}(2^{n_i}-1)]$ can be considered as a corrupted code word in the A_1N-code, the error E_{r1} is within the error correcting capability of the A_1N-code. Therefore, the majority-logic decoding scheme can be applied to $R_m[R\cdot \prod_{i=2}^{L}(2^{n_i}-1)]$, we get $R_m(A_1N)$ correctly.

Defining E_1 by the difference of $R_m[R\cdot \prod_{i=2}^{L}(2^{n_i}-1)]$ and $R_m(A_1N)$ we have:

$$E_1 \equiv E\cdot \prod_{i=2}^{L}(2^{n_i}-1) \mod 2^n-1$$

$$E_1/(2^{n_2}-1) \equiv E\cdot \prod_{i=3}^{L} (2^{n_i}-1) \mod (2^n-1)/(2^{n_2}-1) \tag{4.5.9}$$

$$R_m[E_1/(2^{n_2}-1)] = R_m[R_m(A_2\cdot t) + E_{r2}] \text{ for some } t \tag{4.5.10}$$

where $A_2 = (2^n-1)/(2^{n_2}-1)$ and E_{r2} is obtained by substituting 1 for 2^n in the NAF of $E\cdot \prod_{i=3}^{L} (2^{n_i}-1)$.

The integer $R_m[E_1/(2^{n_2}-1)]$ can be considered as a corrupted code word of the cyclic AN-code generated by A_2, which is of minimum distance $n_1\cdot \prod_{i=3}^{L} n_i$. Since,

$$W(E_{r2}) \leq W[E\cdot \prod_{i=3}^{L} (2^{n_i}-1)] \leq 2^{L-2}\, W(E) \leq 2^{L-2} \left\lceil (\prod_{i=1}^{L-1} n_i-1)/2 \right\rceil$$

$$2^{L-2} \cdot \prod_{i=1}^{L-1} n_i = n_1 \cdot 2^{L-2} \prod_{i=2}^{L-1} n_i \leq n_1 \cdot \prod_{i=3}^{L} n_i \tag{4.5.11}$$

$$W(E_{r2}) \leq \left\lceil ((n_1\cdot \prod_{i=3}^{L} n_i)-1)/2 \right\rceil \tag{4.5.12}$$

the error E_{r2} is a correctable pattern with respect to the AN-code generated by A_2. Using majority-logic decoding scheme on $R_m[E_1/(2^{n_2}-1)]$ we can determine $R_m(A_2\cdot t)$ correctly. Defining E_2 by the difference of $R_m[E_1/(2^{n_2}-1)]$ and $R_m(A\cdot t)$, we have:

$$E_2 \equiv E \cdot \prod_{i=3}^{L} (2^{n_i}-1) \mod 2^n-1 \tag{4.5.13}$$

which has the same form as the congruence (4.5.8). The same majority decoding method can be applied and we get another congruence of the same form. So in general, at the h-th stage we have:

$$E_{h-1} \equiv E \cdot \prod_{i=h}^{L} (2^{n_i}-1) \mod 2^n-1 \tag{4.5.14}$$

$$E_{h-1}/(2^{n_h}-1) \equiv E \cdot \prod_{i=h+1}^{L} (2^{n_i}-1) \mod A_h \tag{4.5.15}$$

where $A_h = (2^n-1)/(2^{n_h}-1)$.

Therefore, at the L-th stage,

$$E_{L-1} \equiv E \cdot (2^{n_L}-1) \mod 2^n-1$$

$$E_{L-1}/(2^{n_L}-1) \equiv E \mod A_L$$

We see that $W(E) \leq \lceil ((\prod_{i=1}^{L-1} n_i)-1)/2 \rceil$ and the cyclic AN-code generated by A_L is of minimum distance $\prod_{i=1}^{L-1} n_i$, hence the majority logic decoding shceme on $E_{L-1}/(2^{n_L}-1)$ will correctly yield the actual error pattern E.

Q.E.D.

In the proof of the above theorem, we have assumed that the weight of E is less than or equal to $\lceil ((\prod_{i=1}^{L-1} n_i)-1)/2 \rceil$; the fact that the L-step majority-logic decoding can correct errors up to this weight gives the lower bound on the minimum distance of the code, hence we have the following:

Corollary 4.5.1: The cyclic AN-code as stated in the theorem 4.5.1 is of distance at least as great as $\prod_{i=1}^{L-1} n_i$.

The result of the above theorem is a straightforward generalization of the 2-step majority-logic decoding scheme. The code constructed has rather high redundancy. To improve the efficiency of the code, we want to reduce some unnecessary factors contained in A. In the following, we shall construct another class of codes which is also L-step majority-logic decodable. This code is generated by a cyclotomic generator, it has better efficiency and a very general structure. Although the decoding for this new code is similar to that of the code in theorem 4.5.1, we shall go through the discussion in detail because some important remarks must be made.

Definition 4.5.1: Any positive integer n can be expressed as a product of powers of prime factors, i.e. $n = \prod_i p_i^{s_i}$. This product of prime powers form is unique for any n and is called the canonical form of n.

Theorem 4.5.2: Let the canonical form of a positive integer n be $n = \prod_{i=1}^{L} p_i^{s_i}$. If the distinct prime factors p_i's satisfy the condition that: $p_i \geq 2^{i-1}(p_1-1) + 1$ for i = 2, 3, ..., L; then the cyclic AN-code of block length n generated by $Q^n(2)$, the primitive cyclotomic factor, can be majority-logic decoded in exactly L steps.

