VERSATILE ERROR-CONTROL CODING SYSTEMS

by

VALDEMAR C. ROCHA, Jr.

A Dissertation submitted for the Degree of Doctor of Philosophy in the Faculty of Natural Sciences at the University of Kent at Canterbury

Electronics Laboratories *May, 1976*

BLL Microfilm No: D19629/77

F 75268

To my wife,

Ursula

Acknowledgements

I would like to give my most grateful thanks to the Brazilian research coordinating institution, CAPES, for the financial support which made this research possible; to Professor R.C. Jennison for providing research facilities; to the Staff of the Electronics Laboratories for friendly assistance; to my colleagues in both the Communications Laboratory and the Computers & Cybernetics Laboratory for useful advice, discussion of relevant matters and sharing experimental equipment; to Miss Denise Paine for her efforts in the typing of this thesis; to Dr. P.G. Farrell for his helpful supervision, guidance, encouragement and the constant interest demonstrated during the course of this research; to my family, friends and especially to my wife for encouragement, patience and understanding.

ABSTRACT

The research reported in this thesis is in the field of error-correcting codes, which has evolved as a very important branch of information theory. The main use of error-correcting codes is to increase the reliability of digital data transmitted through a noisy environment. There are, sometimes, alternative ways of increasing the reliability of data transmission, but coding methods are now competitive in cost and complexity in many cases because of recent advances in technology.

The first two chapters of this thesis introduce the subject of error-correcting codes, review some of the published literature in this field and discuss the advantages of various coding techniques. After presenting linear block codes attention is from then on concentrated on cyclic codes, which is the subject of Chapter 3.

The first part of Chapter 3 presents the mathematical background necessary for the study of cyclic codes and examines existing methods of encoding and their practical implementation. In the second part of Chapter 3 various ways of decoding cyclic codes are studied and from these considerations, a general decoder for cyclic codes is devised and is presented in Chapter 4. Also, a review of the principal classes of cyclic codes is presented.

Chapter 4 describes an experimental system constructed for measuring the performance of cyclic codes initially

perturbed by random errors and then by bursts of errors. Simulated channels are used both for random and burst errors. A computer simulation of the whole system was made in order to verify the accuracy of the experimental results obtained.

Chapter 5 presents the various results obtained with the experimental system and by computer simulation, which allow a comparison of the efficiency of various cyclic codes to be made. Finally, Chapter 6 summarises and discusses the main results of the research and suggests interesting points for future investigation in the area.

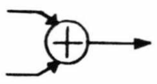
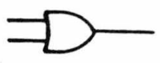
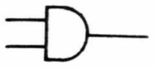
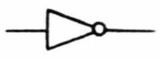
The main objective of this research is to contribute towards the solution of a fairly wide range of problems arising in the design of efficient coding schemes for practical applications; i.e. a study of coding from an engineering point of view.

CONTENTS

Symbols and Abbreviations

α	Primitive element of a Galois Field
A.R.Q.	Automatic repeat request
b	Maximum burst length that a code can effectively correct
B.C.H.	Bose-Chaudhuri-Hocquenghem
B.S.C.	Binary Symmetric Channel
c	Number of parity check digits in a block code
C	Channel capacity
C(x)	Remainder polynomial
CLC	Combinational logic circuit
$C_n^t = \frac{n!}{(n-t)!t!}$	
D	Unit delay Number of errors a code can detect
d	Hamming distance
$[E]$	Noise error pattern
E(x)	Polynomial representation of an error pattern
EB	Error buffer
EC	Error correcting code
ED	Error detecting code
ET	Error trapping
F/B	Feedback
FF	Flip-flop
FEC	Forward error correction
FSR	Feedback shift register
$[G]$	Generator matrix of a linear code

$G(x)$	Generator polynomial of a cyclic code
$GF(q)$	Galois field with q elements
$[H]$	Parity check matrix
$H(x)$	Parity check polynomial of a cyclic code
I/P	Input
$I(x)$	Polynomial representation of k information digits
J	Number of orthogonal parity check equations
k	Number of information digits in a block code
LCM	Least common multiple
MPLX	Multiplexer
n	Block length of a code
(n,k)	Block code with parameters n and k
(n,k,d)	Block code with parameters n, k and d
O/P	Output
p	Exponent to which a polynomial belongs
$P(x)$	Primitive polynomial Quotient polynomial
P_c	Probability of erroneous decoding
P_b	Probability of a burst occurring
P_e	Probability of a random error occurring
PN	Pseudo-noise
PROM	Programmable Read Only memory
q	Number of levels that a digit can assume
$Q(x)$	Quotient polynomial
$[R]$	Received codeword
R	Code information rate or efficiency
$R(X)$	Polynomial representation of a received n-tuple
REM	Remainder (from a division)

$[S]$	Syndrome vector
$S(x)$	Polynomial representation of a syndrome vector
S.C.R.	Syndrome calculating register
S/N	Signal-to-noise power ratio
S.P.C.	Single parity check
S.R.	Shift register
t	Number of random errors a code can correct (per codeword)
$[V]$	Transmitted codeword
$V(x)$	Polynomial representation of a transmitted codeword
$\|X\|$	Determinant X
$\lceil X \rceil$	Largest integer $\leq$ X
$[X]$	Matrix X
$[X]^T$	Transpose of $[X]$
W	Channel frequency bandwidth
$W(v)$	Weight of an n-tuple v
W/F	Waveform
	Two input exclusive-OR gate
	Two input OR-gate
	Two input AND-gate
	Logical inverter

CHAPTER 1

Introduction

People in their desire to communicate with others have, since the very early days of history, been challenged by the problem of how to achieve reliable communication, i.e. reliable transmission of information. Though the formulation of this problem has varied a lot over the centuries, it remains very much alive and is of major concern among the communication engineers of today.

Practical communication systems keep changing their external aspect as technology changes, e.g. the old systems evolved from electromechanical switching relays to vacuum tubes, later to transistors, etc. However, a closer look reveals that in general terms these systems can all be represented by a block diagram like that of Figure 1.1.

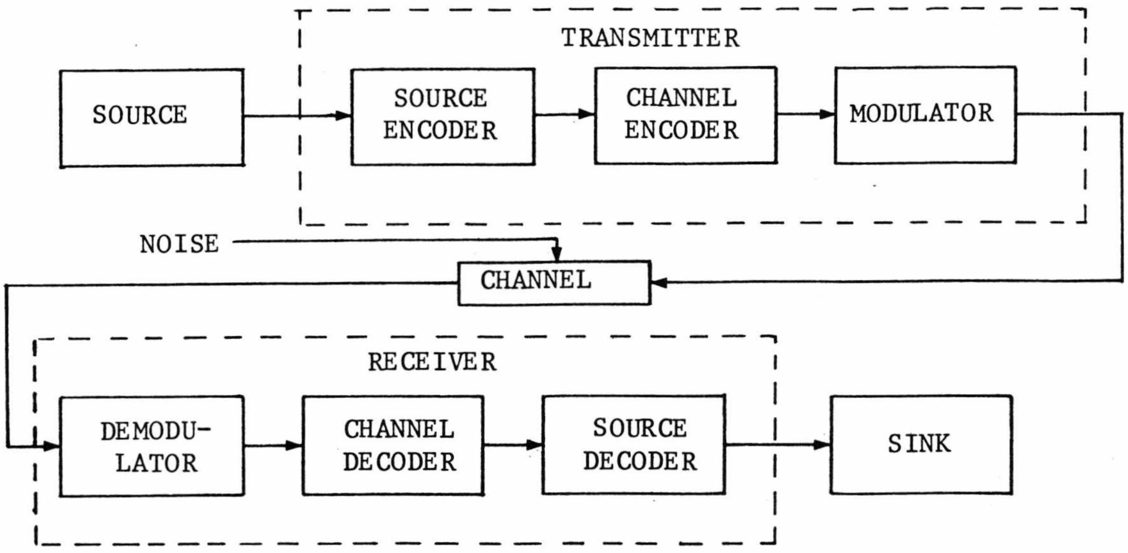


FIGURE 1.1 : General Communication System

SOURCE : Originating point for the information to be transmitted. It can be, for example, the output from a remote sensor in a telemetering system.

TRANSMITTER : Converts the source output into waveforms suitable for transmission over the channel. The function of the transmitter can be subdivided as follows:

1. SOURCE ENCODER

This can be in many cases simply an analogue to digital converter or in other situations a more sophisticated piece of equipment to remove unnecessary detail from the information, as in data compression schemes.

2. CHANNEL ENCODER Controlled redundancy is added to the information by the channel encoder to counteract as far as possible the effects of noise.

3. MODULATOR

In many cases the output from the channel encoder is not matched to the channel. It is the function of the modulator to translate the channel encoder output into a waveform suitable for transmission over the channel.

CHANNEL : Physical path over which the information has to pass before reaching the receiver. A channel can take a variety of forms. Pair of wires, or microwave links, are examples of practical channels. As the information is carried through a channel, it is subjected to unpredictable and unwanted disturbances called noise. As a result of noise part of the information can be badly corrupted.

RECEIVER: Estimates which waveform has been transmitted from the received version, possibly corrupted by noise, of the original waveform. This is normally the most complex part of a communication system and can be subdivided as follows:

1. DEMODULATOR

On receiving a waveform from the channel the demodulator tries to estimate which waveform was sent by the transmitter and outputs the corresponding digital version. Due to noise, this version will not always be a correct one and so estimates containing errors will be passed on to the channel decoder.

2. CHANNEL DECODER

By applying the coding rules to the digits provided by the demodulator, the channel decoder tries to correct possible errors and then produce its estimation of the source encoder output digits.

3. SOURCE DECODER

Processes, to replace the redundancy removed at the transmitter, the channel decoder output before passing it to the sink. If a correct estimate of the transmitted message has been made by the channel decoder, then the source decoder will supply a replica of the original information to the sink.

SINK Final recipient of the information transmitted. The sink can be a human being at the end of a telephone line, or a computer, for example.

No matter how well designed, any communication system will always suffer from the effects of noise, i.e. the messages coming from its output may contain errors. It can take a long time before an error appears, but eventually it will happen. However, the practical problem is not the provision of error-free communication, but the design of systems which provide an acceptably low error rate for the user. For example, an error rate of 10^{-4} for the letters in a book is perfectly acceptable, while the same error rate within a computer would be disastrous.

The ultimate potential of error-correcting codes was established in 1948 with Shannon's "Coding Theorem" for a noisy channel (Shannon, 1948). The coding theorem for a noisy channel states the following:

- every channel has a definite maximum capacity C, and for any rate R less than C, there exist codes of rate R which, with maximum likelihood decoding, have an arbitrarily small probability of erroneous decoding.

This means more specifically that for any given $R<C$ and length n, there exists a code such that $P_c \leq e^{-nE(R)}$* where $E(R) > 0$, for $R<C$, and is specified by channel transition probabilities (Lin, 1970). The coding theorem proves the existence of codes which can make the probability of erroneous decoding very small but gives no clue on how to construct such codes. However, it indicates that P_c can be

* P_c is the probability of erroneous decoding.

reduced by increasing n. When n is increased, the complexity of the systemsincrease, sometimes exponentially with n. The ways by which the error rate of a system can be reduced are all dependent on the parameters contained in the famous expression below, where W represents the channel bandwidth and S/N is the signal-to-noise power ratio

$$C = W \log(1 + S/N)$$

derived by Shannon (1948). The most ingenious of all is the trading of bandwidth and signal-to-noise power ratio by the use of coding (i.e. the controlled addition of redundant information) to allow the receiver to detect and possibly correct errors using a decoder.

From this discussion, it becomes apparent that the objective of coding theory is to:

(1) find long good codes
(2) find practical methods for encoding and efficient decoding

The present need for processing enormous amounts of data, mostly digital, transmitted in many cases automatically and at high-speed, demands solutions that a few years ago would be considered impractical. The recent developments in digital hardware technology have made possible the use of fairly complex coding schemes and as more sophisticated processors become available, thanks to microelectronic technology, the advantages that can be gained by the use of coding will be even greater.

The second chapter in this thesis reviews some of the published literature in the field and discusses the advantages of various coding techniques. After presenting linear block codes, attention is from then on concentrated on cyclic codes, which is the subject of Chapter 3.

The first part of Chapter 3 presents the mathematical background necessary for the study of cyclic codes and examines existing methods of encoding and their practical implementation. In the second part of Chapter 3 various ways of decoding cyclic codes are studied and from these considerations a general decoder for cyclic codes is devised and is presented in Chapter 4. Also a review of the principal classes of cyclic codes is presented.

Chapter 4 describes an experimental system constructed for measuring the performance of cyclic codes initially perturbed by random errors and then by bursts of errors. Simulated channels are used both for random and burst errors. A computer simulation of the whole system was made in order to verify the accuracy of the experimental results obtained.

Chapter 5 presents the various results obtained with the experimental system and by computer simulation which allow a comparison of the efficiency of various cyclic codes to be made. Finally, Chapter 6 summarises and discusses the main results of the research and suggests interesting points for future investigation in the area.

The main objective of this research is to contribute towards the solution of a fairly wide range of problems arising in the design of efficient coding schemes for practical applications; i.e. a study of coding from an engineering point of view.

CHAPTER 2

Survey of Coding

2.1 Introduction

Since the advent of the coding theorem (Shannon, 1948) and of Hamming's Single-Error-Correcting Code (Hamming, 1950), coding theory has developed enormously, and at present it generates a fairly extensive area of research. Apart from the more general applications, there are codes designed for special applications like synchronisation recovery (Stiffler, 1971), asymmetric channels (Kautz and Levitt, 1969), comma-free codes (Scholtz, 1969), etc., but these are not studied here. After a brief discussion of the two main types of code construction this survey is directed towards linear binary group codes. By comparison with other types of codes, linear codes are fairly well developed and understood.

2.2 Block and Convolutional Codes

Depending on how redundancy is added to blocks of information digits two basically different types of code result. Codes for which the redundancy in a block checks for errors only in that particular block are called block codes. Codes where the redundancy in a block checks for errors in more than one block are called convolutional codes (Elias, 1955). There is no intention here of considering convolutional codes any further; their definition has been given just to point out the basic difference between block and convolutional codes. For a treatment of convolutional codes see Peterson (1972). Block and convolutional codes are competitive in many situations. The final choice between them is a function of factors like the data format, the decoding delay, the complexity of the overall system needed to achieve a specified output error rate, etc.

2.3 Linear Block Codes

Block codes can be linear or nonlinear. For linear codes the redundant digits are calculated with modulo-2 adders while nonlinear codes require the use of nonlinear logic like AND, NOR, NAND gates, etc. However, the overwhelming majority of published articles on block codes is concerned with the linear case. The reason for this is the fact that linear block codes turn out to be mathematically more tractable, and in general are simpler to implement in practice, than nonlinear block codes. Despite these difficulties there is still research being done on nonlinear codes, and discoveries like that of a (15,8) code which corrects two random errors per block (Nordstrom et al,1967) which it is impossible to obtain by linear means, stimulates further work in nonlinear code construction techniques. The theory of linear block codes owes much to the work of

Hamming (1950), Golay (1949) and Slepian (1956a, 1956b, 1960). The treatment of linear block codes that follows uses the concept of vector space (see Appendix I for definition of terms), and only binary codes are assumed throughout unless the contrary is specified. Linear block codes are normally represented by the ordered pair (n,k) where n represents the number of digits in each codeword and is referred to as block length, and k is the number of information digits per block; or the ordered triplet (n,k,d) where n and k are the same as before and d is the code minimum distance (to be defined later in this chapter).

DEFINITION 2.1 An (n,k) linear block code is a set of 2^k n-tuples which form a subspace of the vector space of all n-tuples.

2.4 Generator Matrix

An (n,k) binary code has 2^k distinct codewords, each of them n digits long. To use such a code, without further consideration, it is necessary to store $n \times 2^k$ binary digits (bits) at the transmitter. This is one way of having the 2^k codewords ready for transmission. However, when the 2^k n-tuples form a subspace of the space of all n-tuples (i.e. a linear code) it is possible to obtain a set of k linearly independent vectors*, which by linear combinations generate all the elements of the subspace. For example if:

*The words vector and codeword will be used interchangeably throughout this thesis.

$$[V_1],[V_2], \ldots, [V_k]$$

are k independent n-tuples (i.e. form a basis) then any other n-tuple in their subspace can be obtained as:

$$[U] = m_1[V_1] + m_2[V_2] + \ldots + m_k[V_k]$$

where m_i is either zero or one, and $1 \leq i \leq k$.
Then, the way to generate the 2^k codewords for a linear block code is best described in terms of a generator matrix $[G]$. The rows of the generator matrix are chosen to be k independent vectors from the code alphabet; i.e. $[G]$ is a k×n matrix where the rows form a basis:

$$[G] = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_k \end{bmatrix}$$

Any codeword in the code can be generated as follows. Let $[m] = [m_1, m_2, \ldots, m_k]$ be a message sequence. The matrix product $[m].[G]$ results in a vector $[U]$, which is a linear combination of rows of $[G]$:

$$[U] = [m].[G] = m_1[V_1] + m_2[V_2] + \ldots + m_k[V_k]$$

$[U]$ is the codeword associated with the k-tuple message block $[m]$ and the matrix $[G]$ is called the generator matrix of the code. It should be clear that the use of linear block codes reduces considerably the storage requirements at the transmitter. Basically, the encoder for a linear block code can consist of enough storage elements to accommodate the k rows of $[G]$ and a logic circuit to perform a linear combination of these rows according to the incoming message sequence. Since $[G]$ is non-singular (Peterson, 1961), it is possible to write $[G] = [I_k \vdots g]$ where I_k is a k×k unit matrix and g is a k×(n-k) matrix. In this situation $[G]$ is said to be in standard echelon form and the codewords it generates have the first k positions occupied by the information digits while the last n-k digits are linear combinations of these information digits. A code with this structure is said to be <u>systematic</u>. The n-k redundant digits of a codeword are called parity checks and the linear functions that give the parity checks are called parity check equations.

2.5 <u>Parity Check Matrix</u>

Given the k×n matrix $[G]$ of a linear code it is possible to find a (n-k)×n matrix $[H]$ such that the row space of $[G]$ is orthogonal to $[H]$, i.e. if $[V_i]$ is a vector in the row space of $[G]$ then $[V_i].[H]^T = 0$. The $[H]$ matrix is called the parity check matrix of the code and can be represented as $[H] = [h \vdots I_{n-k}]$ where h is an (n-k)×k matrix and I_{n-k} is an (n-k)×(n-k) unit matrix. It can be shown that $[h] = [g]^T$ where $[g]^T$ is the transpose of the $[g]$ matrix. Since the rows of $[H]$ are linearly independent they generate an (n,n-k) linear code which is called the dual of the (n,k) linear code generated by $[G]$. This (n,n-k) code can be regarded as the null-space of the (n,k) code generated by $[G]$.

2.6 <u>Error Syndrome and Decoding</u>

Suppose a codeword $[V]$ in a linear block code with generator matrix $[G]$ and parity check matrix $[H]$ is transmitted through a noisy channel. At the receiver an n-tuple $[R]$ is received which may differ from $[V]$ due to the noise added in the channel during transmission. It is the task of the decoder to recover $[V]$ from $[R]$. The first step is to check whether $[R]$ is a codeword. This step can be represented by the equation below.

$$[R]\,[H]^T = [S]$$

where $[S]$ is an (n-k)-tuple called the syndrome of $[R]$. If $[S] = [0]$ (an all zero (n-k)-tuple) it is assumed that no errors occurred, i.e. $[R]$ is assumed to be equal to $[V]$. However, if $[S] \neq [0]$, $[R]$ does not correspond to a code vector in the row space of $[G]$ and the decoder uses this syndrome for error detection and/or correction purposes. The received n-tuple $[R]$ can be written as $[R] = [V]+[E]$ where $[E]$ is an n-tuple representing the error pattern. A number of relevant terms is presented below which are very useful in establishing the error correcting properties of linear block codes.

<u>DEFINITION</u> 2.2 The number of non-zero components of an n-tuple $[V]$ is called the <u>Hamming weight</u> of $[V]$ and is

denoted by $W_{(V)}$.

<u>DEFINITION</u> 2.3 The number of positions in which two n-tuples $[V_1]$ and $[V_2]$ differ is called the <u>Hamming distance</u> between $[V_1]$ and $[V_2]$ and is denoted by $d_{(V_1,V_2)}$.

<u>DEFINITION</u> 2.4 The smallest distance between any pair of codewords in a code is called the <u>minimum distance</u> of the code, denoted d_{min} or simply d.

Due to the group properties of vector spaces, the addition of two codewords in a linear code gives as a result another codeword. This fact can be represented as:

$$[V_1]+[V_2] = [V_1+V_2] = [V_3]$$

So, $\quad W_{(V_1+V_2)} = W_{(V_3)}$

or $\quad d_{(V_1,V_2)} = W_{(V_3)}$

The last expression above means that for linear codes the minimum distance is equal to the weight of the minimum weight non-zero codeword. With the exception of Hamming codes (d=3 and d=4), which are described later, the problem of constructing non-trivial error correcting codes with a given d is very difficult. In the linear case an important property for code construction is now introduced which relates d with the parity check matrix $[H]$. If the minimum weight non-zero codeword is multiplied by the code parity check matrix $[H]$, the result is obviously an all zero (n-k)-tuple. This all zero (n-k)-tuple can be thought of as resulting from a linear combination of d columns of $[H]$. Consequently, no linear combination of less than d columns of $[H]$ will give as a result an all zero (n-k)-tuple; otherwise the code minimum distance would be less than d. The minimum distance of a linear code can be expressed in terms of the $[H]$ matrix as follows.

THEOREM 2.1 A linear code whose parity check matrix $[H]$ contains d-1 linearly independent columns has minimum distance at least d.

For a formal proof of Theorem 2.1 see Peterson (1972).

Though only the Hamming metric is considered in this thesis, other metrics exist, e.g. the Lee metric (Lee, 1958); the choice of a particular metric, aiming at optimum results, is a function of the type of modulation and channel characteristic to be used (see Berlekamp, 1968).

In order to assess the performance of a coding scheme it is vital to have a knowledge of the statistical behaviour of the channel. In practice, these statistics normally turn out to be very difficult to obtain and a theoretical model of the channel is used instead. One of the most commonly used channel models is that of the binary symmetric channel (B.S.C.). The B.S.C. assumes that errors occur independently (i.e. the channel is memoryless) and that zeros and ones have the same probability of being in error (see Figure 2.1). In some applications it can be more convenient to use other channel models, e.g. the binary erasure channel (Elias, 1954).

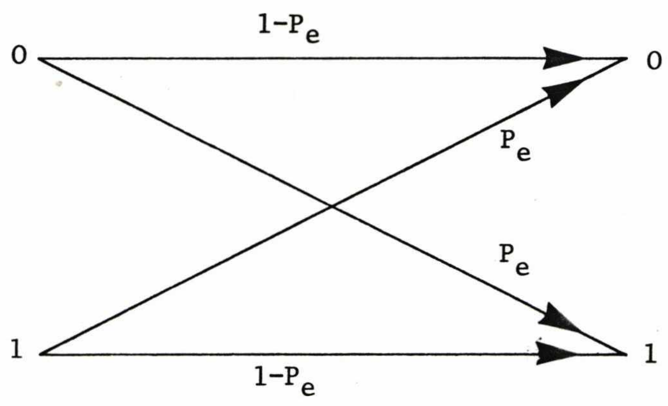


FIGURE 2.1 : Binary Symmetric Channel

The ultimate performance that can be achieved with error correcting codes is theoretically expressed in terms of bounds (Hamming, 1950; Gilbert, 1952; Plotkin, 1960; Varshamov, 1957). There are many classes of codes which meet these bounds for small values of block length, but which soon fall short in performance as n is increased. For a code with minimum distance d to be able to correct t or less errors per codeword, the following inequality must hold:

$d \geq 2\ t+1$ (Peterson, 1961).

In general, to correct C errors and to detect D errors per codeword, the inequality above is expressed as:

$$d \geq C+D+1,$$

where $D \geq C$ as errors must be detected before they can be corrected.

2.7 The Standard Array

When an n-tuple $[R]$ is received, the decoder has to decide among all possible codewords which particular codeword has been transmitted. In order to properly consider this decoding problem, it is sometimes convenient to use the concept of the standard array. This applies to linear codes, and consists of splitting the vector space containing the 2^n distinct n-tuples into 2^k disjoint subsets, each one of them containing one and only one codeword. The 2^k disjoint subsets are constructed as follows. Write all the 2^k code vectors in a row. Below the all zero codeword, write an n-tuple $[E_1]$ which does not appear in the first row. This n-tuple can be associated with an error pattern which the code is to detect and/or correct, as will be shown later. The second row is formed by adding $[E_1]$ to each of the non-zero code vectors as indicated below:

$$\begin{array}{ccccc} [0\ 0\ \ldots\ 0] & [V_1] & [V_2] & \cdots & [V_{2^k-1}] \\ [E_1] & [E_1+V_1] & [E_1+V_2] & \cdots & [E_1+V_{2^k-1}] \end{array}$$

The third and consecutive rows are constructed in a similar manner, every new row starting with an n-tuple not used before. The following table results:

$[0\ 0 \ldots 0]$	$[V_1]$	$[V_2]$	....	$[V_{2^k-1}]$
$[E_1]$	$[E_1+V_1]$	$[E_1+V_2]$	....	$[E_1+V_{2^k-1}]$
$[E_2]$	$[E_2+V_1]$	$[E_2+V_2]$	....	$[E_2+V_{2^k-1}]$
$\vdots$	$\vdots$	$\vdots$		
$[E_{2^{(n-k)}-1}]$	$[E_{2^{(n-k)}-1}+V_1]$		....	$[E_{2^{(n-k)}-1}+V_{2^k-1}]$

TABLE 1

The 2^{n-k} rows in the table above are called cosets and the leading element in each row is called the coset leader. This table is called the standard array or the coset decomposition of the code.

The syndrome $[S]$ associated with an n-tuple $[R]$ is given by:

$$[S] = [R].[H]^T = [E+V].[H]^T = [E].[H]^T + [V].[H]^T = [E].[H]^T$$

since $[V].[H]^T = 0$, i.e. $[S] = [E].[H]^T$. This equation for the syndrome clearly shows that all elements in one coset of the standard array have the same syndrome because it depends only on the coset leader n-tuple $[E]$. This suggests the use of the standard array and the equation $[S] = [E].[H]^T$ to decode linear codes as follows:

(1) Calculate the syndrome $[S]$ of a received n-tuple $[R]$.

(2) Find the coset leader $[E_i]$ associated with this syndrome.

This coset leader is assumed to be the errors introduced by noise on the channel.

(3) Subtract the coset leader found in step 2 above from $[R]$ in order to obtain the estimate of the transmitted codeword, i.e. $[V_i] = [R] - [E_i]$.

From these discussions it is concluded that, based on noise statistics of the channel, the coset leaders should be chosen as the most likely error patterns. However, as mentioned earlier, detailed statistics of the noise are not always available and it usually becomes very difficult to find codes to exactly match the channel. Also, in order to use the standard array, it is necessary to find the coset (and therefore the appropriate coset leader) to which the received n-tuple belongs. This is, in general, not easy to implement, so that the concept of the standard array is more useful as a way of understanding the structure of linear codes, rather than as a practical decoding algorithm.

Two potentially practical methods of decoding linear codes are now presented. Further procedures are described in Peterson (1972) and Lucky et al (1968), but so far the use of the linearity property alone has not resulted in any simple decoding algorithm for linear codes, at least in the published literature.

2.8 Maximum Likelihood Decoding

If the codewords of an (n,k) code are selected independently and all have the same probability of being sent through a channel, an optimum way of decoding them is as follows. On receiving an n-tuple $[R]$, the decoder compares it with all possible codewords in the code. For the binary case this means comparing $[R]$ with the 2^k distinct n-tuples which form the code. Select the codeword which is nearest to $[R]$ in terms of Hamming distance, i.e. the word which differs from $[R]$ in the least number of places. This is assumed to be the transmitted codeword. Unfortunately, the time required to decode a received n-tuple can become prohibitively long even for moderate values of k. It should be noted that the decoder has to compare $[R]$ with 2^k codewords during a time interval corresponding to the duration of n channel digits. This fact makes this process of decoding inadequate for many practical cases.

2.9 Systematic Search Decoding

A general procedure for decoding linear block codes consists of associating a correctable error pattern with each of the non-zero syndromes. It has been mentioned before that one property of the standard array is that all n-tuples belonging to one coset have the same syndrome. Also, the coset leaders should be chosen as the most likely error patterns in their cosets. A simple way of decoding these codes is:

(1) Calculate the syndrome for the received n-tuple.

(2) By systematic search find the correctable error pattern, (coset leader), associated with the syndrome of the received n-tuple.

(3) Subtract the error pattern found in step 2 from the received n-tuple in order to remove the errors from it.

To implement this procedure it is necessary to generate all correctable error patterns successively, and feed them into a combinational circuit that gives the associated syndromes at its output. Using a multiple input logic gate it is possible to detect when the locally generated syndrome matches the syndrome of the received n-tuple. If this (n,k) code is t error correcting, the total number of patterns it is necessary to generate in the search is given by(Lucky et al, 1968):

$$C_n^1 + C_n^2 + \dots + C_n^t = \sum_{i=1}^{t} C_n^i \leq 2^{n-k} - 1$$

From this expression it is easy to see that the number of error patterns increases very rapidly with n and t. This fact sets a limit to the applicability of this technique.

The decoding methods described above in sections 2.8 and 2.9 are generally applicable to all linear block codes. Less complex methods of decoding can be devised for certain classes of linear codes, by virtue of their particular structure. Some simple block codes are now presented.

2.10 Single Parity Check (S.P.C.) Codes

These constitute one of the simplest forms of coding because only one redundant digit (parity check) is used per codeword. This parity check digit is calculated according to the following rule. Make the parity check digit equal to one if the number of <u>ones</u> in the information block is odd; otherwise make the parity check equal to zero. This procedure is equivalent to making the check digit equal to the sum modulo-2 of the information digits in a block, i.e. a linear code results. By using this rule to calculate the parity check, the number of ONES in a codeword is always even*. As a consequence of the presence of the single parity check, the decoder is able to detect any odd number of errors but fails to detect an even number of errors. Thus, these codes have minimum distance d=2. Figure 2.2 shows the diagram of an encoder for these codes. A toggle flip-flop is used to calculate the parity check. The output of this type of flip-flop changes with the next clock pulse whenever the present input is a one. With the flip-flop initially reset, information digits are sent to the channel and simultaneously into the encoder. After k shifts, an odd number of ones in the information section causes a one to appear at the output of the flip-flop; otherwise a zero will be produced. A timing circuit controls the delivery of information and parity digits to the channel.

The decoding rule for S.P.C. codes is simply to count the number of ones in a received block. If the resulting count is even, the received block is assumed to be error-free and can be delivered to the data sink. If the count gives an odd number, errors have been detected and the received block is either tagged and delivered to the data sink or just dis-

---

* An alternative way of calculating the check digit (odd parity) is by making it equal to zero when the number of ones in the information block is odd; otherwise the parity check is made equal to one.

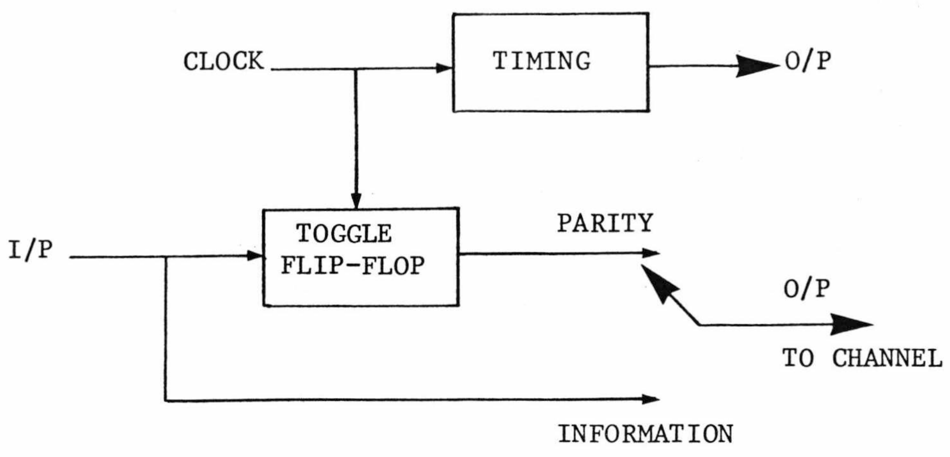


FIGURE 2.2 : Encoder for S.P.C. Codes

carded. Another alternative is to ask the transmitter, assuming a feedback channel is available, to retransmit the erroneous block. This last possibility is called automatic repeat request (A.R.Q.). Though S.P.C. codes permit only error detection, they are very powerful in systems using A.R.Q., because their efficiency is usually high for moderate n and the circuitry required for their implementation is very simple. The efficiency of S.P.C. codes is given by the expression:

$$R = \frac{k}{n} = \frac{n-1}{n} = 1 - \frac{1}{n}$$

It is clear from this expression that R approaches 1 as n is increased. This improved efficiency should always be weighed against the increase in the probability of undetected errors. The block error rate for S.P.C. codes in a B.S.C. channel is given by(Farrell,1969):

$$P_c = C_n^2 \, P_e^2(1-P_e)^{n-2} + C_n^4 \, P_e^4(1-P_e)^{n-4} + \ldots.$$

If P_e is sufficiently small P_c can be approximated as

$$P_c = C_n^2 \, P_e^2(1-P_e)^{n-2} \; .$$

Figure 2.3 shows the diagram of a decoder for S.P.C. codes.

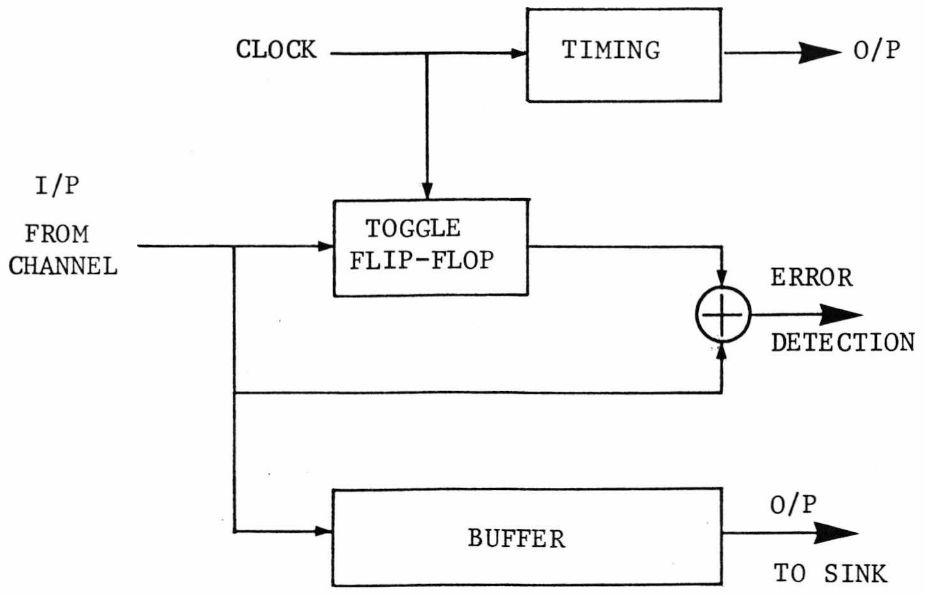


FIGURE 2.3 : Decoder for S.P.C. Codes

2.11 Hamming Codes

These codes (Hamming, 1950), were the first non-trivial error correcting codes to be proposed. Hamming codes are linear single error correcting codes, i.e. their minimum distance is d=3.They have block length $n \leq 2^c-1$, where c is the number of parity check digits. This condition on n ensures the availability of sufficient redundancy to check for single errors in a codeword since then the number of non-zero syndromes (2^c-1) is always greater than or equal to the number of single error positions (n). The construction of the parity check equations for these codes is best explained with the aid of an example.

<u>EXAMPLE</u>. Consider the construction of the (7,4) Hamming code. The ideas described here are easily generalised for any (n,k) Hamming code. The number of check digits for this (7,4) code is c=7-4=3. Consider the non-zero binary numbers that can be formed using c=3 digits.

i.e.	0 0 1	c_1
	0 1 0	c_2
	0 1 1	k_1
	1 0 0	c_3
	1 0 1	k_2
	1 1 0	k_3
	1 1 1	k_4

<u>TABLE 2</u>

Hamming associated the numbers of the form 2^i, i = 0,1,2, ... with parity check positions. The other positions were associated with information digits, as indicated in the table above. Now, looking down the columns, the parity check equations are written as the modulo-2 addition of the information positions where a one appears in the particular column being considered.

$$\text{i.e.}\quad \begin{aligned} c_1 &= k_1 + k_2 + k_4 \\ c_2 &= k_1 + k_3 + k_4 \\ c_3 &= k_2 + k_3 + k_4 \end{aligned}$$

On receiving a codeword, the decoder recalculates the parity checks and adds them modulo-2 to the received ones in order to obtain the syndrome. If, for example, k_3 is in error, parity checks c_2 and c_3 will fail while c_1 will agree because it does not check k_3. This situation is represented as:

$$\begin{array}{ccc} c_3 & c_2 & c_1 \\ 1 & 1 & 0 \end{array}$$

This binary number corresponds to the position of k_3 in the table considered above. The error is thus located and can then be corrected. This method generalises to cover any value of n. For practical reasons the codewords are normally transmitted in a systematic manner. In his work, Hamming (1950) also mentioned that the minimum distance of these codes can be increased by 1 to become d=4, by annexing an overall parity check to each codeword. This overall parity check is determined in the same manner as the one for S.P.C. codes, i.e. by the modulo-2 addition of all other digits in the codeword. This procedure applies not only to Hamming codes but to any code with an odd minimum distance (Peterson, 1972), i.e. if d is odd, by annexing an overall parity check the new minimum distance is d+1. Hamming codes are quite unique in the sense that no other class of non-trivial error correcting codes can be so easily decoded, and also because Hamming codes are perfect or quasi-perfect codes as defined below.

<u>DEFINITION</u> 2.5 An (n,k) t-error-correcting code is called a perfect code if, and only if,

q = number of levels

$$\sum_{i=0}^{t} C_n^i = q^{n-k}$$

Codes in which the information digits are repeated a number of times are called repetition codes and are perfect in a trivial sense. Repetition codes are decoded by taking a majority vote. Apart from the Hamming codes, the (23,12) t=3 Golay code and the ternary (11,6)t=3 code, there are no other non-trivial perfect codes (van Lint, 1970; Tietäväinen, 1973). Codes where

$$\sum_{i=0}^{t+1} C_n^i > q^{n-k} > \sum_{i=0}^{t} C_n^i$$

are called the quasi-perfect if the remaining redundancy can be used to correct only some of the patterns containing t+1 errors. Perfect and quasi-perfect codes are optimum (i.e. they minimise the probability of error) when used in a B.S.C. (Slepian, 1956a).

2.12 Product Codes

These codes are usually linear and result from a combination of two or more codes in order to obtain a more powerful code. When two codes are used to form a product code, the information digits may be arranged in a rectangular matrix form, and row and column parity check digits are calculated in accordance with the coding rules for the two codes respectively, along each of the two dimensions. The minimum distance of this product code is $d = d_1 \times d_2$, where d_1 and d_2 are the individual minimum distances of the two codes used (Elias, 1954). The resulting two dimensional array has the following general structure:

INFORMATION DIGITS	ROW PARITY CHECKS
COLUMN PARITY CHECKS	CHECKS ON CHECKS

Product codes are efficient when used for error detection with A.R.Q. (Farrell, 1969) because a very low probability of undetected error can be achieved. Also they find practical use in coding for digital multiplex systems (Riley, 1975) and in general wherever the data format is rectangular. Product codes have been studied by Elias (1954), Burton and Weldon (1965), Berlekamp (1968), and

Peterson (1972), and others.

2.13 Constant Weight Codes

These codes are also known as: fixed ratio, m out of n and constant ratio codes. Their characteristic is to have codewords all with the same weight i.e., the same number of ONES.

The number of codewords in an m out of n code is given by:

$$M = \frac{n!}{m!(n-m)!}$$

The van Duuren A.R.Q. system (van Duuren, 1961) uses a 3 out of 7 constant weight code and is a good example of the use of such codes on radio-telegraph circuits. In this case any single error causes a received block to have either weight 2 or 4 and so is detected. However, a double error can happen which changes a one into a zero and a zero into a one and this will pass undetected. In general, these codes are guaranteed to detect odd numbers of errors but can fail to detect an even number of them. Constant weight codes are less efficient than S.P.C. codes but can have their error detecting capability improved by carefully reducing the number of codewords. These codes are very often nonlinear, but there are exceptions like the m-sequence codes (see Chapter 3).

The next chapter introduces cyclic codes which are linear codes with some additional mathematical structure. This allows relatively simple decoders to be constructed for codes normally more powerful than the ones presented above.

CHAPTER 3

Cyclic Codes

3.1 Introduction

The codes described in this chapter constitute the most powerful sub-class of linear block codes as far as practical implementation and mathematical structure are concerned. Prange (1957) was the first to study cyclic codes. Subsequently, these codes have been studied in the context of modern algebra using the concepts of Galois fields, (see Peterson, 1972 and Berlekamp, 1968).

3.2 Basic Definitions

Definition 3.1 A linear block code is called a cyclic code if the result of any cyclic permutation of any of its codewords is another valid codeword, i.e. if $\left[V\right] = \left[V_0, V_1, V_2, \ldots, V_{n-1}\right]$ is a codeword, $\left[V^i\right] = \left[V_{n-i}, V_{n-i+1}, \ldots, V_0, V_1, \ldots, V_{n-i-1}\right]$ is also a codeword in the same code, and the indices are reduced modulo-n.

Any n-tuple like $[V]$ above, can be represented in the form of a polynomial V(x) of degree at most n-1 as follows:

$$V(X) = V_0 + V_1x + V_2x^2 + \ldots + V_{n-1}x^{n-1}$$

With the use of the properties of finite fields it can be proved (see Peterson (1961), for example) that all codewords of an (n,k) cyclic code are multiples of a polynomial G(x), of degree n-k, which is unique; and conversely that every polynomial of degree n-1 or less which is a multiple of G(x) must be a codeword. Furthermore, G(x) is called the code generator polynomial, and it divides X^n+1. The mathematical properties just mentioned for cyclic codes are those of a mathematical ideal (see Appendix I for definition). This allows the properties of cyclic codes to be derived from the study of ideals and an equivalent definition of cyclic codes is:

Definition 3.2 Cyclic codes are ideals in the algebra of polynomials modulo X^n+1.

The factorisation of X^n+1 gives as a result: $X^n+1 = (X+\alpha_1)(X+\alpha_2)(X+\alpha_3)\ldots(X+\alpha_n)$, where the roots $\alpha_i (1 \leq i \leq n)$ are elements of an extension field. Each of these n roots can be expressed as a power of α, where α is called a primitive root, i.e. $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{n-1}, \alpha^n=1=\alpha^0$. The lowest degree polynomial with binary coefficients which is a factor of X^n+1 and contains α_i as a root is called the minimum polynomial of α_i. If $n = 2^m-1$ it can be shown that the maximum degree of a minimum polynomial $m_{\alpha_i}(x)$ is m. In the binary case $m^2_{\alpha_i}(x) = m_{\alpha_i}(x^2)$ which implies that if α_j is a root of $m_{\alpha_i}(x)$ so are $\alpha_j^2, \alpha_j^4, \ldots,$ i.e. the minimum polynomial of any even power of α is the same as the minimum polynomial of some odd power of α. Cyclic codes can also be specified in terms of the roots of G(x), in an extension field. If G(x) has non-repeated roots $\alpha_1,\alpha_2,\ldots,\alpha_{n-k}$ then any polynomial V(x) will belong the code if, and only if, $V_{(\alpha_i)}=0$, $1\leq i\leq n-k$.

<u>THEOREM</u> 3.1 If β is a root of a polynomial V(x), then V(x) is divisible by M(x), the minimum polynomial of β.

<u>PROOF</u> Let $V(x) = P(x)M(x)+S(x)$

Then $V(\beta) = P(\beta)M(\beta)+S(\beta)=0$ and,

$S(\beta)=0$, since $V(\beta)=M(\beta)=0$. By the Euclidean division algorithm S(x) is of degree less than M(x) and so it must be zero, (i.e. S(x)=0) because the definition of minimal polynomial guarantees M(x) to be the lowest degree polynomial with β as a root.

Q.E.D.

As a consequence of Theorem 3.1, if the minimum polynomial of α_i is $M_{\alpha_i}(x)$ then V(x) is a codeword in a cyclic code if and only if V(x) is divisible by $M_{\alpha_1}(x),M_{\alpha_2}(x),\ldots,M_{\alpha_{n-k}}(x)$, i.e. V(x) must divide the least common multiple of $M_{\alpha_1}(x),M_{\alpha_2}(x),\ldots,M_{\alpha_{n-k}}(x)$. Therefore, the code generator polynomial G(x) can be written as:

$$G(x) = L\ C\ M\{M_{\alpha_1}(x),M_{\alpha_2}(x),\ldots,M_{\alpha_{n-k}}(x)\}$$

where α_i,($1\leq i\leq n-k$) are the roots of G(x). If a cyclic code is required to have a generator polynomial G(x) with a root α_i repeated p times, then the minimum polynomial of α_i must appear p times as a factor of G(x). The condition for X^n+1 to have only distinct roots is that the block length n and the number of levels q must be relatively prime (Peterson, 1961). In the binary case, this condition simply means that n must be odd for the roots of X^n+1 to be distinct.