Proof: First, let us consider the minimum distance of the code. Since $A = Q^n(2)$, the integer $(2^n-1)/(2^{n/p_1}-1)$ is a multiple of A, hence it is a code word. The arithmetic weight of $(2^n-1)/(2^{n/p_1}-1)$ is p_1, hence the minimum distance of the code is at most p_1. If E is a correctable error pattern, then $W(E) \leq (p_1-1)/2$. We start with the decoding equation:

$$R = AN + E \tag{4.5.16}$$

Multiplying both sides of (4.5.16) by $\prod_{i=1}^{L-1} (2^{n/p_i}-1)$,

$$R \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) = A \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) \cdot N + E \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) \tag{4.5.17}$$

Let $A_1 = A \cdot k = (2^n-1)/(2^{n/p_L}-1)$, then $B = (2^n-1)/A = k \cdot (2^{n/p_L}-1)$ and the code generated by A_1 is of minimum distance p_L. Since,

$$B \cdot Q^n(2) = (2^n-1) = \prod_{n/d} Q^d(2) \tag{4.5.18}$$

the cyclotomic factors contained in B are all the $Q^d(2)$'s with d a proper divisor of n. Clearly, these $Q^d(2)$'s are also contained in $\prod_{i=1}^{L} (2^{n/p_i}-1)$, hence B divides $\prod_{i=1}^{L} (2^{n/p_i}-1)$; which implies that k divides $\prod_{i=1}^{L-1} (2^{n/p_i}-1)$.

Therefore, (4.5.17) can be rewritten as:

$$R \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) = (A \cdot k) \cdot M + E \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) \quad \text{for some } M$$

$$= A_1 M + E \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1) \tag{4.5.19}$$

Taking the residues of both sides of (4.5.19) modulo 2^n-1, we have:

$$R_m[R \cdot \prod_{i=1}^{L-1} (2^{n/p_i}-1)] = R_m[R_m(A_1 M) + E_{r1}] \tag{4.5.20}$$

where E_{r1} is obtained by substituting 1 for 2^n in the NAF of $E\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1)$. In addition,

$$p_L \geq 2^{L-1}(p_1-1)+1$$

$$W(E) \leq (p_1-1)/2$$

$$W(E_{r1}) \leq W[E\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1)] \leq 2^{L-1}\,W(E)$$

$$\leq 2^{L-1}(p_1-1) \leq (p_L-1)/2 \qquad (4.5.21)$$

Hence, E_{r1} is a correctable error pattern with respect to the cyclic AN-code generated by A_1. Also, $R_m[R\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1)]$ can be considered as a corrupted code word of the A_1N-code, then the majority-logic decoding scheme for A_1N-code can be applied to $R_m[R\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1)]$ and the code word $Rm(A_1M)$ can be determined correctly. Defining E_1 by the difference of $Rm[R\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1)]$ and $R_m(A_1M)$, we have:

$$E_1 \equiv E\cdot\prod_{i=1}^{L-1}(2^{n/p_i}-1) \mod 2^n-1 \qquad (4.5.22)$$

$$E_1/(2^{n/p_{L-1}}-1) \equiv E\cdot\prod_{i=1}^{L-2}(2^{n/p_i}-1) \mod (2^n-1)/(2^{n/p_{L-1}}-1) \qquad (4.5.23)$$

$$R_m[E_1/(2^{n/p_{L-1}}-1)] = R_m[R_m(A_2\cdot t)+E_{r2}] \text{ for some } t \qquad (4.5.24)$$

where $A_2 = (2^n-1)/(2^{n/p_{L-1}}-1)$ and E_{r2} is obtained by substituting 1 for 2^n in the NAF of $E\cdot\prod_{i=1}^{L-2}(2^{n/p_i}-1)$. Since,

$$W(E_{r2}) \le W(E \cdot \prod_{i=1}^{L-2} (2^{n/p_i}-1)) \le 2^{L-2}\, W(E)$$

$$\le 2^{L-2}(p_1-1)/2 \le (p_{L-1}-1)/2 \qquad (4.5.25)$$

and the code generated by A_2 is of distance p_{L-1}, E_{r2} is a correctable error. Applying majority-logic decoding scheme to $R_m[E_1/(2^{n/p_{L-1}}-1)]$, we can determine the code word $R_m(A_2 t)$ correctly. Defining E_2 by the difference of $E_1/(2^{n/p_{L-1}}-1)$ and $R_m(A_2 \cdot t)$, we have:

$$E_2 \equiv E \cdot \prod_{i=1}^{L-2} (2^{n/p_i}-1) \mod 2^n-1 \qquad (4.5.26)$$

which has the same form as the congruence (4.5.22). The same majority decoding method can be applied to (4.5.26) and we obtain another congruence of the same form. So in general, at the h-th stage we have:

$$E_{h-1} \equiv E \cdot \prod_{i=1}^{L-h+1} (2^{n/p_i}-1) \mod 2^n-1$$

$$E_{h-1}/(2^{n/p_{L-h+1}}-1) \equiv E \cdot \prod_{i=1}^{L-h} (2^{n/p_i}-1) \mod A_h$$

wher $A_h = (2^n-1)/(2^{n/p_{L-h+1}}-1)$.

Therefore, at the L-th stage we have:

$$E_{L-1} \equiv E \cdot (2^{n/p_1}-1) \mod 2^n-1 \qquad (4.5.27)$$

$$E_{L-1}/(2^{n/p_1}-1) \equiv E \mod A_L \qquad (4.5.28)$$

$$R_m[E_{L-1}/(2^{n/p_1}-1)] = R_m[R_m(A_L \cdot u) + E] \text{ for some } u \qquad (4.5.29)$$

where $A_L = (2^n-1)/(2^{n/p_1}-1)$. Since the code generated by A_L is of minimum distance p_1 and $W(E) \leq (p_1-1)/2$, the majority-logic decoding shceme on $R_m[E_{L-1}/(2^{n/p_1}-1)]$ will correctly yield the actual error pattern E.