3.3 Matrix Representation of Cyclic Codes

From the definition of cyclic codes, the multiples of the generator polynomial G(x) are codewords. So the polynomials $G(x), xG(x), x^2G(x), \ldots, x^{k-1}G(x)$ are all codewords, and are also linearly independent. Using these polynomials the following matrix can be formed, which represents the generator matrix of a cyclic code with generator polynomial G(x).

$$[G] = \begin{bmatrix} x^{k-1}G(x) \\ \vdots \\ x^2G(x) \\ xG(x) \\ G(x) \end{bmatrix}$$

For encoding purposes the cyclic shift property allows a sequential implementation of $[G]$ which is described in section 3.4. Also described there is a sequential implementation of the parity check matrix $[H]$ as a function of the code parity check polynomial H(x). This relatively simple implementation turns out to be of great practical advantage for cyclic codes. Let $V(x) = V_0+V_1x+V_2x^2+\ldots+V_{n-1}x^{n-1}$ be a codeword in a cyclic code with generator polynomial G(x) having roots $\alpha_1,\alpha_2,\ldots,\alpha_{n-k}$, i.e. $V_{(\alpha_i)}=0=V_0+V_1\alpha_i+V_2\alpha_i^2+\ldots+V_{n-1}\alpha_i^{n-1}$, where $1\leq i\leq n-k$. This expression can be written as a matrix product as follows:

$$\left[V_0, V_1, V_2, \ldots, V_{n-1}\right] \cdot \left[1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1}\right]^T = 0$$

or, in other words, V(x) is a codeword if and only if it is in the null space of the matrix $[H]$:

$$[H] = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{n-k} & \alpha_{n-k}^2 & \alpha_{n-k}^{n-1} \end{bmatrix}$$

3.4 Encoding With An (n-k)-Stage Shift-Register

This encoding procedure is based on the property that all codewords in a cyclic code are multiples of the code generator polynomial G(x). The k information digits can be represented by a polynomial I(x) of degree less than k as follows:

$$I(x) = k_1 x^{k-1} + k_2 x^{k-2} + \ldots + k_k$$

where the k_i ($1 \leq i \leq k$) are the information digits. Multiplying the information polynomial I(x) by x^{n-k} gives $x^{n-k}I(x)$ which is of degree not greater than (n-1) and contains no terms of degree less than (n-k). Division of $x^{n-k}I(x)$ by G(x) gives as a result

$$x^{n-k}I(x) = P(x)G(x) + C(x)$$

where P(x) and C(x) are respectively quotient and remainder polynomials. C(x), being the remainder, is of degree smaller than the degree of G(x), i.e. $\leq(n-k)-1$. If C(x) is subtracted from $x^{n-k}I(x)$ the result is a multiple of G(x), i.e. a codeword. Also, since C(x) has **degree** $\leq n-k-1$, it represents the parity checks and does not overlap with the information section represented by $x^{n-k}I(x)$. The operations described can be performed by the circuit shown in Figure 3.1, keeping in mind that modulo-2 addition and subtraction are the same.

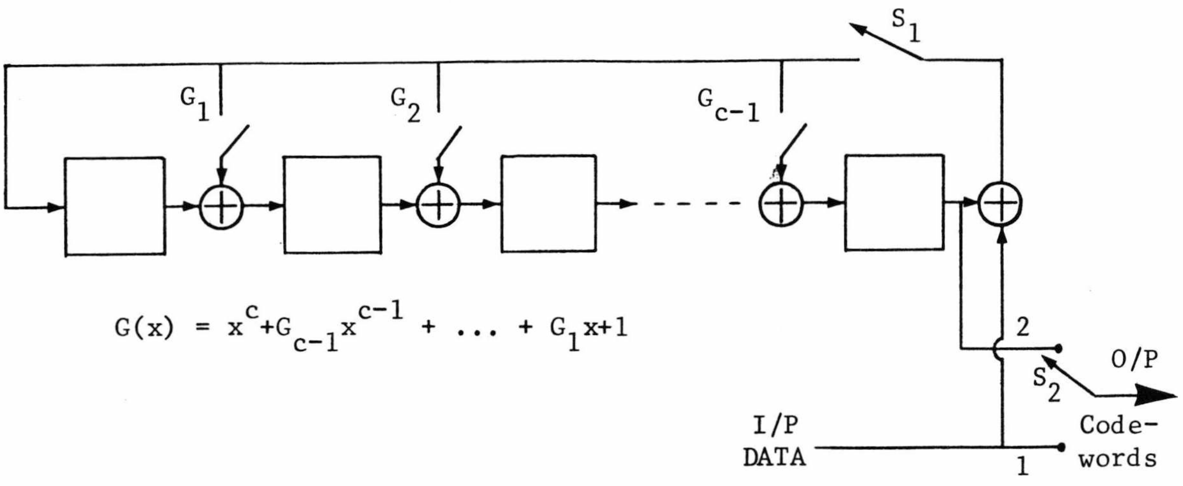


FIGURE 3.1 : Encoder Using (n-k) S.R. Stages

The circuit shown in Figure 3.1 uses (n-k)-stages of shift-register and pre-multiplies the information polynomial by x^{n-k} (Peterson, 1972). The switches $G_1, G_2, \dots, G_{c-1}$ are closed wherever the corresponding coefficient in G(x) is a ONE; otherwise they are left open. Starting with all zeros in the shift register, switches S_1 closed and S_2 in position 1, information digits are sent to the output and simultaneously into the division circuit. After the k information digits have been sent out, and the remainder (i.e. the parity checks) is in the register, switch S_1 is opened and S_2 is moved to position 2. During the next (n-k) clock pulses the parity checks are transmitted. This process is repeated for the next block of information digits.

3.5 Encoding with a k-Stage Shift-Register

Since G(x) divides X^n-1 it follows that

$$x^n-1 = G(x)H(x) \qquad (3.1)$$

The polynomial H(x) completely specifies an (n,k) cyclic code with generator polynomial G(x), as will be shown. Let V(x) be a codeword, i.e.

$$V(x) = P(x)G(x) \qquad (3.2)$$

where $V(x) = V_0+V_1x+V_2x^2+ \dots + V_{n-1}x^{n-1}$. Multiplying both sides in (3.2) by H(x) gives:

$$V(x)H(x) = P(x)G(x)H(x) \qquad (3.3)$$

$$\text{or, } V(x)H(x) = P(x)(x^n-1)=P(x)x^n-P(x) \qquad (3.4)$$

From (3.2) it is seen that P(x) has degree at most k-1. This fact allows us to conclude that (3.4) can not contain the terms $x^k, x^{k+1}, \dots, x^{n-1}$, i.e. their coefficients in (3.4) are zero. This implies that:

$$\sum_{i=o}^{k} H_i V_{n-i-j} = 0 \qquad \text{for } 1 \leq j \leq n-k \qquad (3.5)$$

Since, from (3.1) it is known that $H_o=1$ and $H_k=1$, (3.5) can be written as:

$$V_{n-k-j} = \sum_{i=o}^{k-1} H_i V_{n-i-j} \qquad \text{for } 1 \leq j \leq n-k \qquad (3.6)$$

Expression (3.6) represents a recurrence relation which is also known as a difference equation (Lin, 1970) and provides a rule for calculating the parity check digits V_{n-k-1}, $V_{n-k-2},\ldots,V_o$, given the k information digits $V_{n-1},V_{n-2},V_{n-3},\ldots,V_{n-k}$. So, an (n,k) cyclic code with generator polynomial G(x) is completely specified by the polynomial H(x), which is called the parity polynomial of the code. The following circuit diagram (Figure 3.2) shows an encoder based on expression (3.6), using a k-stage shift-register.

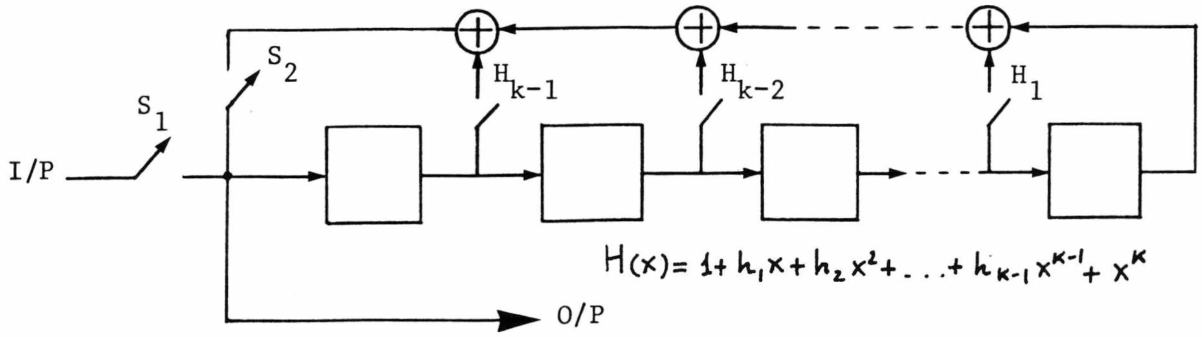


FIGURE 3.2 : Encoder Using k S.R. Stages

This circuit works as follows:

(1) With switch S_1 closed and S_2 open. Information

digits are clocked into the circuit and simultaneously to the output.

(2) After k shifts, switch S_1 is opened and S_2 is closed. At the output appears V_{n-k-1}, where $V_{n-k-1} = H_0V_{n-1} + H_1V_{n-2}+\ldots+H_{k-1}V_{n-k}$, and this constitutes the first parity check.

(3) After one clock shift the first parity check is sent out and into the encoding circuit. The second parity check then appears at the output $V_{n-k-2} = H_0V_{n-2} + H_1V_{n-3}+\ldots+H_{k-2}V_{n-k} + H_{k-1}V_{n-k-1}$.

This procedure continues until the last parity check has been sent out, then switch S_1 is closed and S_2 is opened. The next information block can now be shifted into the encoding circuit. The k-stage shift-register encoder should be used whenever $n-k>k$; otherwise the (n-k)-stage encoding circuit is preferable. It should be observed, however, that the k-stage encoder can be implemented with conventional integrated circuit shift registers since it does not require any external gates to be connected in between stages. This can be an advantage even if $n-k<k$.

On the other hand, for the calculation of syndromes (see next section) and subsequent error correction the n-k stage circuit is normally preferred because the operation of multiplying the syndrome polynomial by x and reducing modulo G(x) is easily accomplished.

3.6 Syndrome Calculation

It is the function of the decoder, on receiving an n-tuple R(x), to check whether or not it is a valid codeword. This is necessary because after passing through a noisy channel, part of the received codeword may be corrupted by noise. If errors are detected, i.e. the syndrome is non-zero, then the decoder starts attempting correction. This section deals only with syndrome calculation and consequent error detection, leaving error correction for a future section. In the case of cyclic codes, both syndrome calculation, and subsequent error correction, are relatively more simple than in the case of linear codes in general.

Since in a cyclic code all the codewords are multiples of the code generator polynomial, the first action of the decoder is to check whether the received n-tuple is divisible by G(x). The remainder of this division is the syndrome, and if it is zero it can be assumed that no errors have occurred. A non-zero syndrome indicates that the decoder has detected errors and can proceed with correction. If R(x) is the received n-tuple it can be written as

$$R(x) = V(x) + E(x) \ ,$$

where V(x) and E(x) represent respectively the transmitted codeword and the error pattern. Division of R(x) by G(x) gives:

$$R(x) = Q(x)G(x) + S(x) \qquad (3.7)$$

where S(x) is the syndrome polynomial and has degree at most (n-k-1) i.e. it is an (n-k)-tuple.
Expression (3.7) above can be written as:

$$V(x) + E(x) = Q(x)G(x) + S(x) \tag{3.8}$$

or $$\mathbf{P}(x)G(x) + E(x) = Q(x)G(x) + S(x) \tag{3.9}$$

Finally, $$E(x) = (Q(x) + \mathbf{P}(x))G(x) + S(x) \tag{3.10}$$

It is clear from expression (3.10) that there is a definite relation between syndrome and error pattern; i.e.

$$S(x) = \text{REM}\left\{\frac{E(x)}{G(x)}\right\}$$

This will be explored later for error correction purposes. It follows from the properties of the standard array (see Chapter 2) that different E(x) polynomials can lead to the same syndrome and in what follows E(x) will always be assumed to be the polynomial of smallest weight satisfying the relation

$$S(x) = \text{REM}\left\{\frac{E(x)}{G(x)}\right\}.$$

For the binary symmetric channel this leads to maximum likelihood decoding (Lucky et al, 1968). The circuits used for syndrome calculation are similar to those for the encoding of cyclic codes. This means that either k or (n-k)-stage syndrome calculating registers can be used. Figure 3.3 shows a circuit diagram for the (n-k) stage type of syndrome calculating register. Initially, the shift register contents are all zeros. The incoming n-tuple is shifted into this circuit and after n clock pulses, the shift register has the syndrome as its contents. Before receiving the next n-tuple this shift register needs to be cleared. No pre-multiplication is required, since the received parity checks must be added to the recalculated parity checks anyway.

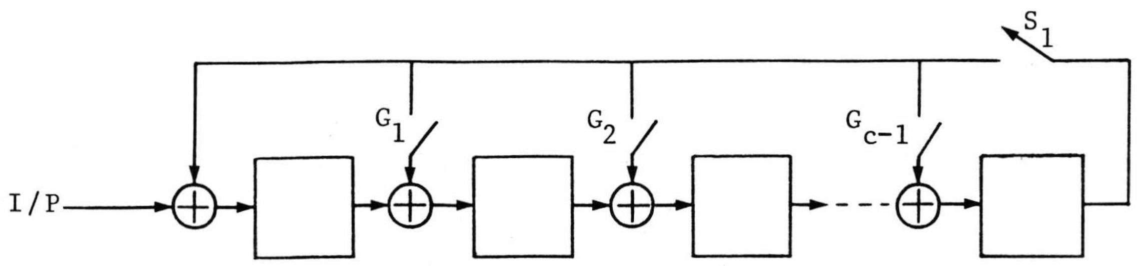


FIGURE 3.3 : (n-k)-Stage Syndrome Calculating Register

A k-stage syndrome calculating register is shown in Figure 3.4. In this circuit the incoming n-tuple is shifted into the register with switch S_1 closed and switches S_2,S_3 and S_4 open. After k shifts switches S_2, S_3 and S_4 are closed and S_1 is opened, the recalculated parity checks are added modulo-2 to the received ones and the syndrome digits appear at the output.

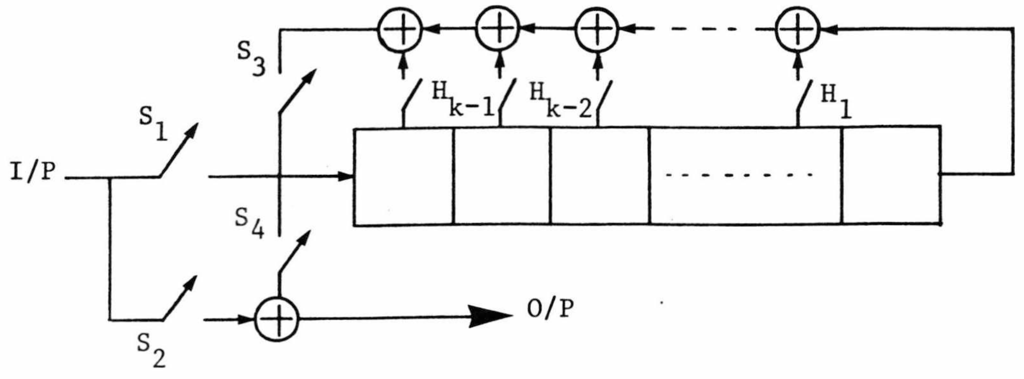


FIGURE 3.4 : k-Stage Syndrome Calculating Register

3.6 Shortened Cyclic Codes

From an engineering point of view, it is important that a particular class of easily decodable codes should cover a wide range of code rates and error-correcting power. Since the generator polynomial of a cyclic code has to divide X^n-1, the designer is left with a relatively small number of cyclic codes for a given n. However, for every $i<k$, it is possible to have an $(n-i,k-i)$ code which is obtained from an (n,k) cyclic code by selecting all those codewords which begin with i zeros, and deleting these first i zeros. There are, of course, 2^{k-i} codewords in this shortened code which is linear but no longer cyclic (Peterson, 1972). Encoding and syndrome calculation can be done with the circuits used for the original code since the i zeros deleted do not affect the calculation of the parity checks. This is the first important point

about shortened cyclic codes. They are as easy to implement as full cyclic codes. The second important point is that the Hamming distance of the shortened code is at least the same as that of the original code. For decoding the codewords are preceded by i zeros if the decoder for the original cyclic code is to be used. This solves the problem of decoding shortened cyclic codes, but the method does not fully exploit the fact that the redundancy of the code remains constant while the number of codewords has been reduced from 2^k to 2^{k-i}. This means that, potentially, more errors could be corrected with the shortened code because there are fewer information positions to be checked. Simplicity of implementation may offset the shortfall in performance, however, thus making shortened cyclic codes attractive in certain circumstances.

3.8 Pseudo-Cyclic Codes

So far what has been written about cyclic codes has implied working with polynomials reduced modulo X^n-1. If instead modulo f(x) is used, where f(x) is an arbitrary polynomial in x, the resulting codes are called pseudo-cyclic codes. Peterson (1972) proves the following theorems for cyclic codes:

Theorem 3.2 Every pseudo-cyclic code with minimum distance greater than 2 is a shortened cyclic code.

Theorem 3.3 Every shortened cyclic code is a pseudo-cyclic code.*

3.9 Decoding Cyclic Random Error Correcting Codes

In this section some of the more important algorithms for decoding cyclic codes are presented together with comments on their advantages and limitations. The decoding of random error correcting codes is in general a difficult matter, and one can say that in most practical situations, it is the complexity of the decoder that sets a limit as to which codes can be used. For cyclic codes, encoding and syndrome calculation have already been shown to be easily attainable with shift registers, modulo-2 adders, a few logic gates and a timing circuit.

3.9.1 Meggitt Decoder

This type of decoder works as follows. A received n-digit codeword, possibly erroneous, is fed to a syndrome calculating register and simultaneously into a buffer register. The syndrome obtained is fed into a combinational logic circuit (CLC) whose output is a ONE if and only if this syndrome is associated with a correctable error pattern having an error in its highest order digit, i.e. an error in the first digit to be read out of the buffer register or the digit with highest degree in a polynomial representation.

* In the multi-level case this still holds true because f(x) is a factor of x^n-1, for some n.

If the output of the CLC is zero, it is assumed that the highest order digit in the buffer is correct. So, the output from the CLC indicates whether the digit to come out from the buffer is in error or not. The contents of the buffer are read out one at a time, the syndrome register being shifted simultaneously. Every time a ONE is output from the CLC it is added modulo-2 to the buffer output and simultaneously to the feedback path of the syndrome register in order to remove the effect of the error from the syndrome calculation. After n shifts the contents of the syndrome register should be all zeros if the error pattern was a correctable one otherwise an uncorrectable error has been detected. A simple example is now given in order to make clear the points explained above.

EXAMPLE Consider the (7,4) single error correcting Hamming code whose generator polynomial is $G(x)=x^3+x+1$. The parity check equations for this code are:

$$c_1 = k_1 + k_2 + k_3$$

$$c_2 = k_2 + k_3 + k_4$$

$$c_3 = k_1 + k_2 + k_4$$

Let the transmitted codeword be:

$$\begin{array}{ccccccccc} & & k_1 & k_2 & k_3 & k_4 & c_1 & c_2 & c_3 \\ [V] & = & [1 & 1 & 0 & 1 & 0 & 0 & 1] \end{array}$$

Let the error vector be:

$$[E] = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

The received n-tuple will be:

$$[R] = [V] + [E]$$

$$[R] = [1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1]$$

The received n-tuple is fed into the circuit shown in Figure 3.5 with the switch S_1 open. The contents of the syndrome register after n shifts is the pattern

$$[S] = \begin{array}{ccc} [0 & 0 & 1] \\ c_1 & c_2 & c_3 \end{array}$$

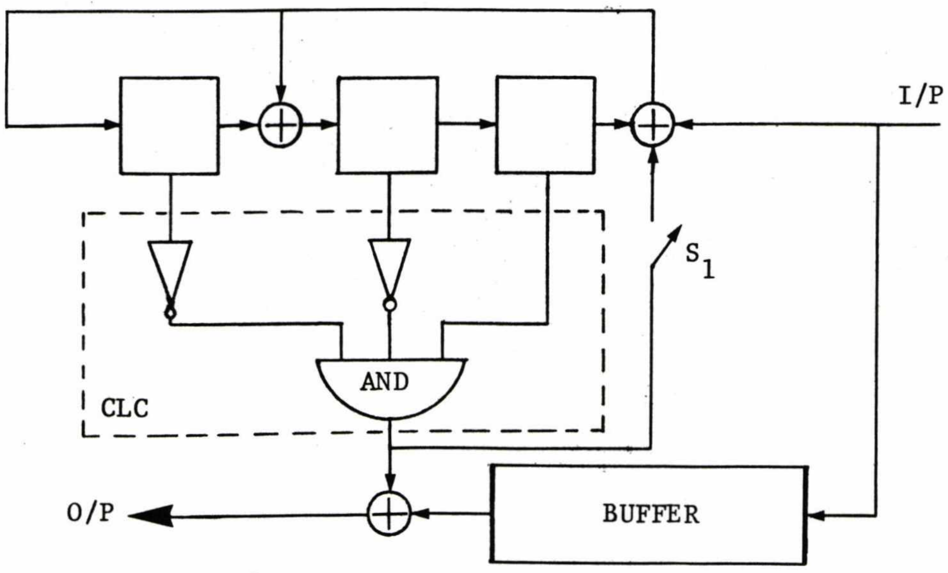


FIGURE 3.5 : Meggitt Decoder for the (7,4) t=1 Cyclic Code

S_1 is now closed and the contents of the buffer start being delivered, as follows in the table:

Shift	Contents of Syndrome Register	Output of CLC	Output of Buffer	Output of Decoder
1	0 0 1	0	1	1
2	0 1 0	0	1	1
3	1 0 0	1*	1	0
4	0 0 0	0	1	1
5	0 0 0	0	0	0
6	0 0 0	0	0	0
7	0 0 0	0	1	1
	c_1 c_2 c_3			

*Error located

TABLE 1

It is thus seen that the error in the third position (k_3) of the received n-tuple is corrected. If one is interested only in correcting errors in the information digits a k-stage buffer should be used. The decoder shown in this example uses pre-multiplication of the incoming sequence by x^{n-k} modulo G(x) so the resulting syndromes appear multiplied by x^{n-k} (Peterson, 1972). When decoding shortened cyclic codes the incoming codeword should either be pre-multiplied by

$$f(x) = \text{REM}\left\{ \frac{x^{n-k+i}}{G(x)} \right\}$$

(where i is the number of information digits removed from the code), or preceded by i zeros and applied to the decoder for the original unshortened code. The next diagram (Figure 3.6) shows the Meggitt decoder for the code resulting from shortening by i=2 the (7,4) code of the above example. Pre-multiplication by

$$f(x) = \text{REM}\left\{\frac{x^5}{x^3+x+1}\right\} = x^2+x+1$$

has been used.

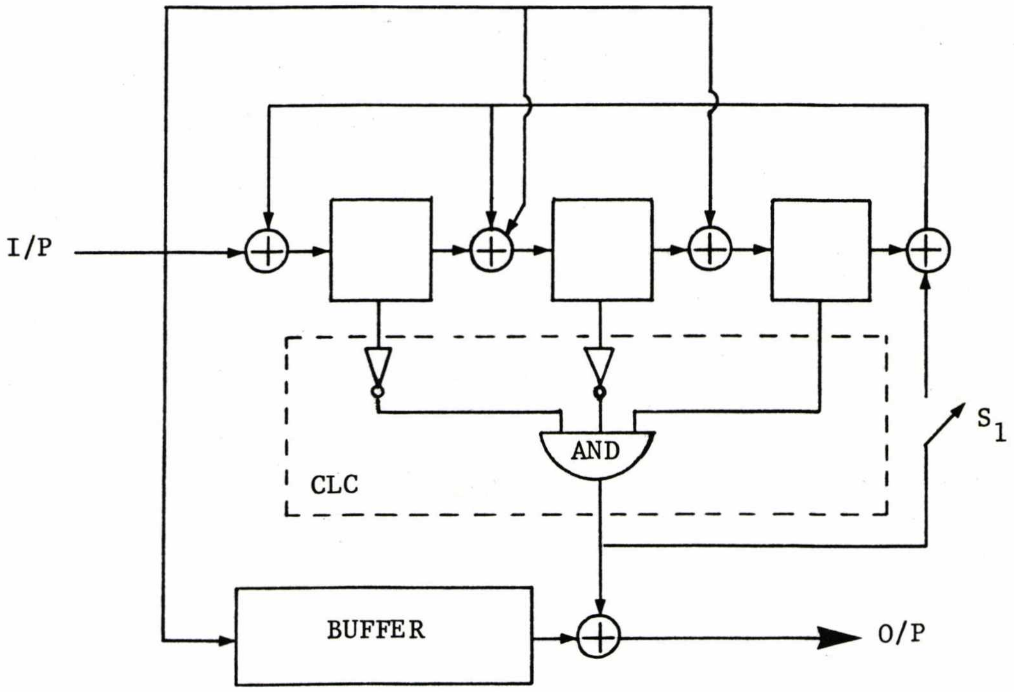


FIGURE 3.6 : Meggitt Decoder for the (5,2) t=1 Shortened Cyclic Code

The difficulty with the Meggitt decoder is the complexity associated with its CLC for codes that correct two or more errors per codeword. However, with the advent of more complex integrated circuits, this decoding procedure can be attractive. The use of programmable read only memories (PROMs) simplifies drastically the design of the

CLC. The syndrome digits in the decoder may be used as addresses to the PROMs. With the aid of a small computer programme, those syndromes which are associated with error patterns having the highest order digit in error are determined, and the PROMs are then programmed to output a ONE in these cases and a zero otherwise. For a code having c check digits a 2^c bit PROM is needed (or equivalent combination of PROMs). It is clear that in this way full use can be made of the 2^c syndromes available. As a consequence, when decoding quasi-perfect codes, one is able to correct some error patterns with t+1 errors which are usually left aside by other methods. Some quadratic residue codes, which have good distance properties but are hard to decode, can be considered for practical implementation using these techniques. The limitation of the extent to which PROMs can be applied is dictated by the available technology. Finally, the Meggitt decoder provides a general method of fully decoding any cyclic code, i.e. it does not suffer from the limitations of bounded distance decoders. The practical implementation of Meggitt decoders requires either the use of duplicate buffers or operation at twice the line clock rate.

3.9.2 <u>Error Trapping Decoding</u>

Error trapping decoding (Rudolph and Mitchell, 1964) is possibly the simplest way to decode a cyclic code when the errors to be corrected do not spread over a span greater than (n-k) digits, including the end around case i.e. regarding the span cyclically. Because of the cyclic nature of the code, such error patterns can be shifted entirely into the parity check section of the received n-tuple. When this happens the errors are said to be "trapped" and can then be corrected, as will now be explained. It has already been shown that the syndrome S(x) of a received n-tuple R(x) is equal to the remainder resulting from the division of the error polynomial E(x) by the generator polynomial G(x) of the code, i.e.

$$E(x) = Q(x).G(x) + S(x) .$$

$$\text{If } E(x) = E_{n-k-1}x^{(n-k)-1}+E_{n-k-2}x^{(n-k)-2}+\ldots+E_1x+E_0,$$

then the errors are confined to the parity check section of R(x), then Q(x) = 0 and E(x) = S(x). Correction is achieved by adding (modulo-2) R(x) to S(x). Now suppose that the errors are not situated in the parity check section and E(x) is of the form:

$$E(x)=E_{n-k-1+i}x^{(n-k)-1+i}+E_{n-k-2+i}x^{(n-k)-2+i}+\ldots+E_{i+1}x^{i+1}+E_ix^i$$

By shifting R(x) cyclically n-i times, the errors will occupy the parity check positions of $R(x)^{(n-i)}$ which is a shifted version of the received n-tuple. So, the syndrome of $R(x)^{(n-i)}$ is identical to the error affecting R(x) and so R(x) can be corrected as in the previous case.

The sequence of operations necessary to decode by this process is therefore:

(1) Shift the received n-tuple into the syndrome register and into a buffer simultaneously.

(2) Calculate the weight of the syndrome W(s).

(a) If W(s)$\leq$t then the errors are confined to the parity check section of the received n-tuple*and the information digits are error-free. No correction is necessary, just deliver the k information digits to the data sink.

(b) If W(s)>t then shift the syndrome register and after every shift check if W(s) has gone down to t or less. Suppose that W(s)$\leq$t after i shifts for 0<i$\leq$n-k. Now, inhibit the feedback connections of the syndrome register and apply (n-k)-i shifts to it. When this is done the first i higher order digits in the syndrome register match the errors in the first i positions of the buffer (Lin, 1970). The other information digits are error-free. Correction is accomplished by shifting both together the buffer and the syndrome register, and adding their outputs modulo-2.

(3) If W(s) does not go down to t or less after (n-k) shifts, then start delivering the buffer contents to the data sink. At the same time the syndrome register is shifted and if W(s)$\leq$t is verified the feedback connections are inhibited. The pattern in the syndrome register matches the errors in the next (n-k) digits to come out from the buffer. The pattern in the syndrome register is then

---

* This converse also holds true.

added to the digits coming out of the buffer as both registers are shifted together. If it happens that the weight of the syndrome never goes down to t or less after k shifts, then either an uncorrectable error has been detected or a correctable error pattern with errors not confined to (n-k) consecutive positions has occurred. Figure 3.7 shows the circuit diagram of a decoder for error trapping.

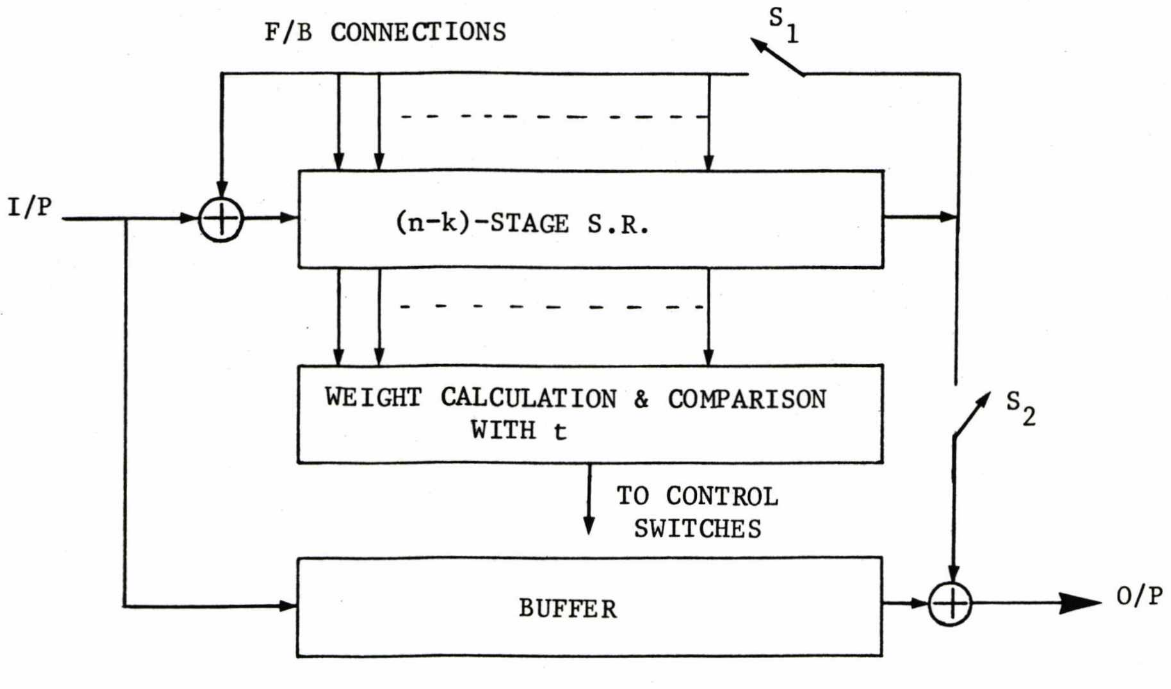


FIGURE 3.7 : Error-Trapping Decoder

In general, for a code to be efficiently decoded by error trapping the following condition must hold:

$$\frac{n}{k} > t$$

(Peterson, 1972). Due to this, error trapping is seen to be effective for low rate and/or small t codes only. Care is necessary when applying error trapping to relatively more powerful codes, because a good deal of the code error correcting ability can be wasted. Many coding theorists have tried to extend the applicability of the error trapping technique to cover the case when the error patterns are not all confined to (n-k) consecutive positions (Kasami, 1964), (MacWilliams, 1964). An example of such an extension is permutation decoding, which is described in the next section.

3.9.3 Permutation Decoding

This is a more sophisticated type of error trapping. In the case of error trapping, only cyclic shifts are applied to the received n-tuple when trying to trap the erroneous digits. It is important to observe that cyclic shifts are code preserving permutations. This means that the same decoder can be used after cyclic shifts are applied to the received n-tuple. If the permutations used are not code preserving, the decoder would normally have to be modified in order to deal with the new syndromes, which is obviously a disadvantage. Successful application of permutation decoding depends on finding a set of code preserving permutations which rearrange the codeword digits in such a way as to leave at least k consecutive correct digits in a codeword containing errors. Error trapping can then be applied to correct the errors and after that an inverse operation is required to restore the original codeword.

Cyclic codes of odd length are preserved by the permutation

$$X^j \rightarrow (X^j)^{2^i}$$

$$j = 0,1,2,\ldots,n-1$$

$$i = 1,2,\ldots$$

applied to the digits of any of their codewords (MacWilliams, 1964). In many cases this permutation together with cyclic shifts is sufficient to decode a given code. However, very little is known about code preserving permutations and for a given (n,k,t) code it is not known in general how to find a set of permutations to decode it, or even if such a set exists. Permutation decoding is treated in Peterson (1972) and MacWilliams (1964).

3.9.4 <u>Majority Logic Decoding</u>

The majority logic decoding algorithm is an efficient scheme for decoding certain classes of cyclic codes; as for example the m-sequence codes and finite projective geometry codes (Lin, 1970). Majority logic decoding can also be applied to other classes of block codes; however, the final circuit for the decoder is not always as simple as for cyclic codes. Convolutional codes can also be decoded by majority logic techniques. Historically Reed (1954) was the first to suggest the idea of majority logic decoding in order to decode the Reed-Muller codes (Muller, 1954). Later his work was extended by other coding theorists and two good accounts of this subject can be found in Massey (1963) and Peterson (1972).

a. Orthogonal parity sums

The syndrome $[S]$ of a received n-tuple in an (n,k) cyclic code with parity check matrix $[H]$ can be written as:

$$[S] = [S_0, S_1, S_2, \ldots, S_{n-k-1}] = [E].[H]^T \tag{3.11}$$

where $[E] = [E_0, E_1, E_2, \ldots, E_{n-1}]$ represents an error pattern. Expanding equation (3.11) leads to:

$$\left.\begin{array}{l} S_0 = \sum\limits_{i=0}^{n-1} E_i H_{0i} \\ S_1 = \sum\limits_{i=0}^{n-1} E_i H_{1i} \\ \vdots \\ S_{n-k-1} = \sum\limits_{i=0}^{n-1} E_i H_{n-k-1i} \end{array}\right\} \tag{3.12}$$

Now consider a combination of syndrome digits as follows:

$$A = a_0 S_0 + a_1 S_1 + \ldots + a_{n-k-1} S_{n-k-1} \tag{3.13}$$

a_i is either zero or one. Due to equation (3.12) A can be written as:

$$A = b_0 E_0 + b_1 E_1 + b_2 E_2 + \ldots + b_{n-1} E_{n-1} \tag{3.14}$$

b_i is either zero or one. An error position E_i is said to be checked by A if its coefficient b_i in equation (3.14) is ONE. Equation (3.14) is called a parity check sum.

Definition 3.3 Given a set of J parity check sums (parity check equations) $A_1, A_2, \ldots, A_J$ such that one error position E_ℓ is checked by all of them and all other error positions appear once and only once in each parity check sum, then this set is said to be orthogonal on the position E_ℓ.

If, instead of having E_ℓ common to all equations, we have $E=E_i+E_j+\ldots+E_\ell$ as the common term in all equations, then the following definition arises.

Definition 3.4 A set of J parity check sums $A_1, A_2, \ldots, A_J$ is said to be orthogonal on the set E if, and only if, E is checked by all the J parity check sums and no error position outside E appears in more than one parity check equation.

Definition 3.3 is the basis for what is known as ONE-STEP MAJORITY LOGIC DECODING while Definition 3.4 can be conceived of as a generalisation of Definition 3.3 and is the basis for L-STEP MAJORITY LOGIC DECODING.

b. ONE-STEP MAJORITY LOGIC DECODING

Given an (n,k) cyclic code suppose we can find J parity check sums orthogonal on the highest order position of the received n-tuple, and therefore on the highest order digit, E_{n-1}, of the error pattern. If no more than J/2 errors affect a transmitted codeword they can be corrected as follows. Suppose the highest order digit is not in error, then at most J/2 parity check sums will fail, leaving at least J/2 of them agreeing. So, if a clear majority of the parity sums are zero (i.e. agree) or if a tie results, the highest order digit is delivered unchanged by the decoder. Now suppose E_{n-1} is in error, this leaves $\frac{J}{2} - 1$ errors to be spread among at most $\frac{J}{2} - 1$ parity equations. This leaves at least $J - \left[\frac{J}{2} - 1\right]$ equations in which only $E_{n-1} = 1$, and so a clear majority of the sums will be ONE. The highest order digit can in this way be corrected by the decoder. In the case of cyclic codes, position E_{n-2} can be corrected in the same way as described above because after a cyclic shift it will occupy the position of E_{n-1}. Once the error effect on E_{n-1} has been removed from the parity sums the estimate of E_{n-2} can proceed. This process continues until the complete codeword is decoded.

The process of decoding by a majority rule is very efficient whenever J is equal to or very close to d-1 because $t = \left\lfloor\frac{d-1}{2}\right\rfloor \geq \left\lfloor\frac{J}{2}\right\rfloor$ is the error correcting capability of the code. Also some patterns having more than t errors can be corrected. Codes for which J = d-1 are said to be completely orthogonalisable.

<u>EXAMPLE</u> Consider the m-sequence code with parameters n=7, k=3, d=4 and generator polynomial $G(x) = \frac{x^7+1}{P(x)}$, where P(x) is a primitive polynomial of degree k. If P(x) is chosen to be x^3+x+1, then the following G(x) results:

$$G(x) = (x^3+x^2+1)(x+1) = x^4+x^2+x+1.$$

Depending on the way the parity check sums are constructed two basically different implementations result. They are known as Type I and Type II majority logic decoders (Massey, 1963). The polynomial $P'(x) = x^k P(\frac{1}{x})$ is the reciprocal of P(x) and is also primitive (Peterson, 1972). It is in the null space of the (7,3) code.* In this example $P'(x) = x^3+x^2+1$ and the following set of equations can be used for parity checking.

$$\left.\begin{aligned} P'(x) &= x^3+x^2+1 \\ xP'(x) &= x^4+x^3+x \\ x^3P'(x) &= x^6+x^5+x^3 \end{aligned}\right] \blacktriangleright \qquad (3.15)$$

It is easily seen that this set is orthogonal on position x^3. The next diagram (Figure 3.8) shows a syndrome calculating register using pre-multiplication by x^{n-k}, i.e. x^4. By doing that, the highest order position in the code (x^{n-1}, i.e. x^6) will appear as $x^6.x^4 = x^{10}$ or x^3 modulo x^7+1 and so (3.15) can be used to check for errors on the highest order position of the received codewords. In terms of ONES and ZEROS (3.15) can be written as:

$$\begin{array}{ccc|c|cccl} k_1 & k_2 & k_3 & c_1 & c_2 & c_3 & c_4 & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & \qquad (3.16) \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & \\ E_2 & E_1 & E_0 & E_6 & E_5 & E_4 & E_3 & \dagger \end{array}$$

*The polynomial P'(x) and its multiples taken modulo X^7+1 form the null space of the (7,3) m-sequence code, and so can be used as parity check equations.

†Digits rearranged after multiplication by x^4.

$$\left.\begin{aligned} A_1 &= S_3 \\ A_2 &= S_0 \\ A_3 &= S_1+S_2 \end{aligned}\right\} \qquad (3.17)$$

$$\left.\begin{aligned} A_1 &= E_6+E_5+E_3 \\ A_2 &= E_6+E_4+E_0 \\ A_3 &= E_6+E_2+E_1 \end{aligned}\right\} \qquad (3.18)$$

The set of equations (3.17) express orthogonality in terms of syndrome digits and is used to construct a Type I one-step majority logic decoder (Figure 3.8), while (3.18) refers to Figure 3.9 where a Type II decoder is shown.

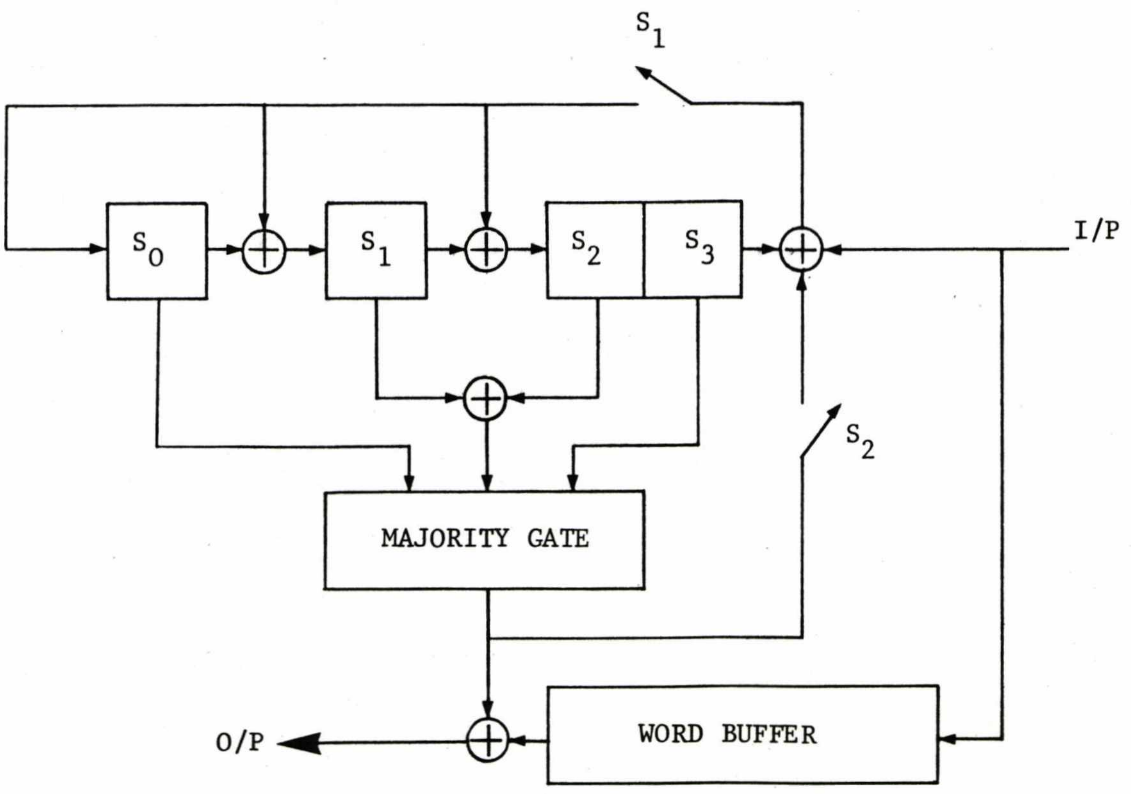


FIGURE 3.8 : Type I One-Step Majority Logic Decoder

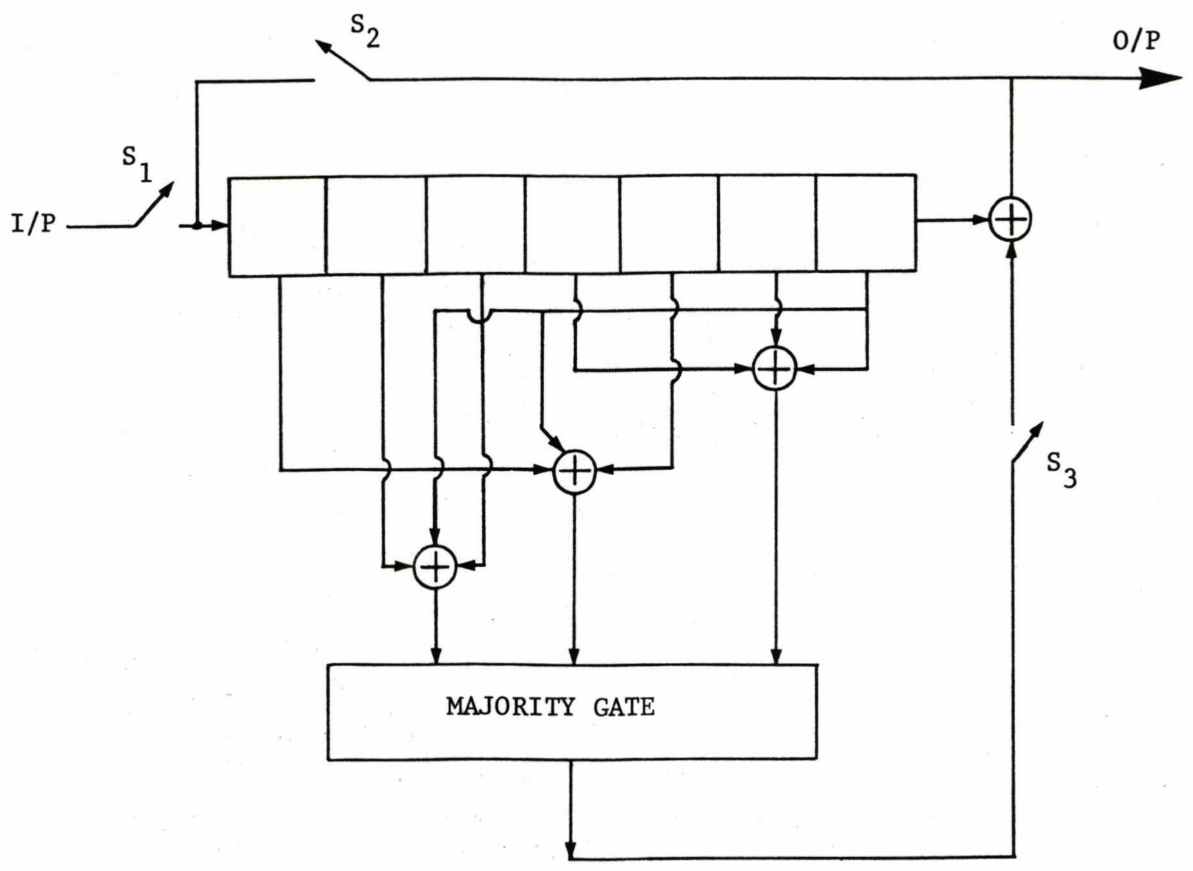


FIGURE 3.9 : Type II One-Step Majority Logic Decoder

TYPE I DECODER: The syndrome of a received n-tuple is calculated in the usual manner with S_1 closed and S_2 open. The output from the majority gate is used to correct the digits coming out from the word buffer, and it is also fed back to the syndrome calculating register to remove the effect of the errors corrected from the syndrome. After the last digit stored in the word buffer has been delivered, the contents of the syndrome register should be all zeros, otherwise an uncorrectable error has been detected.