Q.E.D.

As mentioned previously, the code generated by $Q^n(2)$ is of minimum distance at most p_1. But from the above theorem, it has been shown that the code can correct all arithmetic errors of weight up to $(p_1-1)/2$, hence we have the following:

Corollary 4.5.2: If the code length $n = \prod_{i=1}^{L} p_i^{s_i}$, and $p_i \geq 2^{i-1}(p_1-1) + 1$ for $i = 2, 3, \ldots L$, $s_i \geq 0$ for all $i = 1, 2, \ldots, L$, then the minimum distance of the cyclic AN-code generated by $Q^n(2)$ is exactly equai to p_1.

Again, it should be mentioned that while the conditions $p_i \geq 2^{i-1}(p_1-1) + 1$ for $i = 2, 3, \ldots L$, imposed on the code guarantee the correction of errors up to minimum distance, the conditions are not necessary for the L-step majority-logic decoding scheme to work.

The majority-logic decoding scheme developed in this chapter is a completely different approach as compared with the permutation of residues technique, no search is required at all. Those majoriy-logic decodable AN-codes can then be implemented easily. It is our hope that the majority-logic decoding scheme presented here will be applicable to other classes of AN-codes and will lead to other practical decoding implementations.

5. APPLICATIONS OF CYCLIC AN-CODES TO COMPUTER ARITHMETIC

5.1 Concurrent Checking in Computer Arithmetic

A logic fault occurs in a digital computing system when a defective component or external interference causes a deviation of a logic component from its prescribed value. In general, there are two approaches which can be employed to check the presence of errors. In periodic checking, ordinary operation is periodically interrupted and a checking program is carried out which detects the presence of an error and indicates its location. In concurrent checking, special logic circuits are used to detect the presence of errors in computer words concurrently with ordinary operation of the computer. Both the well-known replication of processors with output voting and the arithmetic encoding of the operands followed by the application of decoding algorithm to the results are the methods of concurrent checking in digital computing systems.

In the triplication of processor method, the majority vote-taker gives the correct result only when two or three of the identical processors operate normally. If more than one processor had errors, the majority voting scheme would not work. In other words, the application of the triplication of processor is based on the assumption that all the logic faults are restricted to only one of the three identical processors. This model is questionable sometimes, especially when multiple error patterns occur in a complicated processor.

On the other hand, when the arithmetic coding method is used, the redundancy is incorporated into the operands themselves which are being processed. A multiple error-correcting code is necessary for some processors when single error protection is not reliable enough. As pointed out in the previous chapters, the decoding of multiple errors is much more difficult than the decoding of single error, the reliability of the complicated decoder circuitry can not be overlooked. The assumption that decoders are error-free then is not reasonable.

In this chapter, a new concurrent checking technique, called the "separate checking" is studied. This separate checking technique possesses the advantages of both the coding approach and the replication of processors. It combines the ideas of coding and replication of processors properly so that the error model is more realistic and the checking procedure is much simpler. Furthermore, since only single error-correcting codes are used in this technique, the decoding is simple and thus reliable.

The application of cyclic AN-codes to the high speed multiplier is also investigated in this chapter. It will be shown that the majority-logic decodable cyclic AN-codes constructed in the previous chapters can be applied practically. All these applications render the evidence that the arithmetic coding is a promising approach to the study of digital computing systems diagnosis and reliability.

5.2 Separate Concurrent Checking Techniques

Suppose that two integers $N_1 = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ and $N_2 = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)$ are added in a n-stage one's complement binary adder. If an error E occurs, then the result from the output of the adder is

$$R = \text{Res}[\text{Res}(N_1+N_2) + E] \qquad (5.2.1)$$

where Res(·) denotes the residue of x modulo 2^n-1.

Since the operands N_1 and N_2 are not coded, it is not possible for one to correct the error from the erroneous sum R. Instead of putting redundancy into N_1 and N_2, we use several separate checkers to execute the fault-tolerant operation as follows:

Let us call the n-stage one's complement adder the "main adder." We use several independent checkers, each of these checkers is also an adder, performing modulo $2^{n_i}-1$ arithmetic, where n_i is less than n. In other words, each checker is just an n_i-bit long one's complement arithmetic unit. If the n_i's are small, we assume that only a small number of errors occur in the shorter adding units.

Suppose we assume that only single error occurs in each short adding unit, then instead of encoding the integers N_1 and N_2 in the main adder, we encode them by different single-error-correcting cyclic AN-codes and add them in the independent checkers. Let us assume that k independent checkers are used, each of length n_i, $1 \leq i \leq k$; the single error correcting cyclic AN-code used in the i-th checker is generated by A_i, where $A_i B_i = 2^{n_i}-1$.

Since each checker performs modulo $2^{n_i}-1$ arithmetic, the possible erroneous sum from the i-th checker is

$$r_i = R_i[R_i(A_i(N_1+N_2))+E_i] \qquad (5.2.2)$$

where $R_i(x)$ denotes the residue of x modulo $2^{n_i}-1$. Since $A_i B_i = 2^{n_i}-1$, (5.2.2) can be rewritten as

$$r_i = R_i[A_i R_{B_i}(N_1+N_2)+E_i] \qquad (5.2.3)$$

where $R_{B_i}(x)$ denotes the residue of x modulo B_i.