TYPE II DECODER: The incoming n-tuple is fed into the parity check register with S_1 closed and S_2 and S_3 open. For the next n-shifts the output from the majority gate is added modulo-2 to the received n-tuple as this is shifted out of the parity check register, in order to correct possible errors. The final output is fed back to the input and after n-shifts the inputs to the majority gate should all be zero; i.e. the parity check register contains a valid codeword. Otherwise an error has been detected which cannot be corrected.

The following theorem establishes an upper bound for the performance of one-step majority logic decoding.

THEOREM 3.4 (Peterson, 1972) Given an (n,k) code with $\overline{d}$ as the minimum distance of its dual code, the maximum number of errors t_1 per codeword that can be corrected using one-step majority decoding is

$$t_1 \leq \frac{n-1}{2(\overline{d}-1)}$$

This result means that many codes cannot be efficiently decoded by the one-step majority logic method, e.g. the Golay code, for which $t_1 = 1$.

c. L-STEP MAJORITY LOGIC DECODING

ONE-STEP decoding is easy to implement but relatively few classes of cyclic codes can be decoded in this way because the orthogonality condition is very restrictive. Implementation of definition 3.4, which generalises the idea of orthogonality, together with the procedures described above, allows a larger number of cyclic codes to be decoded by majority logic techniques, by means of several levels of majority gates. At each level a number of sums of orthogonal digits are estimated. This process is continued until a set of J or more parity check sums orthogonal on a single position E is obtained. The value of E can then be estimated as in the one-step case. The following theorem gives an upper bound on the performance of L-step majority logic decoding.

THEOREM 3.5 (Peterson, 1972). Given an (n,k) code, with $\bar{d}$ as the minimum distance of its dual code, the maximum number of errors t_L that can be corrected per codeword using L-step majority logic decoding is:

$$t_L \leq \frac{n-\left\lceil \frac{\bar{d}}{2} \right\rceil}{2\left(\bar{d}-\left\lceil \frac{\bar{d}}{2} \right\rceil\right)}$$

i.e. $t_L \leq \frac{n}{\bar{d}} - \frac{1}{2}$ for $\bar{d}$ even

$t_L \leq \frac{n+1}{\bar{d}+1} - \frac{1}{2}$ for $\bar{d}$ odd

Thus, for the Golay code, $t_L = 2$.

d. Alternatives to the basic majority logic decoding process

(1) The use of feedback, as in the example given, in many situations allows correction of patterns having slightly over t errors. This happens when the decoder succeeds in correcting an error in a pattern containing

say t+1 errors which leaves then a pattern with t errors, which can also be corrected.

(2) The use of non-orthogonal check sums (Peterson, 1972). This process applies mainly to codes based on finite geometries (Carmichael, 1937), and has a complexity comparable to L-step majority decoding but usually corrects fewer errors.

(3) Rudolph (1973) observed that in some cases there are orthogonal parity check equations which can be obtained from others in the same set by cyclic shifts. In this case some majority gates can be saved by employing a sequential technique to calculate and store the different estimates to be used for the final majority voting.

(4) Variable threshold (Townsend and Weldon, 1967).

The majority gates considered in this procedure have an adjustable threshold T, which is set initially at its highest value T = d-1. Decoding is then attempted. If, after the first n shifts, no error has been detected the threshold is lowered by one, the cycle is repeated and if no error is detected the threshold is lowered again by one. This goes on until $T = \frac{(d-1)}{2}$ is reached or an error is found in which case the threshold is increased by one. When $T = \frac{(d-1)}{2}$ is reached, the decoding process is considered terminated, i.e. the received word is considered to have been decoded. Whenever an error is found, for some $T > \frac{d-1}{2}$, the threshold is increased by one and after a complete cycle (n shifts) it is lowered by one and decoding continues. The end of this process is achieved when either T drops to $\frac{(d+1)}{2}$ and remains there, or enters a limit cycle where it keeps changing between two or more values. In the second case T is forced to $\frac{(d+1)}{2}$ after a specified number of revolutions and then fixed threshold decoding is tried. This usually results in correction of more errors than is possible with conventional majority logic decoding (fixed threshold).

e. Final comment

A description of the majority logic decoding algorithm has been presented with emphasis on its application for cyclic codes. When compared to the BCH decoding algorithm, in terms of complexity, majority logic decoding is generally much simpler for moderate values of n. However, the codes for which majority logic decoding is efficient are slightly inferior to the BCH codes in the same range. For large n the large number of majority gates required makes majority logic decoding unattractive and the BCH codes, besides having a better performance, have also simpler decoding algorithms.

3.10 Bose-Chaudhuri-Hocquenghem (B.C.H.) Codes

These codes were discovered independently by Hocquenghem (1959) and Bose and Chaudhuri (1960). Peterson (1960) pointed out that they were cyclic codes and described the first algebraic procedure for their decoding. Though originally described for binary symbols, the BCH codes were generalised later for the multi-level case (Gorenstein and Zierler, 1961); and the decoding procedure was refined by Chien (1964), Berlekamp (1968) and Massey (1969). The BCH codes constitute one of the most important and extensive classes of constructive codes. However, their importance stems mainly from the fact that there exists a way of decoding them which is practical for relatively large values of n. Using 50 ns switching logic, it is feasible to construct decoders for BCH codes with $n \leq 1023$ and a bit rate of 1 MHz (Wolf, 1973). Berlekamp (1968) points out that the distance properties of BCH codes are asymptotically disappointing, and other cyclic codes likes the quadratic residue codes perform much better but are harder to decode.

3.10.1 Basic Properties of BCH Codes

For a thorough understanding of BCH codes a knowledge of how to operate with Galois field is required. An elementary treatment of binary BCH codes is presented in this section. Appendix I gives some of the basic definitions of modern algebra but for a more complete coverage the reader is referred to Peterson (1972) and Berlekamp (1968).

For any positive integers $m, t (t<2^{m-1})$ there exists a BCH code with parameters:

$$n = 2^m - 1$$

$$n-k = c \leq mt$$

$$d \geq 2t+1$$

<u>THEOREM</u> 3.6 Peterson (1961) The BCH code whose generator polynomial has d-1 consecutive roots $\alpha, \alpha^2, \alpha^3 \ldots,$ α^{d-1} has minimum Hamming distance at least d.

<u>Proof</u> As shown before (see section 3.3), the statement that G(X) has roots $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{d-1}$ is equivalent to saying that the code is the null space of the matrix $\boxed{H}$ below.

$$\boxed{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots\ldots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots\ldots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \ldots\ldots & \alpha^{3(n-1)} \\ \vdots & & & & \\ 1 & \alpha^{d-1} & \alpha^{2(d-1)} & \ldots\ldots & \alpha^{(d-1)(n-1)} \end{bmatrix}$$

By selecting any set of d-1 distinct columns of $\boxed{H}$, the following determinant results.

$$D = \begin{vmatrix} \alpha^{j_1} & \alpha^{j_2} & \ldots & \alpha^{j_{d-1}} \\ (\alpha^2)^{j_1} & (\alpha^2)^{j_2} & \ldots & (\alpha^2)^{j_{d-1}} \\ \vdots & & & \\ (\alpha^{d-1})^{j_1} & (\alpha^{d-1})^{j_2} & \ldots & (\alpha^{d-1})^{j_{d-1}} \end{vmatrix} = \alpha^{j_1}\alpha^{j_2}\ldots\alpha^{j_{d-1}} D_1$$

By factoring α^{j_i} $(1 \leq i \leq d-1)$ from every column of D, a new determinant D_1 is obtained which has the form:

$$D_1 = \begin{vmatrix} 1 & 1 & \cdots\cdots & 1 \\ X_1 & X_2 & \cdots\cdots & X_{d-1} \\ X_1^2 & X_2^2 & & X_{d-1}^2 \\ \vdots & & & \\ X_1^{d-2} & X_2^{d-2} & \cdots\cdots & X_{d-1}^{d-2} \end{vmatrix} = \prod_{i>j} (X_i - X_j) \quad \text{where } X_i = \alpha^{j_i}$$

D_1 is recognised as a Vandermonde determinant and so $D_1 = 0$ if, and only if, two of its columns are identical. Given the way [H] has been constructed, there are no repeated columns, hence there are no combinations of d-1 or fewer columns of [H] which are linearly dependent. That means the BCH code which is the null space of [H] has minimum distance at least d.

Q.E.D.

Because G(X) has $\alpha,\alpha^2,\alpha^3,\ldots,\alpha^{d-1}$ as roots and since $m^2(x) = m(x^2)$, it is possible to write:

$$G(x) = \text{LCM}(m_1(x), m_3(x), \ldots, m_{2t-1}(x)), \text{ where } m_i(x)$$

is the minimal polynomial of α^i, and $d \geq 2t+1$. This expression for G(X) can have at most t factors of degree at most m, so the degree of G(x) is always $c \leq mt$.

3.10.2 Decoding of the B.C.H. Codes

BCH codes can be decoded by some of the methods previously described, e.g. Meggitt decoder, error-trapping, etc., but none of them explore fully the mathematical structure of BCH codes. A brief description of a procedure described by Peterson (1961) for decoding BCH codes which makes use of the structure of these codes is presented here.

The matrix $[H]$ from section 3.10.1 constitutes a parity check matrix for BCH codes. Assuming that a transmitted codeword is received with errors, the associated syndrome vector can be written as:

$$[S] = [V+E][H]^T = [V][H]^T + [E][H]^T = [E][H]^T$$

(since $[V]$ is a codeword $[V][H]^T = [0]$)

An error pattern containing t errors will be characterised by the positions of these errors, denoted by $\beta_1, \beta_2, \ldots, \beta_t$ which are called <u>error-location numbers</u>. Let $[S] = [S_1, S_2, S_3, \ldots, S_{d-1}]$, the expansion of $[E][H]^T$ can be represented as:

$$\left.\begin{aligned} S_1 &= \beta_1 + \beta_2 + \ldots. + \beta_t \\ S_2 &= \beta_1^2 + \beta_2^2 + \ldots. + \beta_t^2 \\ S_3 &= \beta_1^3 + \beta_2^3 + \ldots. + \beta_t^3 \\ &\vdots \\ S_{d-1} &= \beta_1^{d-1} + \beta_2^{d-1} + \ldots. + \beta_t^{d-1} \end{aligned}\right\} \quad (3.19)$$

Equations (3.19) above are called <u>power sum symmetric functions</u>. The algebraic decoding of BCH codes depends on solving this system of equations, to determine the β's from the S_i's. This system of equations is nonlinear and so far has defied simple solutions. Peterson (1961) observed that the error-location numbers can be expressed in terms of <u>elementary symmetric functions</u> as:

$$\sigma_1 = \beta_1 + \beta_2 + \ldots\ldots + \beta_t$$
$$\sigma_2 = \beta_1\beta_2 + \beta_1\beta_3 + \ldots\ldots + \beta_1\beta_t + \beta_2\beta_3 + \ldots\ldots + \beta_2\beta_t + \beta_{t-1}\beta_t$$
$$\sigma_3 = \beta_1\beta_2\beta_3 + \beta_1\beta_2\beta_4 + \ldots\ldots$$
$$\vdots$$
$$\sigma_t = \beta_1\beta_2 \ldots\ldots \beta_t$$

The power sum symmetric functions are related to the elementary symmetric functions by <u>Newton's identities</u> as follows:

$$S_1 + \sigma_1 = 0$$
$$S_3 + S_2\sigma_1 + S_1\sigma_2 + \sigma_3 = 0$$
$$S_5 + S_4\sigma_1 + S_3\sigma_2 + S_2\sigma_3 + S_1\sigma_4 + \sigma_5 = 0$$
$$\vdots$$
$$S_{2t-1} + S_{2t-2}\sigma_1 + \ldots\ldots + S_{t-1}\sigma_t = 0$$

These equations can be expressed in matrix form as

$$[S'] = [M_t][\Sigma] \qquad \text{where:}$$

$$[S'] = \begin{bmatrix} S_1 \\ S_3 \\ S_5 \\ \vdots \\ S_{2t-1} \end{bmatrix}, \quad [\Sigma] = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_t \end{bmatrix}, \quad [M_t] = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots\ldots & 0 \\ S_2 & S_1 & 1 & 0 & \ldots\ldots & 0 \\ S_4 & S_3 & S_2 & S_1 & 1 \ldots & 0 \\ \vdots & & & & & \\ S_{2t-2} & \ldots & \ldots & \ldots & \ldots & S_{t-1} \end{bmatrix}$$

$[M_t]$ is non-singular (i.e. has t independent rows) provided exactly t errors occurred. In this case, the σ's can be determined from:

$$[\Sigma] = [M_t]^{-1}[S']$$

Given the σ's, the error location numbers can be determined from the final decoding equation

$$\Sigma(x) = (x+\beta_1)(x+\beta_2)(x+\beta_3) \ \ldots \ (x+\beta_t)$$

by simple trial and error substitution. If β_i is a root of $\Sigma(x)$, i.e. $\Sigma(\beta_i) = 0$, the digit in position β_i of the received n-tuple is changed.

In case less than t errors occurred $[M_t]$ can be changed by reducing the number of equations and another tentative decoding made. Eventually, a system of equations will result which can be solved for the elementary symmetric functions. Berlekamp (1968) improved this algorithm in a way which avoids matrix inversion and Chien (1964) developed an efficient process of search for finding the roots of $\Sigma(x)$. However, algebraic decoders normally correct only up to t errors per codeword and in this sense do not make full use of the error correcting properties of the code.

3.11 Burst Error Correction

This section deals with cyclic burst error correcting codes capable of correcting a single burst of errors per block. Conditions for existence, construction and decoding are presented. These codes have been created to be used in channels where errors do not occur independently, i.e. the occurrence of one error in one digit increases the chances of having the next digit also in error. Channels of this type are said to have memory. In fact, as pointed out by Gallager (1968), with the possible exception of space channels almost all other types of channels have memory to a certain degree. This effect of memory can appear in the form of intersymbol interference due to band-limiting a signal in frequency or as intermodulation noise due to inadequate filtering. In telephone lines bursts can be caused by a stroke of lightning or a momentary overloading of an amplifier. Defects on a magnetic tape usually cause errors that occur in bursts. A burst is said to have length b with respect to a guard space g when it satisfies the following conditions:

(1) The first and last digits of a burst are ONES, the other b-2 digits can be zero or one.

(2) The burst is preceded and followed by at least g error free positions.

(3) There can be no sequences of g or more consecutive zeros inside a burst.

For example, if g=3, then

```
              b=3          b=4
     0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0
```

represents a stream of binary digits containing one burst of length 3 and another of length 4. The following theorems establish conditions for a linear code to be able to correct bursts, and the minimum number of parity checks needed. The proofs for these theorems can be found in Lin (1970) and Peterson (1972), and are based on the concept of the standard array presented in Chapter 2.

THEOREM 3.7 A necessary condition for an (n,k) linear code to correct all bursts of length b or less is that no burst of length 2b or less can be a codeword.

THEOREM 3.8 An (n,k) linear code which contains no burst of length 2b or less as a codeword has at least 2b parity checks, i.e. $n-k \geq 2b$

Combining Theorems 3.1 and 3.2 it can be seen that a burst b error-correcting code must have at least 2b parity checks. This can also be interpreted as $b \leq \left\lceil \frac{n-k}{2} \right\rceil$ which constitutes an upper bound on the burst correcting capaability of an (n,k) linear code called the Reiger bound (Reiger, 1960). Codes for which $2b = n-k$ are said to be optimum. A figure of merit for burst correcting codes can be defined as $z = (n-k)-2b$. With this definition optimum burst correcting codes have $z = 0$.

3.12 Decoding of Burst-Correcting Cyclic Codes

3.12.1 Burst Trapping Decoder

By observing the condition established in the Reiger bound ($2b \leq n-k$), it can be seen that any correctable burst pattern can be cyclically shifted into the parity check section of a received n-tuple, and consequently can be trapped in the syndrome calculating register. In this way, a simple decoder can be constructed using basically a syndrome calculating register, a counter, and an $(n-k)-b$ input OR-gate connected to the higher order end of the syndrome calculating register as shown in Figure 3.10 below.

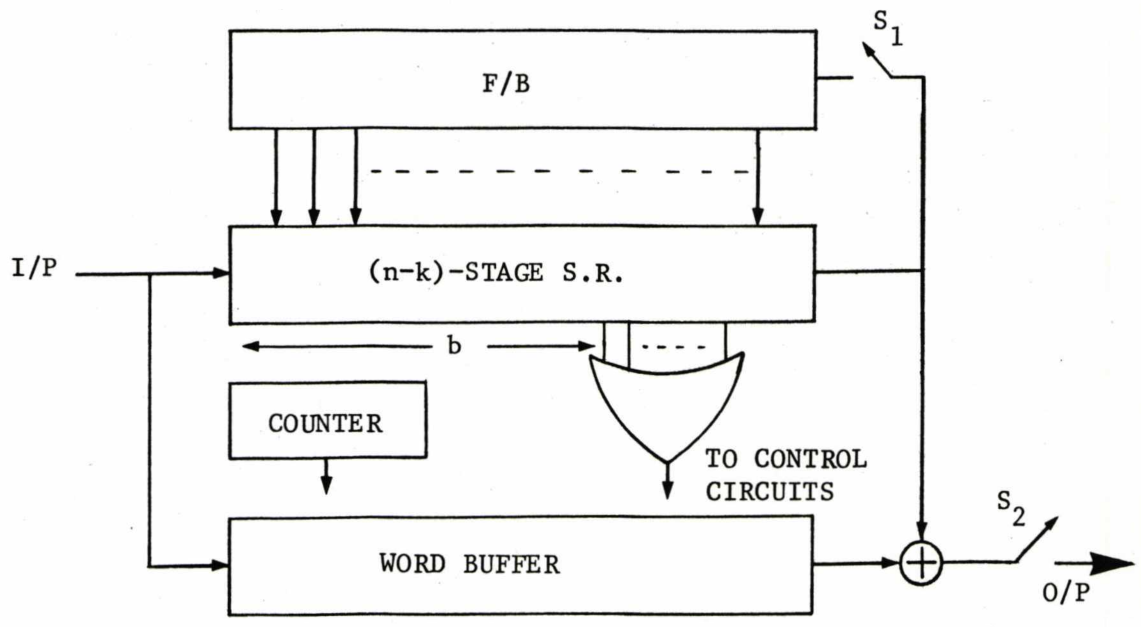


FIGURE 3.10 : Burst Trapping Decoder

The OR-gate in Figure 3.10 is used to detect, after the syndrome has been calculated, whether the burst is trapped i.e., when the $(n-k)-b$ higher order positions of the syndrome register contain zeros. The counter reads the number of cyclic shifts required to achieve that. Correction then follows in the same way as error trapping for random errors. If after n shifts the output of the OR-gate has not indicated the condition for "burst trapped" then a burst of length greater than b has been detected.

3.12.2 Gallager Optimum Burst Decoder

It is interesting to observe that for burst correction a given burst correcting code has available $2^{n-k}-1$ syndromes and for the guaranteed burst length b it uses $n.2^{b-1}$ of them, which in many cases is far less than $2^{n-k}-1$. The burst trapping technique described above can be modified in order to make full use of the redundancy available and to correct bursts of length ℓ, $b<\ell\leq n-k$. Gallager (1968) proposed a modification of the burst trapping technique which turned out to be equivalent to minimum distance decoding for channels where shorter bursts are more likely to occur than longer ones, i.e. a burst of length ℓ is more likely to occur than one of length $\ell+1$. The idea is to calculate the syndrome of a received n-tuple, and if this syndrome is non-zero, to shift the syndrome register n times and keep a record of the position of the syndrome with the longest run of consecutive zeros (using a counter, for example). This syndrome corresponds to the shortest burst and correction achieved by adding modulo-2 the contents of the syndrome register, with feedback connections inhibited, to a section of the received n-tuple, stored in a buffer register, determined by the burst location counter. More details of this type of decoder are given in Chapter 4, where an experimental system is described. If desired, this decoder can be used in a combination of error correction and error detection. A question can be asked about what to do when the decoder is faced with two "possible bursts" of the same length $\ell > b$. In this case provision can be made to decide, for example, to select the burst with the smaller weight.

3.12.3 High-Speed Burst Decoder

Chien (1969) introduced an interesting technique for decoding some special types of burst correcting cyclic codes, namely the Fire codes and their generalisation, which are covered later in this chapter. The burst decoders previously described required at least n shifts for locating and correcting a burst. This can be seen as a decoding delay due to the process of error correction and for long values of n it can be an unacceptable disadvantage. Chien's solution to the problem is to reduce the decoding delay by factorising G(x) and computing a partial syndrome for each of its factors. For example, let $G(X) = (x^c-1)P(x)$, where $P(x)$ is an irreducible polynomial belonging to an exponent p, and c is relatively prime to p. The partial syndrome obtained with x^c-1 is used to trap the burst and to determine its location to within a multiple of c. This burst is then fed into a feedback shift register, connected according to p(x), and shifted until the contents of this register matches the partial syndrome obtained with $P(x)$. The number of shifts required to trap the burst in the X^c-1 register and the number of shifts needed to match the syndrome in the $P(x)$ register are used as residues. The Chinese Remainder Theorem (Berlekamp, 1968) is then applied to determine the exact burst location using these residues. This implies using some means of computation to perform simple arithmetic multiplication and addition. For long codes ($n \leq 1000$), this decoder achieves speeds several orders of magnitude higher than it is possible with conventional burst decoders. There are codes which are more powerful than the ones for which this decoder is intended; however, their decoding can be costly because of the reduction of parallel processing, thus making them unattractive from a practical point of view.