Since A_i generates a single error correcting code, the single error pattern E_i can be easily corrected and the residue $R_{B_i}(N_i+N_2)$ can be obtained from the output of the decoder. The same argument holds for all the independent checkers, hence we can determine all the residues $R_{B_i}(N_1+N_2)$ for i = 1, 2, ...,k.

Suppose that the lengths of the checkers are properly chosen so that all the B_i's are pairwise relatively prime, i.e.,

$$(B_i,B_j) = 1 \quad \text{for } i \neq j; \quad 1 \leq i,\ j \leq k \qquad (5.2.4)$$

where (a,b) denotes the greatest common divisor (or g.c.d.) of the integers a and b. Writing

$$\begin{aligned} N_1 + N_2 &\equiv R_{B_i}(N_1+N_2) \quad \text{mod } B_1 \\ N_1 + N_2 &\equiv R_{B_2}(N_1+N_2) \quad \text{mod } B_2 \\ &\vdots \\ N_1 + N_2 &\equiv R_{B_k}(N_1+N_2) \quad \text{mod } B_k \end{aligned} \qquad (5.2.5)$$

By (5.2.4) and the well-known Chinese Remainder theorem [49], we can solve the simultaneous congruence (5.2.5) and uniquely determine the residue $N_1 + N_2$ modulo $\prod_{i=i}^{k} B_i$. We have

$$N_1 + N_2 = N + t \cdot \prod_{i=1}^{k} B_i\ , \quad \text{for some } t \qquad (5.2.6)$$

where $N = [(N_1+N_2) \text{ mod } \prod_{i=1}^{k} B_i]$.

Let us assume the cyclic AN-code generated by A_i is of rate $(rt)_i$, and the independent checkers are so chosen that the product

$$2^{n+2} > \prod_{i=1}^{k} B_i > 2^{n+1} \qquad (5.2.7)$$

Since $0 \leq N_1, N_2 < 2^n-1$, we have

$$0 \leq N_1 + N_2 < 2(2^n-1) \qquad (5.2.8)$$

Combining (5.2.6), (5.2.7) and (5.2.8), we have $t = 0$ and

$$N_1 + N_2 = N \qquad (5.2.9)$$

Therefore, the output of the decoder gives the correct sum of N_1 and N_2. In other words, instead of using the longer n-stage adder, the addition of N_1 and N_2 can be executed by the k independent shorter checking adders; we use $\sum_{i=1}^{k} n_i$ elemental adding units to perform the n-bit binary addition, the redundant $\sum_{i=1}^{k} n_i - n$ adding units are incorporated for error correction.

Suppose that the B_i's are chosen to be $B_i \geq 2^{b_i}$, then the rate of the code is

$$(rt)_i = \log_2 B_i/n_i \geq b_i/n_i \qquad (5.2.10)$$

On the other hand, in order to have (5.2.7), the b_i's satisfy

$$\sum_{i=1}^{k} b_i \geq n + 1 \qquad (5.2.11)$$

From (5.2.10) and (5.2.11) it is seen that

$$\sum_{i=1}^{k} (n_i) \geq \sum_{i=1}^{k} b_i/(rt)_i \geq (n+1)/(rt)_{max}$$

$$\frac{n}{\left(\sum_{i=1}^{k} n_i\right)} \leq (rt)_{max} \qquad (5.2.12.a)$$

and

$$\sum_{i=1}^{k} (n_i) = \sum_{i=1}^{k} \left((\log_2 B_i)/(rt)_i\right)$$

$$\leq \left(\sum_{i=1}^{k} \log_2 B_i\right)/(rt)_{min} \leq (n+2)/(rt)_{min}$$

$$\frac{n+2}{\left(\sum_{i=1}^{k} n_i\right)} \geq (rt)_{min} \qquad (5.2.12.b)$$

where $(rt)_{max} = \max_i (rt)_i$ and $(rt)_{min} = \min_i (rt)_i$.

The expression $n/\sum_{i=1}^{k} (n_i)$ is the efficiency of this separate checking technique. From (5.2.12.a) and (5.2.12.b), the efficiency is bounded by the maximum and minimum rates of the independent cyclic AN-codes. To increase the efficiency we should choose the best single-error-correcting AN-code for each independent checker. The single-error-correcting An-codes discovered by Perterson are all perfect, they should be used whenever it is possible. One question is that some perfect codes are constructed with arithmetic modulo $2^k + 1$, while the above discussion assumes that the checkers perform one's complement arithmetic. However, it is easy to see that all the above discussion also holds true when some of the checkers perform arithmetic modulo $2^{n_i}+1$; this broadens the possibility of using perfect codes for the checkers. Let us consider the following example:

Example 5.2.1: Suppose that a 34-stage main adder is checked by separate checking technique. We use six independent checkers. The first five checkers perform one's complement arithmetic, with lengths 9-bit, 10-bit, 11-bit, 12-bit and 15-bit respectively. The cyclic AN-codes used for these five checkers are:

i	n_i	A_i	B_i	D_{min}
-	9	73	7	3
2	10	3.31	11	4
3	11	23	89	3
4	12	5.7	3·3·13	3
5	15	7.31	151	3

For the last checker, we use a 14-bit adder which performs arithmetic modulo $2^{14} + 1$. The AN-code for this checker is a perfect single-error-correcting code (but not cyclic) with generator $A_6 = 29$ and $B_6 = 5 \cdot 113 = 565$.

It can be seen that all the B_i's are pairwise relatively prime. Also,

$$\prod_{i=1}^{6} B_i = 7\cdot 11\cdot 89\cdot 3\cdot 3\cdot 13\cdot 151\cdot 565$$

$$= 68,405,652,315 > 2^{35} = 2^{34+1}$$

Therefore, the 34-stage main adder can be replaced by the six independent adders.