3.13 Important Classes of Cyclic Codes

3.13.1 Hamming Codes

The parameters for these codes have already been given in Chapter 2, i.e.

$$n = 2^c-1$$

$$k = 2^c-1-c$$

$$d = 3$$

The generator polynomial of a Hamming cyclic code can be any primitive polynomial of degree c, which by definition is a factor of X^n+1. These codes are easily decoded with a Meggitt or error-trapping decoder. Also, Hamming codes are orthogonalisable in L = c-1 steps.

3.13.2 Maximum Length Sequence Codes

For any $c \geq 2$ there exists a maximum length sequence code (m-sequence code) with the following parameters:

$$n = 2^p - 1$$
$$k = p$$
$$d = 2^{p-1}$$

The generator polynomial of an m-sequence code is given by

$$G(x) = \frac{X^n - 1}{P(x)} ,$$

where P(x) is a primitive polynomial of degree p, i.e. the dual code of an m-sequence code is a Hamming code. M-sequence codes are completely orthogonalisable in one step and consequently are easily decodable by a majority logic technique. Error trapping decoding is also an efficient way of decoding m-sequence codes because these are normally low rate codes.

3.13.3 Reed-Solomon Codes

The multi-level BCH codes defined by the following parameters:

$$n = q - 1$$
$$n - k = 2t$$
$$d = 2t + 1$$

constitute the important class of Reed-Solomon codes (Reed-Solomon, 1960; and Gorenstein & Zierler, 1961). The generator polynomial of these codes has roots $(\alpha,\alpha^2,\ldots,\alpha^{d-1})$, i.e.

$$G(x) = (1-\alpha).(1-\alpha^2)\ldots(1-\alpha^{d-1})$$

Reed-Solomon codes with $q=2^m$ are used in many practical applications, as the multi-level digits can be represented by binary m-tuples. Since $d=(n-k)+1$, these codes are also maximum distance separable (Berlekamp, 1968). Some very powerful burst and random and burst error correcting codes result from using a Reed-Solomon code over GF(2) to correct any combination of t binary m-tuples in error, in a code containing m.n binary digits per block. Also, concatenated codes (Forney, 1966) use normally a Reed-Solomon code as their outer code.

3.13.4 Codes Based on Finite Geometries

Euclidean geometry codes and finite projective geometry codes (Lin, 1970 and Peterson, 1972) are important classes of majority logic decodable cyclic codes, whose construction is based on the properties of finite geometries (Carmichael, 1937). The Hamming and the Reed-Muller codes (Muller, 1954) are particular cases of Euclidean geometry codes while the finite projective geometry codes contain the difference set codes (Lin, 1970) as a particular case. The similarities among the various types of majority logic decodable codes mentioned above were explored and combined to form the class of polynomial codes (Kasami, Lin and Peterson, 1968). Polynomial codes are cyclic codes and contain as sub-classes the BCH codes, Reed-Solomon codes, generalised Reed-Muller codes and finite geometry codes.

3.13.5 Quadratic Residue Codes

Definition An integer r is a quadratic residue of a prime number n if, and only if, there exists an integer X such that:

$$X^2 \equiv r \mod.n$$

It can be shown that if $n = 8m\pm1$, then 2 is a quadratic residue of n (Berlekamp, 1968). In this case, X^n+1 can be factored into $(x+1)G_r(x)G_{\overline{r}}(x)$ (Peterson, 1972)

$$G_r(x) = \prod_{r\varepsilon R_o} (x+\alpha^r)$$

$$G_{\overline{r}}(x) = \prod_{\overline{r}\varepsilon \overline{R}_o} (x+\alpha^{\overline{r}})$$

where α is a primitive element in an extension field of GF(2), R_o is the set of quadratic residues modulo-n and $\overline{R}_o$ is the set of quadratic non-residues modulo-n. The cyclic codes with generator polynomials $G_r(x)$, $(1+x)G_r(x)$, $G_{\overline{r}}(x)$ and $(1+x)G_{\overline{r}}(x)$ are referred to as quadratic residue codes. Berlekamp (1968) points out that permutation decoding is a very efficient way of decoding certain quadratic residue codes but permutation decoders are normally more complex than comparable BCH decoders. However, for practical applications, the relatively large minimum distance of these codes, together with the difficulty in decoding them, makes quadratic residue codes more suited for use in situations where an incomplete decoding scheme is appropriate. An incomplete decoding scheme uses the code redundancy to correct up to $t_1 < t$ errors and to detect $d-(t_1+1)$ errors per codeword thus requiring a simpler decoder than when up to t errors have to be corrected per codeword.

3.13.6 Fire Codes

These codes were discovered by Fire (1959) while trying to generalise the work of Abramson (1959) on burst error correcting codes. Fire codes can correct a single burst of errors per codeword. A burst b error correcting Fire code is generated by the polynomial

$$G(x) = (x^r+1)P(x)$$

where P(x) is a primitive polynomial of degree m and order $p = 2^m-1$, $b \leq m$ and $r \geq 2b-1$. Also, p must not divide r. The block length for this code is given by

$$n = \mathrm{LCM}(r,p)$$

Chien (1969) generalised the Fire codes by defining G(x) as follows:

$$G(x) = (x^r+1)\prod_{j=1}^{s} P_j(x)$$

where $P_j(x)$ is an irreducible polynomial of degree m_j and order p_j. Also, it is required that all $P_j(x)$ be distinct and that the $p_j(j=1,2,\ldots,s)$ do not divide r. Consequently G(x) has no repeated factors and the code length is given by:

$$n = \text{LCM}(r,p_1,p_2,\ldots,p_s)$$

The main advantage of the generalised Fire codes is their high speed decoding process, described in section 3.12.3. Other powerful burst correcting codes are obtained by interleaving random or burst error correcting codes (Lin, 1970). A simple type of interleaving is obtained by storing a number of codewords at the transmitter as rows of a two-dimensional array which is transmitted by columns. This technique tries to randomise the bursts by spreading the erroneous digits among different codewords so that the receiver can correct them as random errors. Also, by using computer search techniques, many good burst error correcting codes have been discovered, see for example, Peterson (1972) or Lucky et al (1968).

CHAPTER 4

Experimental System and Computer Simulation

4.1 Introduction

In order to investigate the practical performance of cyclic codes, an experimental system was built. The need to test a number of different codes, both for burst and random errors, imposed restrictions on the design parameters of the decoder, like decoding time and overall complexity. For that reason it was decided to limit the code length to $n \leq 31$ and the number of parity check digits to $c \leq 10$. By comparison with the decoder, the design of the encoder was simple, and in what follows we shall concentrate our attention mainly on the decoder.

Figure4.1shows the system constructed in block diagram form.

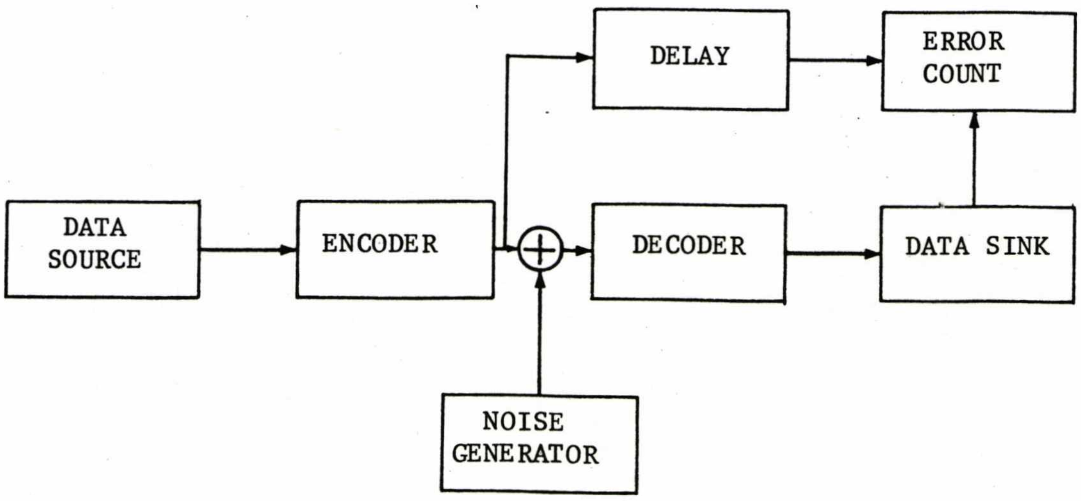


FIGURE 4.1

A brief explanation of what each block contains and its function is now given. A more detailed treatment comes later in this chapter.

The data source generates pseudo-random data which is fed to the encoder. The encoder constructed uses a c-stage shift register and adds redundancy to the data according to the code generator polynomial for which it is set.

The codewords thus generated are sent to the channel. Noise is then added to the transmitted codewords by modulo-2 addition. In this way whenever an error comes from the noise generator it changes either a one into a zero or a zero into a one. The probability with which the errors occur is externally set and can be varied over seven decades. The error generator constructed can provide both random and burst errors.

The decoder receives the codewords coming from the channel and depending on external controls operates on them either as a random or a burst error corrector. After decoding the received n-tuples are compared with their transmitted original versions, suitably delayed, and errors that either escaped correction or were introduced by the decoder can then be counted.

This experimental system was constructed with exclusively TTL logic integrated circuits. A list of the integrated circuits used is given in Appendix III.

4.2 <u>Data Source</u>

The data source was a binary m-sequence generated by the polynomial x^5+x^2+1. This sequence has length $2^5-1 = 31$. The pseudo-random properties of m-sequences (Golomb, 1967) make them appropriate for use as sources of random data. See Figure 4.2 for the diagram of the data generator.

4.3 Clock and Timing Waveforms

The main clock for the system was set at 1 MHz and was derived from a dual NAND Schmitt trigger (SN 7413). The accuracy of this frequency was not a critical parameter since all other frequencies in the system were derived from the 1 MHz source by division.

The main timing circuit is shown in block diagram form in Figure 4.3. Its job is to generate a periodic waveform composed of n-c consecutive ones followed by c consecutive zeros. This circuit works as follows. Initially, the shift register contains only zeros. The flip-flop is preset to one. When the clock starts ones begin to enter the shift register. This continues until the magnitude of word b becomes greater than or equal to that of word a. Then a one appears at the output of the word magnitude comparator which resets the flip-flop to zero. From now on, the output from this circuit is a periodic waveform of period n selected by the multiplexer and having a number of consecutive ones which is determined by the switch which is set to represent word a in the word magnitude comparator. For example, if n=7 and the switch is in position 3, we get at the output the sequence

1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 ...

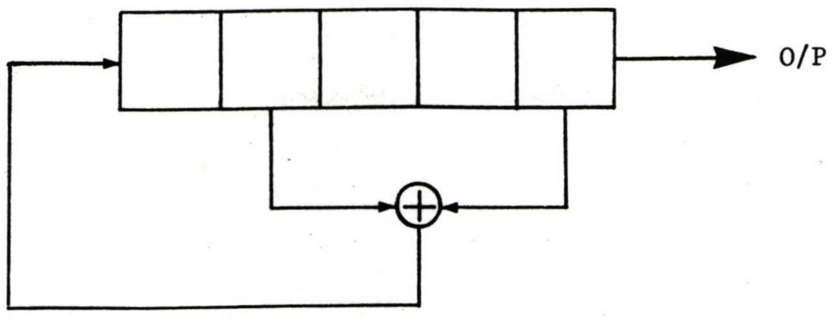


FIGURE 4.2 : Data Generator

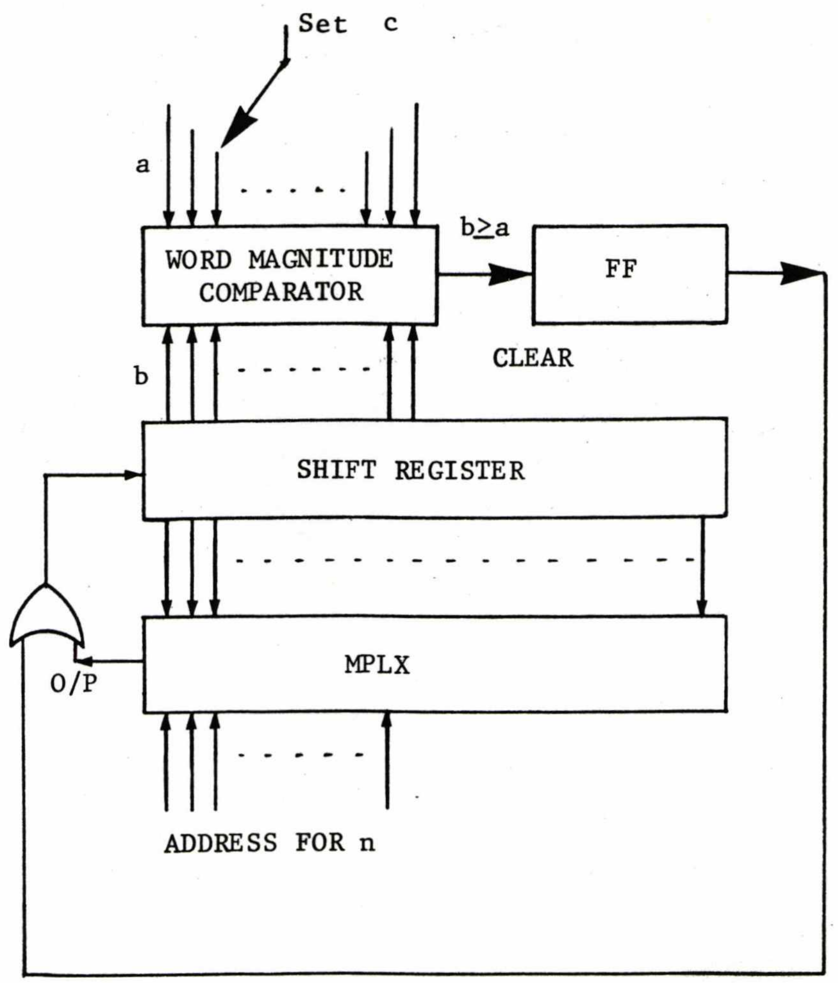


FIGURE 4.3 : Timing Waveform Generator

4.4 Encoder

The encoder constructed was of the (n-k) stage shift register type described in Chapter 3. With this encoder (see Figure 4.4) any generator polynomial of degree less than 11 can be implemented by closing the switches corresponding to terms which are missing in G(X) and leaving them open for the terms which are present. The timing waveform generator shown in Figure 4.3 supplies the necessary control waveform for the gates in Figure 4.4.

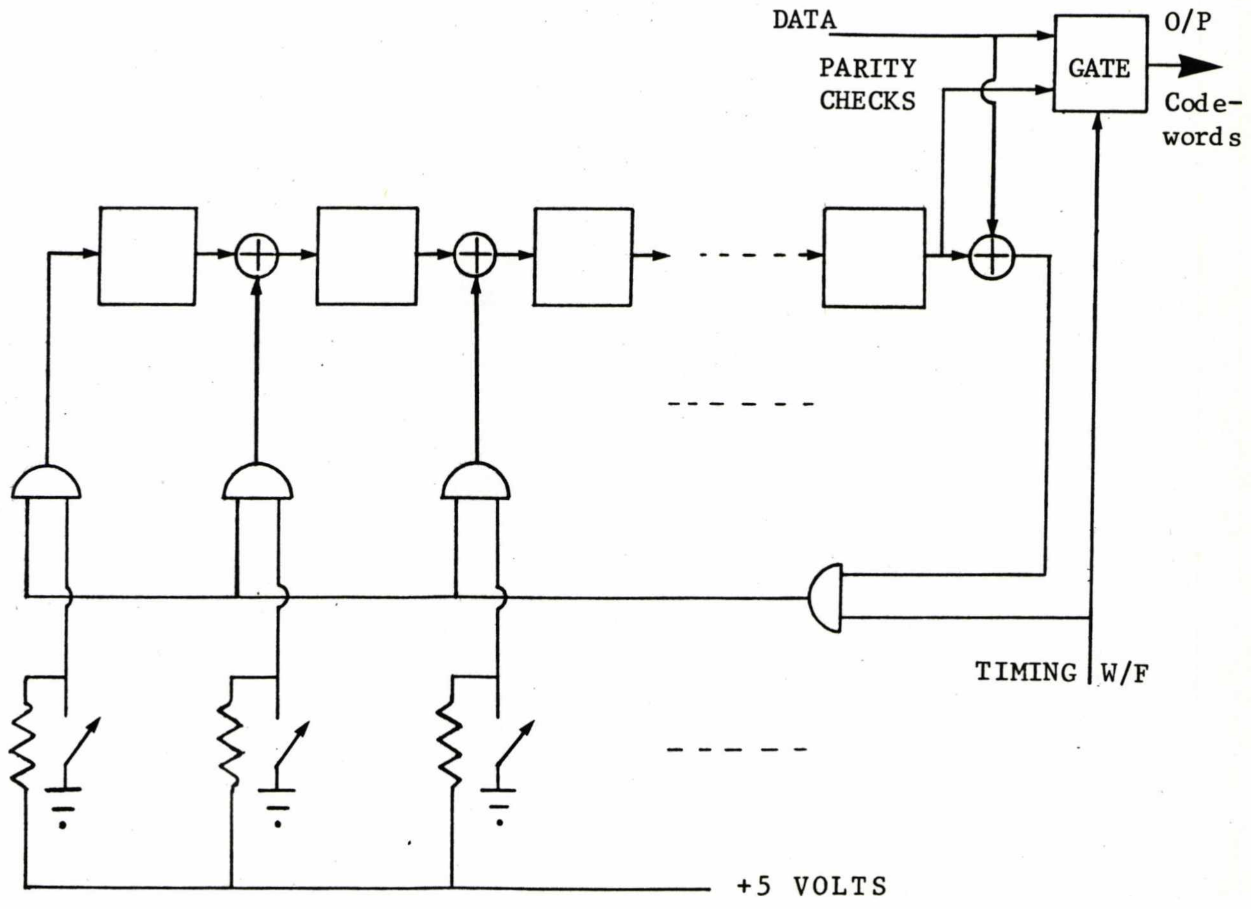


FIGURE 4.4: General Encoder For Cyclic Codes

4.5 Error Generator

4.5.1 Random Errors

In order to simulate a binary symmetric channel (BSC), an error generator was constructed and the approach taken is now explained.

It is known that the weight distribution of run lengths in a maximal-length sequence (m-sequence) follows a binomial law (Davies, 1967). One way to obtain pseudo-random noise with a binomial probability distribution then would be to use the outputs from the stages of a feedback shift register that generates an m-sequence. The next point is to determine how long this m-sequence should be. This will be set by the smallest probability required. Just for the sake of argument say it is 10^{-5} i.e., a certain condition would be detected once every 100,000 times on average. The m-sequence to be chosen should be at least 10 times 10^5 digits long otherwise the result would have a poor statistical behaviour. A good choice would be an m-sequence of length $2^{30}-1 \simeq 10^9$ which is generated by a primitive polynomial of degree 30.

If, without further considerations, one connects a feedback shift register to implement this m-sequence and combines its outputs according to some logic in order to obtain the desired probabilities of error, the result will be that the average number of errors will be as calculated, but that, however, their distribution might differ considerably from a binomial one. This is so because with the described procedure one would be implicitly assuming that each output from the feedback shift register (FSR) is:

(a) A pseudo-random source of zeros and ones with approximately equal probabilities.

(b) The outputs from the FSR stages are statistically independent.

It is assumption (b) that is not correct, on the contrary those outputs are strongly dependent since each one (excepting the lowest order one) will be equal to the previous one after one clock pulse. To overcome this difficulty we relied on the work published by Hurd (1974) of which a brief description is given below.

Consider the following arrangement where stages 3, 7 and 11 are toggle flip-flops (see Figure 4.5)

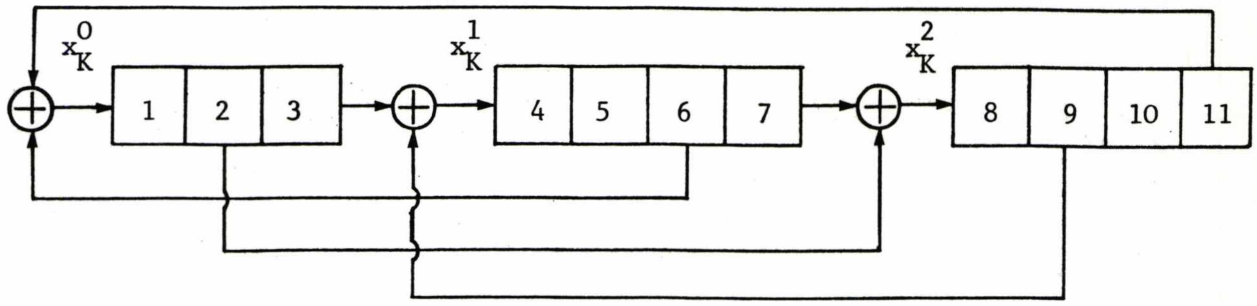
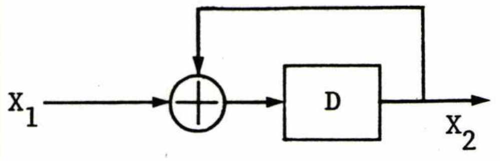


FIGURE 4.5

A toggle flip-flop is equivalent to the following circuit:



D represents a unit of delay

$$X_2 = X_1 + DX_2$$

$$X_2(1+D) = X_1 \qquad \text{or} \qquad \boxed{\frac{X_2}{X_1} = \frac{1}{1+D}}$$

The output X_2 changes with the next clock pulse whenever X_1 is a ONE.

The reason for using toggle flip-flops 3, 7 and 11 is to make the feedback digits (x_K^0 , x_K^1 , x_K^2) depend on more than two digits (so as to avoid trinomial-like recursions, e.g. when the FSR is described by a three term polynomial).

x_K^0 , x_K^1 and x_K^2 can be written as:

$$x_K^0 = \frac{D^4}{1+D} x_K^2 + D^3 x_K^1 \qquad\qquad x_K^0 + D^3 x_K^1 + \frac{D^4}{1+D} x_K^2 = 0$$

$$x_K^1 = \frac{D^3}{1+D} x_K^0 + D^2 x_K^2 \qquad \text{i.e.} \qquad \frac{D^3}{1+D} x_K^0 + x_K^1 + D^2 x_K^2 = 0$$

$$x_K^2 = \frac{D^4}{1+D} x_K^1 + D^2 x_K^0 \qquad\qquad D^2 x_K^0 + \frac{D^4}{1+D} x_K^1 + x_K^2 = 0$$

In order for this system of equations to be indeterminate, one must have

$$P(D) = \begin{vmatrix} 1 & D^3 & D^4/1+D \\ D^3/1+D & 1 & D^2 \\ D^2 & D^4/1+D & 1 \end{vmatrix} = 0$$

Solving for P(D) we get:

$$P(D) = D^{11}+D^{10}+D^9+D^7+D^6+D^3+D^2+D+1$$

or $P(X) = X^{11}+X^{10}+X^{9}+X^{7}+X^{6}+X^{3}+X^{2}+X+1$

which is a primitive polynomial and so generates an m-sequence.