The number of redundant adding units is

$$\sum_{i=1}^{6} n_i - n = (9+10+11+12+14+15)-34 = 37$$

and the efficiency of the overall checked adding system is

$$n/\left(\sum_{i=1}^{6} n_i\right) = 34/71 = 0.48$$

which is greater than one third.

The efficiency of the over-all checked system is better than the system checked by the triplication method; also, not like the triplicated system that all errors are assumed to occur in one of the three identical processors, in separate checking approach we allow errors to occur in any independent checking unit. The assumption that only single error occurs in each independent checker is reasonable when the length of the checker is chosen to be short. In addition, the decoding procedure is done separately. For each checker, the decoding is just a single error correction, which is much simpler and much more reliable when compared with the approach of using an over-all multiple-error-correcting AN-code for the main adder.

5.3 Multiple Iterative Error-Correcting Codes for High Speed Multiplier

In a high speed arithmetic unit, the multiplier is divided into a number of blocks with two (or more) bits each. Each block is then multiplied with the multiplicand to form partial sums. The partial sums are appropriately shifted and added in a multi-input adder with minimum carry provisions. If a faulty circuit occurs in the multiplier, the error patterns have the following properties:

a. Since the partial sums are shifted by multiples of the length of a block, the erroneous bits in each partial sum will be equally spaced when the result is obtained, usually they span a fixed number of blocks in the binary representation of the result. This error is said to be iterative in nature.

b. Since a stuck-at-1 or 0 type of logic fault causes either a carry or a borrow error but not both, all the erroneous bits of a single iterative error are of the same polarity. Utilizing these specific properties of the iterative error patterns, Chien and Hong [16] have found a large class of arithmetic AN-codes which can correct any iterative error caused by single component failure. These codes are good since they possess high efficiency and can be implemented quite easily.

If more than on component of a high speed multipler fail, the errors from the output can be recognized as several groups of single-iterative-errors. We call this type of errors the multiple-iterative-errors. Following the investigation of Chien and Hong's work, however, it is difficult to generalize their single-iterative-error-correcting code. In this section, we apply the majority-logic decodable AN-codes constructed in the previous chapter to the error-correction in high speed multipler. Instead of making correction at the end of multiplication, we correct errors contained in each partial sum; it can be shown in the following that the majority-logic decodable codes are suitable for this purpose, and can correct any multiple iterative-errors.

First, let us consider that the multiplier is divided into r blocks and each block contains m bits, then the multiplier is of length rm. Usually, the multiplicand requires the same number of bits in its binary representation as the multiplier, hence the product requires at most 2rm binary bits. A single iterative error pattern E can be expressed as

$$E = \pm 2^{k} \sum_{i=0}^{2r-1} e_i 2^{mi} \text{ ; where } 0 \leq k < m \text{ and } e_i = 0,1 \qquad (5.3.1)$$

It is easy to see that a multiple, say, t-iterative-error E_t can be expressed as

$$E_t = \pm 2^{k_1} \sum_{i=0}^{2r-1} e_{1i} 2^{mi} \pm 2^{k_2} \sum_{i=0}^{2r-1} e_{2i} 2^{mi} \pm \ldots \pm 2^{k_t} \sum_{i=0}^{2r-1} e_{ti} 2^{mi} \qquad (5.3.2)$$

where $0 \leq k_1, k_2, \ldots, k_t < m$; k_j's are distinct and $e_{ji} = 0, 1$ for all j's and i's.

The error patterns (5.3.1) or (5.3.2) are presented in the final result at the output of a high speed multiplier. If we consider the <u>partial error</u>, which is the error contained in each partial sum, we have the following form partial error at the i-th partial sum

$$\triangleq E_{pi}$$

$$\approx e_{1i}\,2^{k_1+mi} + e_{2i}\,2^{k_2+mi} + \ldots + e_{ti}\,2^{k_t+mi} \qquad (5.3.3)$$

where $e_{ji} = 0$, 1 or -1, $0 \leq k_1, \ldots k_t < m$ and all the k_j's are distinct. Therefore, each partial error of a t-iterative error pattern can be considered as a random error of arithmetic weight t or less. This suggests a possible way of correcting any t-iterative-errors.

We encode the multiplicand by a t-random-error-correcting AN-code. Instead of making correction of the final product we use the t-error-correcting code to correct the partial errors contained in the partial sum. Certainly, the decoding of the t-error code must be very easy, otherwise the approach is highly impractical. The majority-logic decodable AN-codes constructed in Chapters 3 and 4 immediately render the potential of being the t-random-error-correcting codes.

Design of the Codes

Suppose that both the multiplier and the multiplicand require rm bits in their binary representations, then the product needs 2rm binary bits. We want to design an arithmetic AN-code of length 2rm, with the generator A a proper divisor of $2^{2rm}-1$, then the code is cyclic and all the code words AN form a principal ideal in the ring of integers modulo $2^{2rm}-1$.

Let s be a divisor of rm, then $rm = d \cdot s$, for some d. The integer $2^{(d-1)s} + 2^{(d-2)s} + \ldots. + 2^{s} + 1$ can be chosen as the generator of the code.

First, we see that

$$A = 2^{(d-1)s} + 2^{(d-2)s} + \dots + 2^s + 1$$

$$= (2^{rm}-1)/(2^s-1) \tag{5.3.4}$$

Since $A \mid (2^{rm}-1)$, the integer $2^{rm}-1$ is a code word of weight 2 and the code generated by A is of minimum distance only 2. It seems that the code has no error correcting capability, yet it can correct all t-iterative-errors if $2t + 1 \leq d$.

Let the multiplicand be encoded as a code word AM, and let the multiplier be N. In a high speed multiplier the binary form of N is divided into r blocks, each block contains m bits; hence N can be expressed as

$$N = \sum_{i=0}^{r-1} n_i 2^{mi}, \text{ where } 0 \leq n_i \leq 2^m-1 \tag{5.3.5}$$

Denoting S_i as the i-th partial sum, we have

$$S_i = AM \cdot n_{i-1} \quad \text{for } i = 1, 2, \dots, r \tag{5.3.6}$$

Since $0 \leq AM < 2^{rm}-1$ and $0 \leq n_{i-1} \leq 2^m-1$,

$$0 \leq S_i < 2^{(r+1)m}-1 \tag{5.3.7}$$

If there exist some faulty circuits in the multiplier, the partial sum is corrupted with the partial error,

$$Y_i = S_i + E_{pi} = AMn_{i-1} + E_{pi} \tag{5.3.8}$$

where Y_i is the erroneous result at the i-th stage of multiplication. Denoting the residue of x modulo $2^{rm}-1$ by $R_{rm}(x)$, (5.3.8) can be written as

$$R_{rm}(Y_i) = R_{rm}[R_{rm}(AMn_{i-1}) + E_{pi}] \qquad (5.3.9)$$

If t or less iterative errors occur in the multiplication, then from (5.3.3) the weight of E_{pi} is less than or equal to t. Since $A = (2^{rm}-1)/(2^s-1) = 2^{(d-1)s} + \dots + 2^s + 1$ and $ds = rm$, the residue of the integer AMn_{i-1} modulo $2^{rm}-1$, i.e. $R_{rm}(AMn_{i-1})$ contains d indentical blocks in its binary representation form, with each block of length s. From (5.3.9), it is seen that $0 \leq |E_{pi}| < 2^{rm}-1$, $W(E_{pi}) \leq t$; hence the majority-logic decoding scheme developed in Chapter 4 is applicable to the residue $R_{rm}(Y_i)$. A majority decision on the binary bits of $R_{rm}(Y_i)$ can correctly yield the partial error pattern E_{pi} provided that $d \geq 2t + 1$. The same majority-logic decoding scheme can be applied to all $R_{rm}(Y_i)$ for $i = 1, 2, \dots, r$, and all the partial errors $E_{p1}, E_{p2}, \dots E_{pr}$ can be determined.

The design of the above code is based on the assumption that both the multiplicand and the multiplier have rm bits in their binary forms. As a matter of fact, this result can be easily generalized to any case when the multiplicand and the multiplier are of different length, say, r_1m bits for the multiplicand and r_2m bits for the multiplier. In these cases, the generator of the code should be modified as

$$A' = 2^{(d-1)s} + 2^{(d-2)s} + \dots + 2^s + 1$$

where $ds = r_1m$, and the length of the code is $(r_1+r_2)m$. The generator A' may not be a divisor of $2^{(r_1+r_2)m}-1$ and the code may not be cyclic. Nevertheless, the error correcting procedure of the code is still the same.

Also, it should be remarked that the code devised above can correct iterative errors caused by any possible logic faults (not only the stuck-at type of fault) in the circuitry of a high speed multiplier. The erroneous bits of a correctable iterative error pattern are not restricted to have the same polarity.

Rate of Code

The information rate (or the efficiency) of an arithmetic code is defined as

$$\text{Rate} = 1 - [(\text{number of bits in the binary form of A})/\text{length of code}] \tag{5.3.10}$$

For the code of length 2rm, generated by $A = (2^{rm}-1)/(2^{s}-1)$, where $ds = rm$, the rate can be readily calculated as

$$\text{Rate (t-iterative-error-correcting code)}$$
$$= 1 - \frac{(d-1)s}{2rm}$$
$$= \frac{rm + (rm-ds+s)}{2rm}$$
$$= \frac{1}{2} + \frac{s}{2rm} \tag{5.3.11}$$

It is interesting to see that the rate of any t-iterative error correcting code is greater than 0.5. For $t = 1$ (single-iterative-error), we take $d = 2t + 1 = 3$. Then $3s = rm$ and the rate is

Rate (single-iterative-error-correcting code)

$$= \frac{1}{2} + \frac{s}{2rm} = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} = \frac{2}{3}$$

Therefore, the rate of any t-iterative-error-correcting code of length 2rm lies in the range

$$\frac{1}{2} + \frac{1}{rm} \leq \text{rate (t-iterative-error code)} \leq \frac{2}{3} \qquad (5.3.12)$$

For different-length multiplicand and multiplier, there may exist some t-iterative-error-correcting codes with efficiency even better than 2/3.

From the above discussion, we see that the construction of the code is simple, the efficiency of the code is satisfactory and more importantly, the decoding can be achieved by using simple one-level majority logic. On the other hand, since the error correcting is done at each stage of the multiplication when a partial sum is obtained, there is definitely a time delay in the speed of multiplication.

6. CONCLUSIONS

6.1 Conclusions

The goals of this thesis have been to investigate the two unsolved problems in the theory of arithmetic coding mentioned in the introductory chapter, to develop new checking techniques for computer arithmetic and to examine the theoretical structure of the arithmetic codes. For the code construction problem, several new classes of multiple-error-correcting arithmetic codes are found. By using some interesting properties of the binary representation forms of certain integers the error control capability of the codes can be determined analytically. The synthesis of these classes of codes are simple, in fact, for any two relatively prime integers n_1 and n_2, the generator of the code is chosen to be $(2^{n_1n_2}-1)/(2^{n_1}-1)(2^{n_2}-1)$. The fact that p_1p_2-code, which is a subclass of the codes constructed, possesses a root-distance relationship similar to that of the BCH bound indicates additional analogy between arithmetic AN-codes and cyclic parity-check codes.