It is obvious that the tap points in the shift register of the example have to be carefully chosen otherwise P(X) might not be primitive and so the resulting sequence would not be of maximal length. At the moment, there are no known systematic ways of finding arrangements that lead to m-sequences, other than trial and error.

Some important properties of pseudo-noise (PN) sequences are now presented.

(1) The randomness properties of PN sequences tend to improve with the degree and complexity of the recursion relation. (That is why trinomials should be avoided).

(2) It is possible to use different (widely shifted) phase-shifts of the same sequence as essentially independent noise sequences.

(3) Statistical dependence is usually not observed unless the recursion relation is a very simple one.

So, with circuits similar to the one just described, one can generate m-sequences using the minimum number of stages of shift-register, and also many different and widely spaced phase shifts are generated after every clock pulse. As pointed out by Hurd those digits in the shift register are "linearly independent and retain the important statistical property that all disjoint subsets considered as binary numbers, are independent and jointly uniformly distributed". This is so because all the 2^n-1 non-zero states occur equally often. Also, the results of tests on the weight distribution indicated that there was no evidence to distinguish between truly random sequences of independent and equally likely bits and the m-sequences generated in the way just described.

The error generator constructed used two m-sequences of relatively prime lengths $2^{11}-1$ and $2^{20}-1$

$$2^{11}-1 = 23\times 89$$

$$2^{20}-1 = 3\times 5^2\times 11\times 31\times 41$$

The primitive polynomial for the $2^{11}-1$ bit sequence is the same as the one we used before, i.e.

$$f_1(x) = x^{11}+x^{10}+x^9+x^7+x^6+x^3+x^2+x+1$$

The way the m-sequence generator for this polynomial is implemented has already been shown. For the implementation of the $2^{20}-1$ bit sequence, the following primitive polynomial was used

$$f_2(x) = x^{20}+x^{17}+x^{16}+x^{14}+x^{12}+x^{11}+x^9+x^8+x^5+x^4+1 \quad .$$

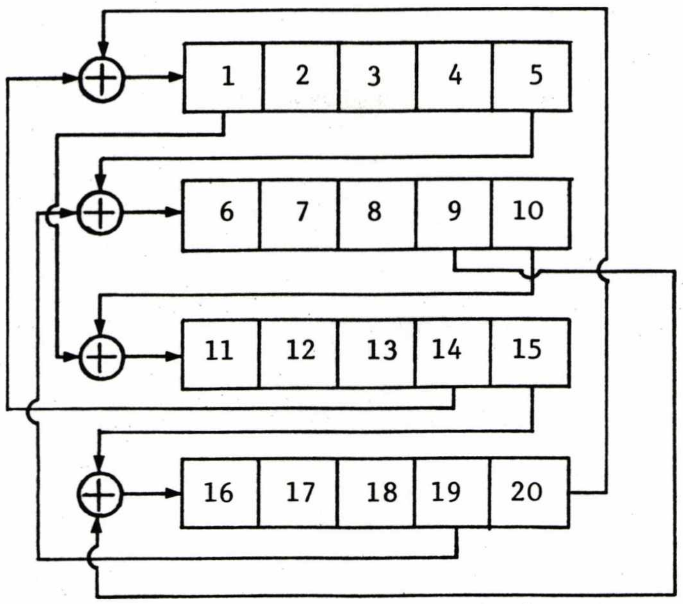


FIGURE 4.6

All flip-flops are D-type except 5, 10, 15 and 20 which are toggle flip-flops.

Twenty-five outputs were arithmetically added using binary full adders and fed to a binary word magnitude comparator. In this way, using a set of switches, we can vary the binary number at the other input to the comparator and obtain at its output the errors according to well defined probabilities. The lowest probability of error (P_e) that can be obtained is $(\frac{1}{2})^{25} \simeq 3\times10^{-8}$.

4.5.2 Generation of Bursts

Figure 4.7 shows a diagram of the circuit used to generate bursts of errors. As can be seen the bursts are triggered by the random error generator. Thus, the rate at which bursts occur is controlled by the threshold settings used in the random error case. The length of the bursts can be varied by adjusting the time constant of the monostable. An m-sequence is used to "fill" the bursts with pseudo-random digits while the flip-flop and gates guarantee that the bursts will always start and end with a one.

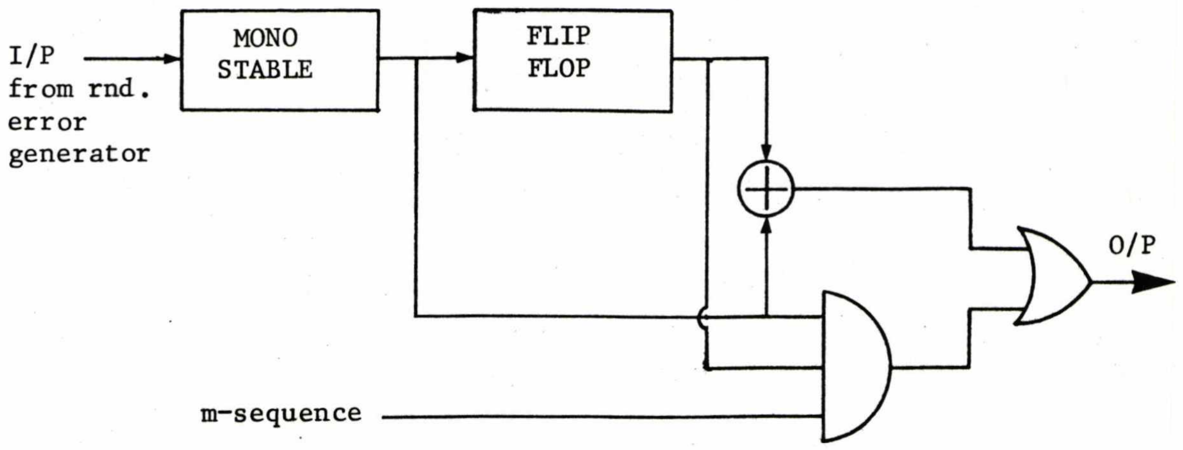


FIGURE 4.7 : Burst Error Generator

4.6 Random Error Correction

To decode random error-correcting cyclic codes an error-trapping (E.T.) decoder was chosen due to its relative simplicity. However, as mentioned before, E.T. suffers from limitations and some means of overcoming those had to be provided, since otherwise the code performance would be degraded. The reason for choosing E.T. when more efficient methods are available lies in the fact that to change from one code to another not many changes are required in an E.T. decoder. This is not generally true of other decoders and the efficiency of E.T. decoding is reasonably good for n and c in the range given above. To complement E.T. when necessary, a process of systematic search was chosen. Again, for n and c in the range given, this is feasible i.e. not too time-consuming and is simple to implement.

4.6.1 Decoder

Figure 4.8 shows a block diagram of the decoder constructed. We shall start by describing how the buffers in the decoder work.

While buffer 1 is being serially loaded with an incoming n-tuple from the channel, its contents are being delivered to the data sink. Simultaneously, the decoder is working on the contents of buffer 2, checking for errors and attempting correction if necessary. After buffer 1 has been loaded with the incoming n-tuple the roles of the two buffers are interchanged. This means that buffer 2 now receives the digits coming from the channel while delivering its contents (corrected codeword) to the data sink. Simultaneously, the contents of buffer 1 are being checked by the decoder.

This process is carried on repeating itself over and over. The number of stages required for each of these buffers is n, the codeword length, which is specified as an address for their respective multiplexers. It should be clear from Figure 4.8 that the multiplexers are used in order to achieve variable wordlength in a simple way.

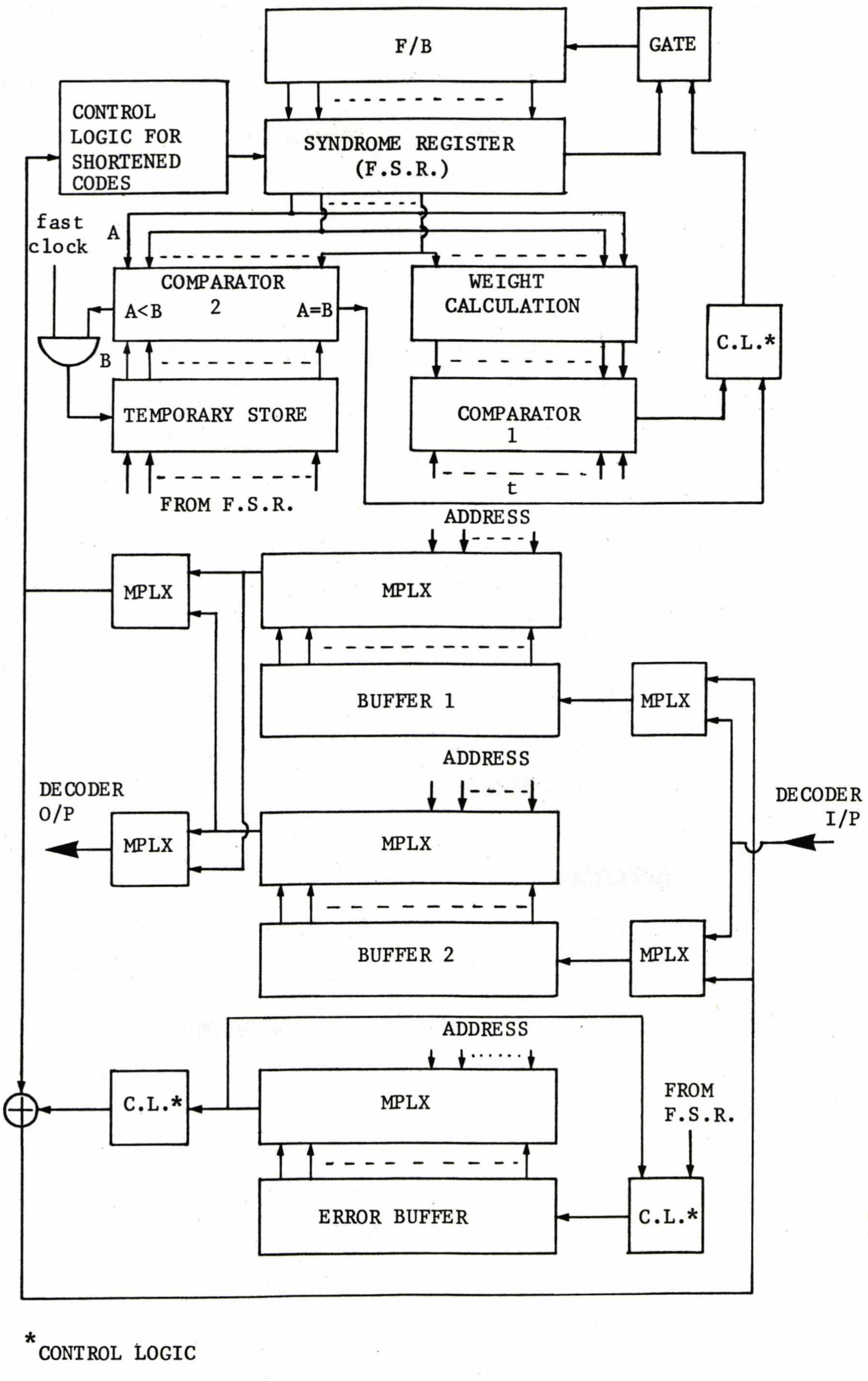
Due to the number of operations the decoder needed to perform during the time duration of one codeword, it had to be provided with a "fast clock". This "fast clock" was 20 times faster than the rate at which digits were received from the channel. When using search this allowed the decoder enough time to go through all its possible error patterns, and be stopped safely before the next codeword was due to be decoded.

4.6.2 Error Trapping

Figure 4.8 can be used to illustrate the decoder section concerned with random error-correction.

Using the fast clock the stored received sequence ready for processing is fed into the syndrome "calculating" register (SCR) and at the same time cyclically shifted in its buffer.* This stored sequence is kept circulating in its buffer while the SCR is shifted trying to trap the error pattern. If E.T. succeeds, then the error buffer (EB), which is an n-stage shift register, is serially loaded with the contents of the S.C.R., which has its feedback connections inhibited. During this time (c fast clock pulses) the fast clock for the stored codeword is inhibited. After that the contents of the E.B. are added modulo-2 to the stored codeword. A timing circuit (not shown) controls the number of shifts required in order to achieve correction and to restore the

* BUFFER 1 OR BUFFER 2



*CONTROL LOGIC

FIGURE 4.8 : General Decoder For Cyclic Codes

original bit positions in the buffer.

If E.T. fails to decode the error then one can choose to search or simply to reset the SCR and associated logic and wait for the next codeword.

4.6.3 Systematic Search

When the decoder is using E.T. plus search, the search process only starts after E.T. has failed. The search process can be described as follows.

First, the syndrome is transferred to a temporary store. After that, the SCR is cleared and the E.B. is loaded with the pattern $X^{c-1}+1$. Now, using the fast clock, the contents of the E.B. are shifted into the SCR and simultaneously cyclically shifted. After the nth shift, an extra clock pulse is inserted in the clock streams for the SCR and E.B. while X+1 is added to both registers. What we just said is equivalent to the following

$$X(X^{c-1}+1) + (X+1) = X^{c}+1$$

The contents of E.B. and SCR are now $X^{c}+1$ and its syndrome respectively. While the fast clock keeps shifting the contents of the codeword buffer, E.B. and SCR, the outputs from the SCR are being compared against the contents of the temporary store, to see if the syndromes agree. If that happens after i shifts $(i<n)$ the contents of the E.B. represent the error pattern responsible for the syndrome match. Correction of the error is then achieved by adding

modulo-2 the contents of E.B. and codeword buffer. This requires n shifts but (n-i) shifts are still necessary in order to restore the initial positions of the digits in the codeword buffer register. This codeword is then ready for delivery to the data sink. In the case that after n shifts no matching of syndromes has been achieved again an extra clock pulse is applied as before to E.B. and SCR and X+1 is added to both in order to get $X^{c+1}+1$ inside the E.B. and its associated syndrome in the SCR.

Now, X^{c+1} can be tried as a possible error pattern and the process is repeated in the same way as before. This process has to be controlled in order to avoid overflow i.e. running out of decoding time. For a worst case situation a fast clock 20 times more rapid than the line clock was found to be sufficient.

After all possible patterns of the form $X^{c+i}+1$ have been tried, and no matching of syndromes has occurred, the contents of the codeword buffer are left unchanged to be delivered to the data sink.

4.6.4 Shortened Cyclic Codes

This section will explain how the decoder buffers operate when shortened cyclic codes are used.

Since the codeword length in a shortened code is (n-i)* this implies receiving and delivering codewords of length (n-i) while processing them with length n since the same

*n is the original word length and i is the number of digits by which it is shortened.

unaltered decoder is used for both shortened and non-shortened codes. This means that the addresses for buffers 1 and 2 (Figure 4.8) need to alternate between n and (n-i) depending on whether they are being used for receiving or decoding codewords respectively. When calculating the syndrome of a received (n-i)-tuple, the input to the SCR is made equal to zero during the first i clock pulses. This is equivalent to preceding each (n-i)-tuple with i zeros which is the same as treating them as n-tuples having the first i positions filled with zeros.

4.7 Burst Error Correction

4.7.1 Encoder

The encoder remains unchanged since the codes used for burst error correction are also cyclic.

4.7.2 Decoder

By means of a switch the decoder constructed can be manually changed from the random error-correcting mode to the burst correcting mode.

This decoder shared some parts of the hardware used for correction of random errors as will be clear from the description that follows. See Figure 4.8.

The first task is to calculate the syndrome of the received n-tuple and store this syndrome in the temporary store. The syndrome register is shifted n times. After every shift the contents of temporary store and syndrome register are compared as binary numbers. Every time the number in the temporary store is greater than the one in the syndrome register a pulse is generated which transfers the contents of the syndrome register to the temporary store. In this way, after n shifts, the temporary store contains the smallest binary number·that appeared during the shifts or in other words the syndrome with the longest run of consecutive zeros. Between two possible bursts of same length the decoder chooses the one with smaller density. The syndrome register continues being shifted until its contents match those of the temporary store. During all this processing the received n-tuple is kept circulating in its respective buffer. The fast clock for this buffer is inhibited, the error buffer serially loaded with the contents of the syndrome register and then added modulo-2 to the contents of the codeword buffer.

A timing circuit controlled the number of shifts required to correct the received n-tuple and leave it ready for delivery.

In terms of number of clock pulses, the time needed by this decoder to operate is:

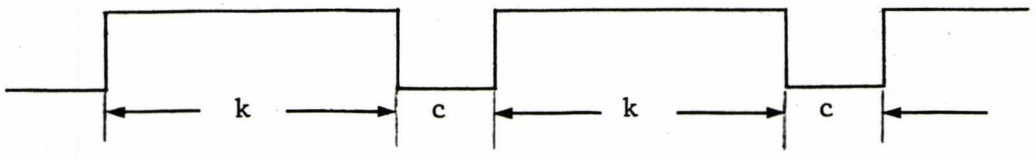
Number of Clock Pulses	Operation
n	Syndrome calculation
n	Find shortest burst
n+c	Loading of error buffer plus correction

Total: 3n+c

4.8 Counting Errors

When the error rate is sufficiently high, some codewords are so badly corrupted that the decoder is unable to correct them. In these cases the data delivered to the sink contain digits in error. In order to count those errors and thus measure the performance of the codes used, the following procedure was adopted. The transmitted codewords were delayed by means of shift register buffers and then compared with their decoded versions by modulo-2 addition. The output from the modulo-2 adder (Figure 4.9) was passed through an AND-gate with the clock, to be able to count consecutive errors. The output from this AND-gate was then fed to a counter (Advance Instruments, timer counter TC14). By the use of external logic it was possible to start and stop the counter automatically, and also to measure bit and block error rate both over the whole codeword and over the information section only.

In the circuit given in Figure 4.9, buffers 1 and 2 are part of the decoder as explained before. Buffers 3 and 4 store the encoded data for comparison and error count. The flip-flop is reset at the beginning of every codeword and any error occurring after that will preset the flip-flop again to one causing the counter to increment its reading by one.



TIMING WAVEFORM

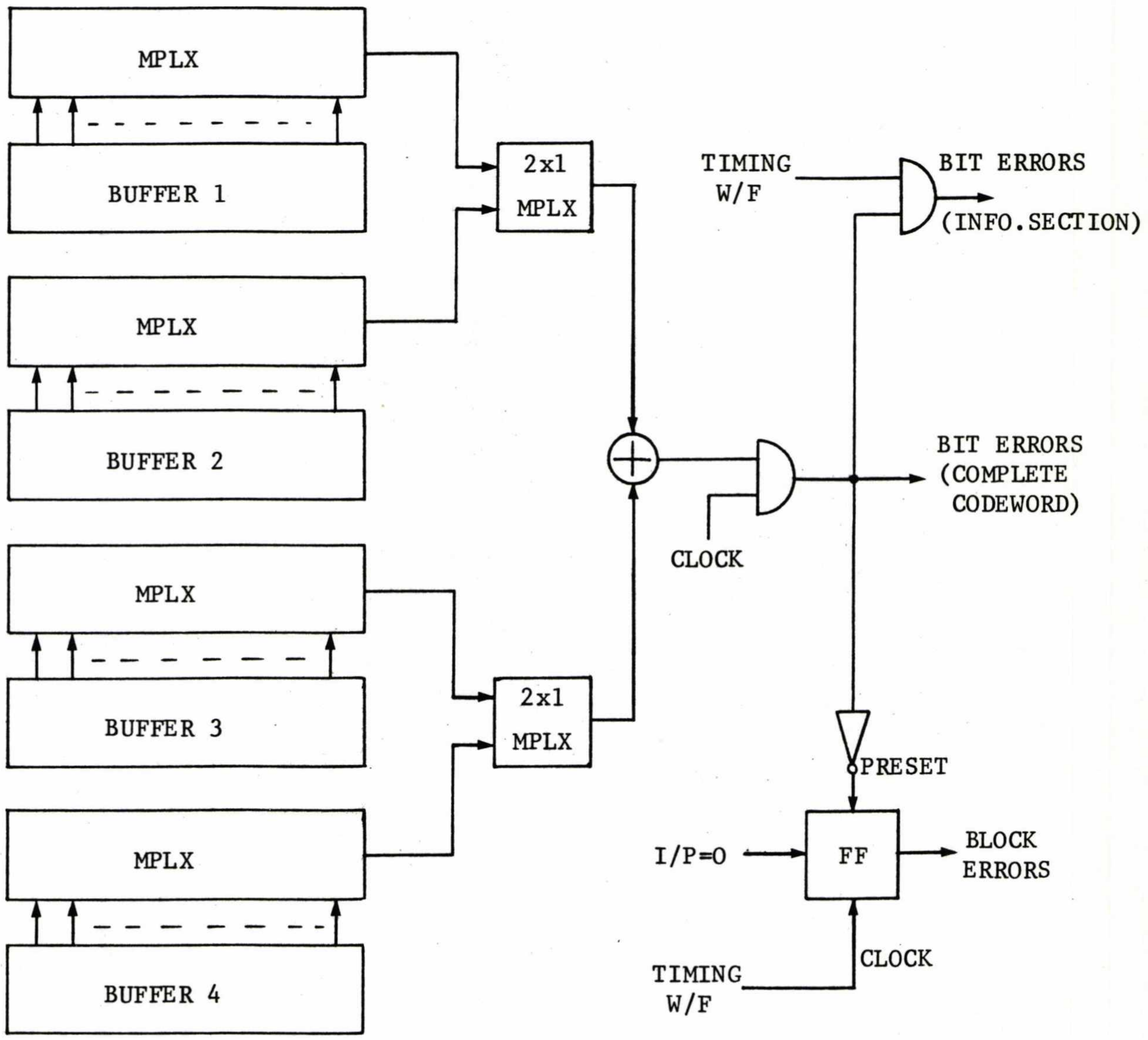


FIGURE 4.9 : Circuit For Counting Errors

4.9 Computer Simulation

The performance of the experimental system was checked by means of a simulation, using the programmes given in Appendix II.

The computer used was a Honeywell DDP 516 and the programmes were written in MINIC language. MINIC is a high level language of which BASIC is a subset. It is also much faster than BASIC and is specially designed (Glover, 1974) to deal efficiently with patterns and maps; i.e. arrays of binary numbers obeying certain rules. In order to generate the errors the computer random number generator was tried but proved unsatisfactory because its probability distribution was found not to be binomial. It was decided then to use a software version of the hardware error generator, and this proved successful. The error generator programme given in the Appendix is actually a simpler version of the one constructed which makes the programme run faster.