For the decoding implementation problem, the majority-logic decoding scheme for arithmetic codes is devised. This is a completely different approach from the attempts made by using various search techniques [29,30], the implementation of the codes can be achieved by using majority logic alone. It has been shown that the arithmetic AN-codes constructed systematically in this thesis are majority-logic decodable. The majority-logic decoding scheme is applied to other classes of AN-codes and a generalized multi-step decoding algorithm is obtained. The number of steps of majority-logic decision required in a multi-step decoding can be shown to equal the number of distinct prime factors conatained in the block length of the code. It has also been noticed that the minimum distance of these majority-logic decodable codes can all be determined exactly.

A new checking technique for the binary adders has been developed. This separate checking technique provides a more reliable arithmetic error checking model, the redundant circuits used are less than that required for a triplicated system. Since the arithmetic coding scheme is used separately for each shorter adding unit, the single-error assumption is made and the decoding is much simpler. Furthermore, the application of majority-logic decodable arithmetic AN-codes to the error correction in high speed multiplier has been examined.

Some theoretical results about the structure of arithmetic AN-codes have been obtained. By using the concepts of number theory, the upper and lower bounds on any cyclic AN-code have been formulated; also, a fast algorithm for finding the binary representation forms of certain integers has been derived. In summary, the results presented in this thesis gives a better belief that the use of arithmetic coding for the error control in digital computing systems is indeed very promising.

6.2 Suggestion for Future Work

It seems that the next major advance in arithmetic coding from the theoretical viewpoint will be the complete formulation of the class of cyclic AN-codes analgous to the Bose-Chaudhuri-Hocquenghem parity-check codes. When this problem is solved, the computer designer interested in incorporating coding for arithmetic error control will have an easy way of constructing codes. From practical point of view, the separate checking technique discussed in this thesis will find its way into actual computer hardware designed to operate reliably either faster or for a longer time than non-coded equipment. How to systemactically choose the various length of the independent checking unit so that the redundancy is minimized would be a problem which requires further examination.

The implementation of decoder by using majority-logic increases the possibility of widespread application of arithmetic coding. Future work will be the discovery of other more efficient majority-logic decodable arithmetic AN-codes and other practical decoding methods based on the concepts of majority-logic decision.

REFERENCES

[1] Avizienis, A., "Concurrent Diagnosis of Arithmetic Processors," Digest First Annual IEEE Computer Conference, 34-37, (September, 1970).

[2] Avizienis, A., "A Set of Algorithms for a Diagnostic Arithmetic Unit," Jet Propulsion Laboratory, Pasadena, California, Technical Report 32-546 (March, 1964).

[3] Avizienis, A., " A Study of the Effectiveness of Fault-Detecting Codes for Binary Arithmetic," Jet Propulsion Laboratory, Pasadena, California, Technical Report 32-711 (September, 1965).

[4] Avizienis, A., "Design of Fault-Tolerant Computers," AFIPS Conference Proceedings 31, 733-743 (1967).

[5] Barrows, J. T., Jr., "A New Method for Constructing Multiple Error Correcting Linear Residue Codes," Report R-277, Coordinated Science Laboratory, University of Illinois, Urbana (January, 1966).

[6] Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York (1968).

[7] Bernstein, A. J., and W. H. Kim, "Linear Codes for Single Error Correction in Symmetric and Asymmetric Computational Processes," IEEE Trans. Information Theory IT-8, 29-34 (January, 1962).

[8] Brown, D. T. "Error Detecting and Error Correcting Binary Codes for Arithmetic Operations," IRE Trans. Electronic Computers EC-9, 333-337, (1960).

[9] Berger, J. M. "A Note on Error Dectection Codes for Asymmetric Channels," Information and Control, Vol. 4, pp 68-73, (March, 1961).

[10] Chang, H. Y., E. Manning, and G. Metze, Fault Diagnosis of Digital Systems, Wiley-Interscience, New York, (1970).

[11] Chang, S. H., and N. Tsao-Wu, "Distance and Structure of Cyclic Arithmetic Codes," Proc. Hawaii International Conference on System Sciences, 463-466, (1968).

[12] Chang, S. H., and N. Tsao-Wu, "Discussions on Arithmetic Codes with Large Distance," IEEE Trans. Information Theory IT-14, 174-176 (January, 1968).

[13] Chiang, A. C. L., and I. S. Reed, "Arithmetic Norms and Bounds of the Arithmetic AN Codes," IEEE Trans. Information Theory IT-16, 470-476 (July, 1970).

[14] Chien, R. T., "On Linear Residue Codes for Burst Error Correction," IEEE Trans. Information Theory IT-10, 170 (April, 1964).

[15] Chien, R. T., and S. J. Hong, "On a Root-Distance Relation for Arithmetic Codes," Coordinated Science Laboratory, University of Illinois, Urbana, Memorandum (1970). Submitted to IEEE Trans. Information Theory.

[16] Chien, R. T., and S. J. Hor "Error Correction in High Speed Arithmetic," IEEE Trans. Computers, (May, 1972).

[17] Chien, R. T. S. J. Hong, and F. P. Preparata, "Some Contributions to the Theory of Arithmetic Codes," Proc. of Hawaii International Conference on System Sciences, 460-462 (1968).

[18] Chien, R. T., S. J. Hong, and F. P. Preparata, "Some Results in the Theory of Arithmetic Codes," Coordinated Science Laboratory, University of Illinois, Urbana, Report R-440 (October, 1969). To appear in Information and Control.

[19] Dadayev, Y. G., "Arithmetic Divisible Codes with Correction for Independent Errors," Engineering Cybernetics (translation of Tekhnicheskaya Kibernetika) (6), (1965).

[20] Diamond, J. M., "Checking Codes for Digital Computers," Proc. IRE 43, 487-488 (April, 1955).

[21] Dickson, L. E., "On the Cyclotomic Function," American Math. Monthly, Vol. 12, pp. 86-89 (1905).

[22] Elisa, P., "Computation in the Presence of Noise," IBM J. Res. Dev. 2, 346-353 (October, 1958).

[23] Erosh, I. L. and S. L. Erosh, "Arithmetic Codes with Correction of Multiple Errors," Problemy Peredachi Information, Vol. 3, pp. 72-80, (1968).

[24] Garcia, O. N. and T. R. N. Rao, "On the Methods of Checking Logical Operations," Proc. Second Annual Princeton Conference on Information Science and Systems (March, 1968).

[25] Garcia, O. N., "Error Correction in Parallel Binary Adders," Proc. Southeastern Symposium on System Theory, Virginia Polytechnic Institute, Blacksburg, Virginia (May, 1969).

[26] Garner, H. L., "Arithmetic Error Correction," NASA Electronics Research Center, Cambridge, Mass., Spaceborne Multiprocessor Seminar, (October, 1966).

[27] Goto, M. and T. Fukumura, "The Distance of Arithmetic Codes," Memoirs of the Faculty of Engineering, Nagova University, Japan, 20, (2) 474-482, (November, 1968).

[28] Henderson, D. S., "Residue Class Error Checking Codes," Proc. Sixteenth National Meeting of the Association for Computing Machinery, Los Angles, (September, 1961). Abstract appears in Comm. of ACM 4, 307 (July,1961).

[29] Hong, S. J., "On Bounds and Implementation of Arithmetic Codes," Proc. National Electronics Conference 25, 292-296 (December, 1969).

[30] E. R. and N. T. Tsao-Wu, "The Decoding of Arithmetic Cyclic Codes by Permutations of Residue," Proceedings of Princeton Conference on Information and System, (1969).

[31] Lehmer, E. "On the Magnitude of the Coefficients of the Cyclotomic Polynomial," Bull. Am. Math. Soc., Vol. 42, 389-392, (1936).

[32] Mandelbaum, D., "Arithmetic Codes with Large Distance," IEEE Trans. Information Theory IT-13, 237-242, (April, 1967).

[33] Mandelbaum, D., "Multivalued Arithmetic Burst Error Codes," IEEE International Covention Record 14(7), 54-59, (March, 1966).

[34] Massey, J. L., Threshold Decoding, M.I.T. Press, Cambridge, Massachusetts, (1963).

[35] Massey, J. L., "Survey of Residue Coding for Arithmetic Errors," International Computation Center Bulletin 3, UNESCO, Rome, Italy, 3-17 (October, 1964).

[36] Massey, J. L, and O. N. Garcia, "Error-Correcting Codes in Computer Arithmetic," Advances in Information Systems Science, Vol. 4, Plenum Press New York, (1972).

[37] Peterson, W. W., Error-Correcting Codes, M.I.T. Press and Wiley, Cambridge, Massachusetts, and New York (1961).

[38] Peterson, W. W., "On Checking an Adder," IBM J. Res. Dev. 2, 166-168 (April, 1958).

[39] Preparata, F. P., "On the Representation of Integers in Nonadjacent Form", SIAM Journal of Applied Mathematics, Vol. 21, No. 3 (December, 1971).

[40] Rao, T. R. N., and O. N. Garcia, "Cyclic and Milti-Residue Codes for Arithmetic Operations," IEEE Trans. Information Theory, (January, 1971).

[41] Rao, T. R. N., "Bi-Residue Error Correcting Codes for Computer Arithmetic," IEEE Trans. Computers C-19, (May, 1970).

[42] Rao, T. R. N. and A. K. Trehan, "Single-Error-Correcting Nonbinary Arithmetic Codes," IEEE Trans. Information Theory IT-16, 604-608 (September, 1970).

[43] Reitwiesner, G. H., "Advances in Computers" (F. L. Alt, ed.) Vol. 1. 232-308, Academic Press, New York (1960).

[44] Rothstein, J., "Residues of Binary Numbers Modulo 3," IRE Trans. Electronic Computers EC-8, 299 (June, 1959).

[45] Sellers, F. F., M. Y. Hsiao, and L. W. Bearson, Error Detecting Logic for Digital Computers, McGraw-Hill, New York, (1968).

[46] Stein, J. J., "Prime Residue Error Correcting Codes," IEEE Trans. Information Theory IT-10, 170 (April, 1964).

[47] Szabo, N. S. and R. I. Tanaka, Residue Arithmetic and Its Applications to Computer Technology, McGraw-Hill, New York (1967).

[48] Vandel Waerden, Modern Algebra, 3rd Printing, Frederick Ungar Publishing Company, New York (1964).

[49] Zuckerman, H. S. and I. Niven, An Introduction to the Theory of Numbers, 2nd edition, John Wiley and Sons, Inc. New York, (1968).

VITA

Chao-Kai Liu was born on June 14, 1946 in Kansu, China. He received the B.S. in electrical engineering from National Taiwan University, Taipei, Republic of China, in June, 1967.

He began his graduate studies in electrical engineering at the University of Waterloo, Waterloo, Ontario in October, 1968, and received the M.A.Sc. in September 1969. He continued his graduate studies at the University of Illinois, Urbana, Illinois. Since September, 1969 he has been a research assistant in the Coordinated Science Laboratory.