Since the syndromes depend only on the error patterns (i.e. are independent of the codewords transmitted) the output from the error generator was fed straight to the decoder. The decoder was provided with information about the code length, t and generator polynomial being used. The probability of error can be adjusted by varying a threshold in the error generator. The programme used for simulation of the decoder for random error correcting codes was easily adapted for the simulation of forward error correction with A.R.Q. Instead of starting systematic search after error trapping has failed, all that is needed is to ask for a repeat and inhibit the error count for that particular codeword.

For the generation of bursts, the random error generator is used to start the burst and the computer random number generator is used to "fill" the burst with ones and zeros.

The average number of bursts of length b will only coincide with the average number of random errors at small values of probability of error ($<10^{-3}$). When the noise level is high, the average number of bursts will be smaller than the average number of random errors because during the duration of one burst more than one random error can occur. This fact has been taken into account when plotting the curves for the performance of burst correcting codes described in the next chapter.

The rate at which bursts occurred was controlled by adjusting a threshold in the random error generator. The burst lengths were made adjustable so that a code could be tested under many different burst conditions.

Though the programmes given in Appendix II are fairly easy to follow with the assistance of the remark statements, extra information has been given on the side whenever it was felt it would increase clarity.

CHAPTER 5

Experimental and Computer Simulated Results

5.1 Introduction

Using the hardware described in the previous chapter, it was possible to investigate the performance of various cyclic codes both in the presence of random (pseudo-random) and burst noise. A computer simulation of the whole system was also made and the results obtained are shown for comparison with the experimental results.

5.2 Cyclic Random Error Correcting Codes

We shall present first the results obtained for the performance of random error correcting cyclic codes in the presence of pseudo-random errors with a binomial probability distribution. The graphs that follow show a plot of the probability of error after decoding (P_c) versus the probability of error in the channel (P_e). Both block and bit output error rates have been plotted because each one has its own interest and one of them alone does not accurately describe the other. P_e is input bit error rate in all cases.

Graphs 1 and 2 show the block and bit error rate respectively for the following single error correcting codes (7,4), (15,11), (31,26). Those are perfect Hamming cyclic codes with generator polynomials and rates given below.

Code	Generator Polynomial	Rate
(7,4)	x^3+x+1	0.571
(15,11)	x^4+x+1	0.733
(31,26)	x^5+x^2+1	0.839

It is interesting to observe that for a probability of error $P_e \leq 10^{-2}$ the curves in graphs 1 and 2 become very nearly parallel and for sufficiently small probabilities of error they are for all purposes parallel. This means, for example, that the (7,4) code gives a reduction in block error rate of about 23.7 against the (31,26) code for $P_e \leq 10^{-2}$. This point is important when trading code rate and block (or bit) error rate.

Graphs 3 and 4 show block and bit error rate for two double error-correcting codes and one triple error-correcting with the following parameters.

Code	t	Generator Polynomial	Rate
(15,7)	2	$x^8+x^7+x^6+x^4+1$	0.467
(31,21)	2	$x^{10}+x^9+x^8+x^6+x^5+x^3+1$	0.677
(15,5)	3	$x^{10}+x^8+x^5+x^4+x^2+x+1$	0.333

It is seen that the curves for codes with different t have different slopes and as a consequence of this the advantage to be gained is a function of P_e. For example, if we compare the codes (15,5) and (31,21) on graph 3, for a reduction in code rate of about 50% we obtain an improvement factor of around 500 for the block error rate at $P_e = 10^{-2}$ and this improvement keeps increasing as P_e decreases while the code rate remains constant.

Graphs 5 to 10 show pairs of codes having approximately the same rate (k/n). It is clearly seen that the longer codes have better error-correcting properties and that the curves normally cross over at a noise error rate close to 10^{-1}. This is a relatively high probability of error since in many practical applications P_e is $\leq 10^{-3}$ except during bad fading or bursty conditions. A definite advantage is thus obtained by using longer codes. The limit to the code length to be used will be set by the cost and complexity of the particular coding scheme.

5.2.1 Shortened Cyclic Codes

Various shortened cyclic codes were tested and the results obtained are well described by graphs 11 and 12 where the (15,11) S.E.C. code has been chosen. It can be seen that only a relatively small reduction in probability of error after decoding is obtained as a result of shortening, i.e. even with quite a severe reduction in efficiency (k/n). For this particular (15,11) code the block error rate is reduced 18 times when it is shortened to (5,1). In terms of code rate this represents a reduction from .733 to .200. It should be noted that with a repetition code (5,1) one is able to correct double errors inside a block while with the (5,1) shortened code mentioned above, the error-correcting capability remains t=1 if the same decoder for the (15,11) code is used preceding each word by 10 zeros. In this sense, both code efficiency and error-correcting power are being lost. However, in many applications where the original code can be decoded with simple equipment, it can be advantageous to stick to this decoder rather than to design a special type where the syndromes are difficult to relate to error patterns and/or a lot of parallel computation is required.

5.2.2 Error Trapping Plus Systematic Search

Graphs 13 and 14 illustrate the advantage of using search to complement error trapping.

The (31,21) t=2 cyclic code was used here to compare the results of decoding by error trapping alone and when, in addition, search is used. For $P_e = 10^{-3}$ a reduction of around 30 times is obtained for block error rate when search is used and this improvement increases as P_e decreases.

So, whenever error trapping alone is not sufficient to fully decode the code, one should consider the option of complementing the decoding with search before considering another decoding process altogether.

For the (31,21) code mentioned above, the following error patterns and their cyclic shifts could not be trapped: $X^{10}+1$, $X^{11}+1$, $X^{12}+1$, $X^{13}+1$, $X^{14}+1$, $X^{15}+1$.

Generally, if the search can be done sequentially, the amount of hardware required is considerably reduced and the limiting factor will be the maximum speed at which the circuit components used can work.

Another code that required search was the (17,9) t=2 cyclic code generated by $G(X) = X^8+X^5+X^4+X^3+1$. In this case, the decoder needed to use search only for the X^8+1 pattern and its cyclic shifts.

5.3 Cyclic Burst-Error-Correcting Codes

For the study of burst error-correcting codes in a bursty channel, it is necessary to know the following characteristics of the channel:

(1) Frequency of occurrence of bursts.

(2) Probability distribution of burst length.

(3) Distribution of number of errors in a burst; i.e. distribution of burst density.

The codes studied did not show any sensitivity to burst density since, as explained before, all bursts of length $\leq b$ are corrected and those bursts longer than b which can be corrected are not selected on a burst density basis.

Instead of using a particular probability distribution of burst length it was decided to obtain results for various fixed values of b. Graphs are plotted showing block and bit output error rates against input burst probability, P_b. These results can be combined by using appropriate values for coefficients satisfying a given distribution of burst length, and the performance of the code in a given environment can thus be determined.

Graphs 15 and 16 show block and bit error rate results obtained with the codes:

(7,3) b=2 $G(x) = x^4+x^3+x^2+1$

(14,6) b=4 $G(x) = x^8+x^6+x^4+1$

The (14,6) code was obtained by interleaving the (7,3) code to degree 2.

It is interesting to observe in those graphs that the (7,3) code deteriorates much faster than the (14,6) code when bursts longer than b occur. The reason for that lies in the fact that the (7,3) code uses its redundancy very efficiently to correct the bursts of length ≤ 2 and is left with no "spare" redundancy to cope with longer bursts. On the other hand, the (14,6) code after allowing for all bursts of length ≤ 4 is left with $2^8-1-14\times 2^3 = 143$ unused syndromes which are then used by the decoder to correct bursts longer than 4. This (14,6) code corrects 62.5% of the bursts of length 5 and 25% of the bursts of length 6.

Graphs 17 and 18 show block and bit error rate for the codes:

(15,10) b=2 $G(x) = x^5+x^4+x^2+1$

(30,20) b=4 $G(x) = x^{10}+x^8+x^4+1$

The results are very similar to the ones just discussed. However, these codes are longer. The codes have the same rate since the (30,20) code resulted from interleaving the (15,10) code to degree 2.

Peterson, 1972 (p.363), shows that random error-correcting codes can be used to correct bursts of errors. When used for correcting single bursts the lower bound on b is given by:

$$b \geq \frac{3d-8}{4}$$

where d is the minimum distance of the code.

A few random error-correcting codes were tried on the computer for correction of bursts of errors and the results are given below.

n,k,t	b	$\frac{3d-8}{4}$	z = c-2b
15,7,2 BCH	4	7/4	0
15,5,3 BCH	5	13/4	0
17,9,2 BCH	3	7/4	2
23,12,3 Golay	5	13/4	1
23,11,3 Golay modified	6	4	0
31,21,2 BCH	4	13/4	2

TABLE 1

It can be seen that some well established random error correcting codes are optimum (z=0) when used for burst correction. Such codes, however, cannot be used to correct both burst and random errors because in their coset decomposition, one finds "correctable" bursts and "correctable" random error patterns in the same coset.

5.4 Forward Error Correction with A.R.Q.

Some of the random error-correcting codes, mentioned above, are studied in this section and used as hybrid ED/EC codes in combination with Automatic Repeat Request (A.R.Q.), and the results obtained are presented in graphs 19 to 26.

To use A.R.Q. it is necessary to have a feedback channel, and to have some means of storage at the transmitter. The results presented here are ideal in that a noiseless feedback channel, together with infinite buffer storage at the transmitter, are assumed.

From the graphs mentioned above, it can be seen that the probability of error after decoding is substantially reduced as the codes are used to correct less and to detect more errors. As the error-detecting power of these codes is increased, an increase in the number of repeat requests is observed. The percentage of retransmitted blocks is shown on the graphs for block error rate. For $P_e \geq 10^{-1}$ the efficiency of the channel is very low since the number of retransmissions then becomes very large.

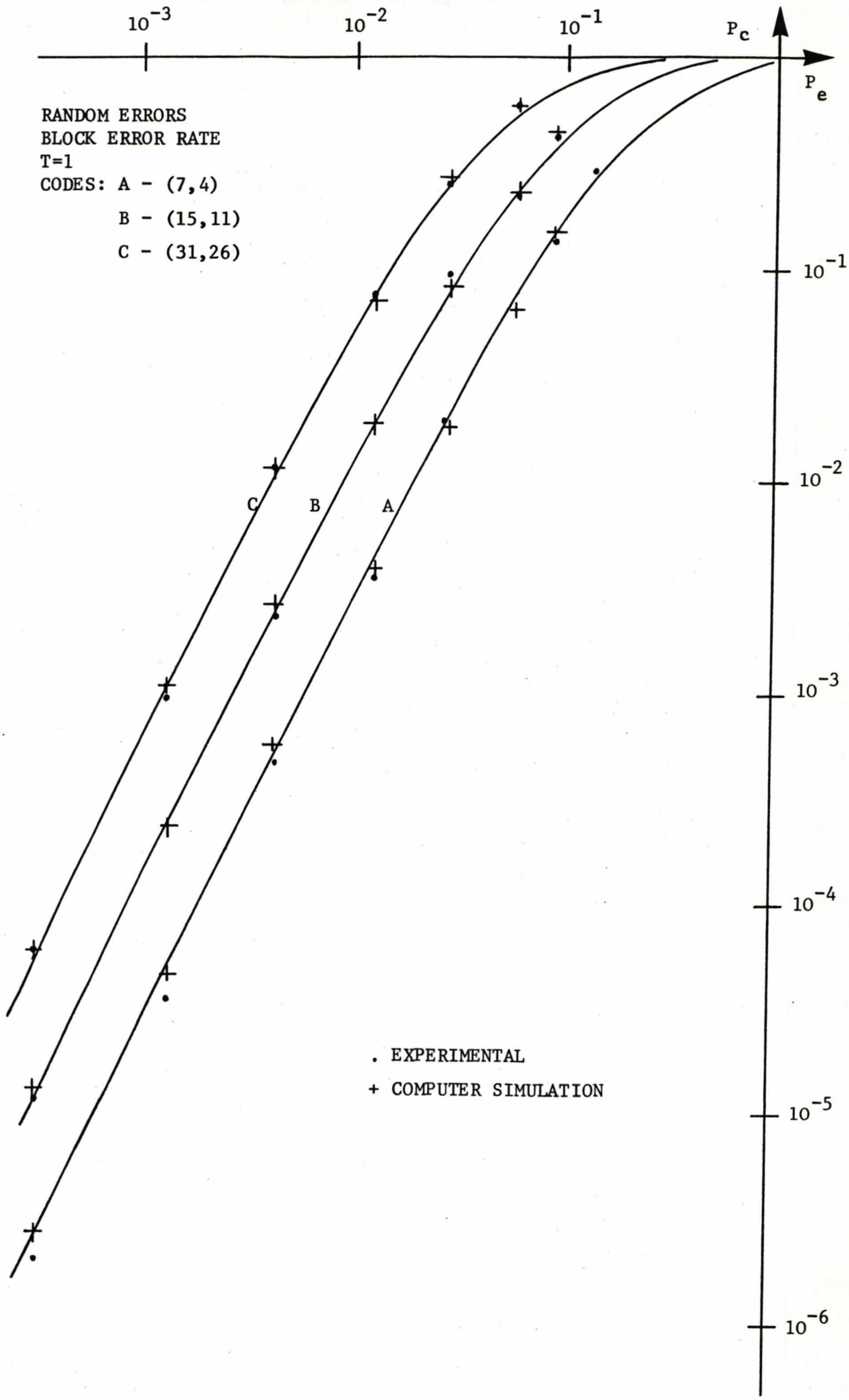
Error detecting codes (t=0) become very practical in terms of efficient use of the channel for values of $P_e \leq 10^{-3}$ since the number of repeat requests drops to about 1 in 10^3 or less.

The use of combined error-trapping and A.R.Q. effectively transforms the (15,10)d = 4(single error-correcting double-error detecting code) into a double error-correcting code which is both efficient and easy to decode. Another interesting point is the possibility of correcting some patterns of errors of weight t+1 when using quasi-perfect codes. The (15,7) t=2 code shown in graphs 19 and 20 has a block error rate 10 times smaller (on average) than when decoded by error trapping alone. This is so because on the forward error-correcting mode normally more than half of the total number of syndromes available are not used at all (they correspond to weight t+1 error patterns).

5.5 Possible Sources of Inaccuracy in the Results

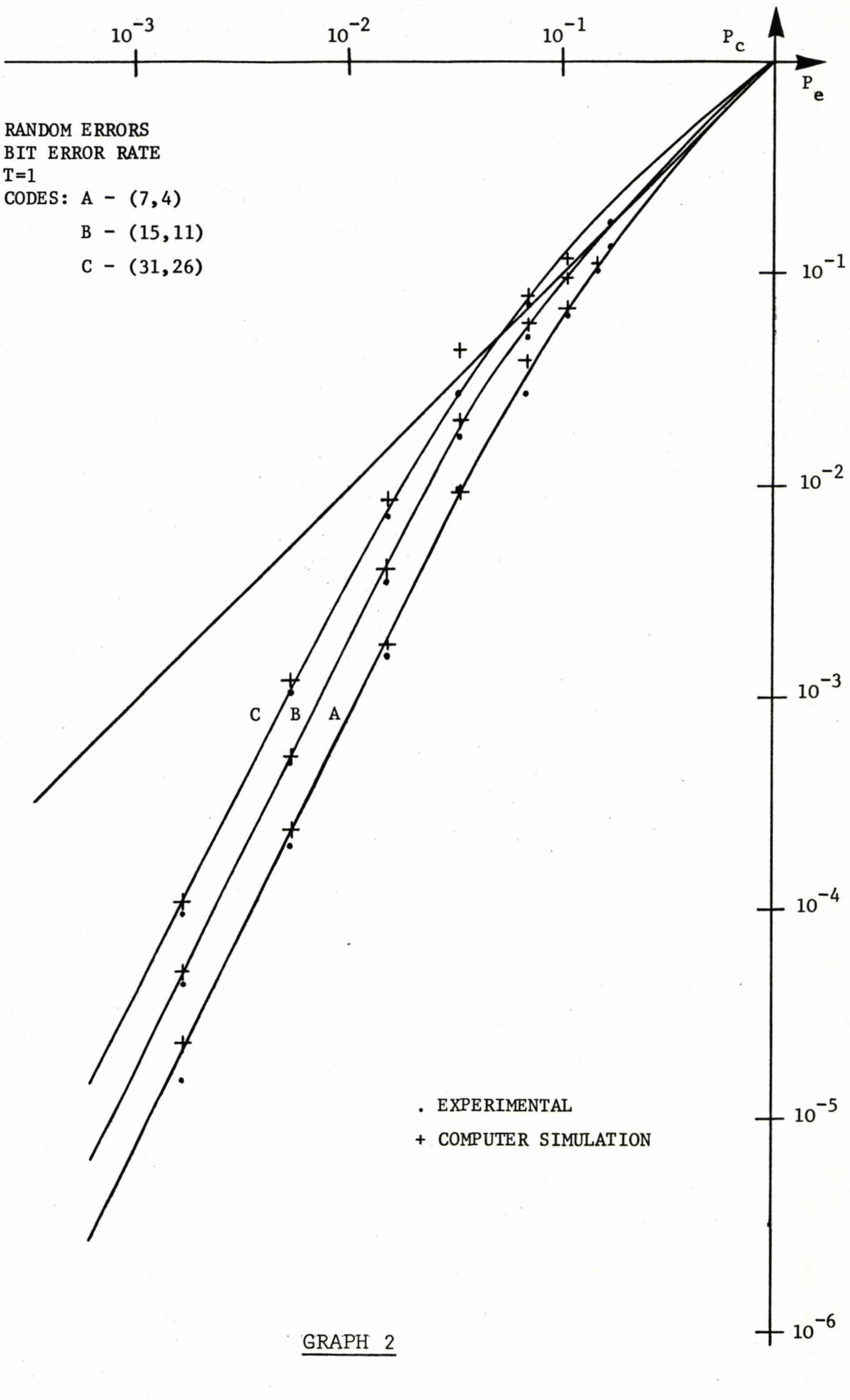
We were faced with some difficulties when obtaining experimental results for low error rates of the order of 1 in 10^6. This was caused by transient noise in the form of spikes coming from the mains supply and was eliminated by decoupling the power rails with capacitors and adjusting a threshold in the counter. In this way, only pulses exceeding a given duration and height were counted.

For the computer simulation we did not use the computer random number generator as the random error source because its probability distribution is arbitrary and we needed a binomial distribution. Thus, an error generator of the type constructed in hardware was simulated in software. In order to keep the computer programme within a manageable size, and at the same time not too slow, a shorter m-sequence was used. This is a possible source of inaccuracy, particularly at low error rates. Contrasting with what has been said about difficulties in getting results experimentally and by computer simulation, it is relatively easy to predict the code performance at low error rates once a sufficient number of points have been obtained. This is so because the curves P_c versus P_e, when plotted on a log × log graph, tend to straight lines for $P_e < 10^{-3}$.



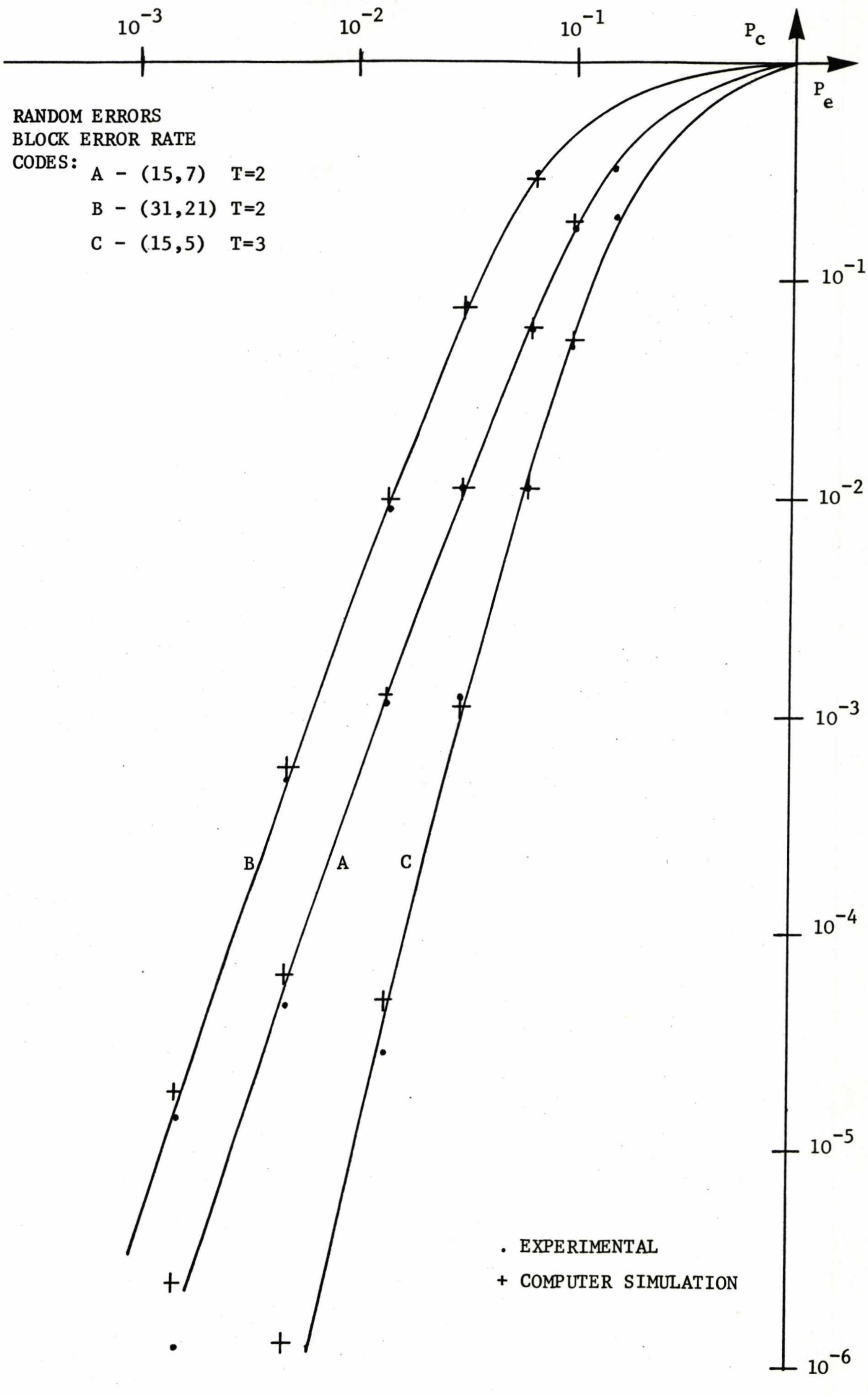
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
T=1
CODES: A - (7,4)
B - (15,11)
C - (31,26)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
C
B
A
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 1



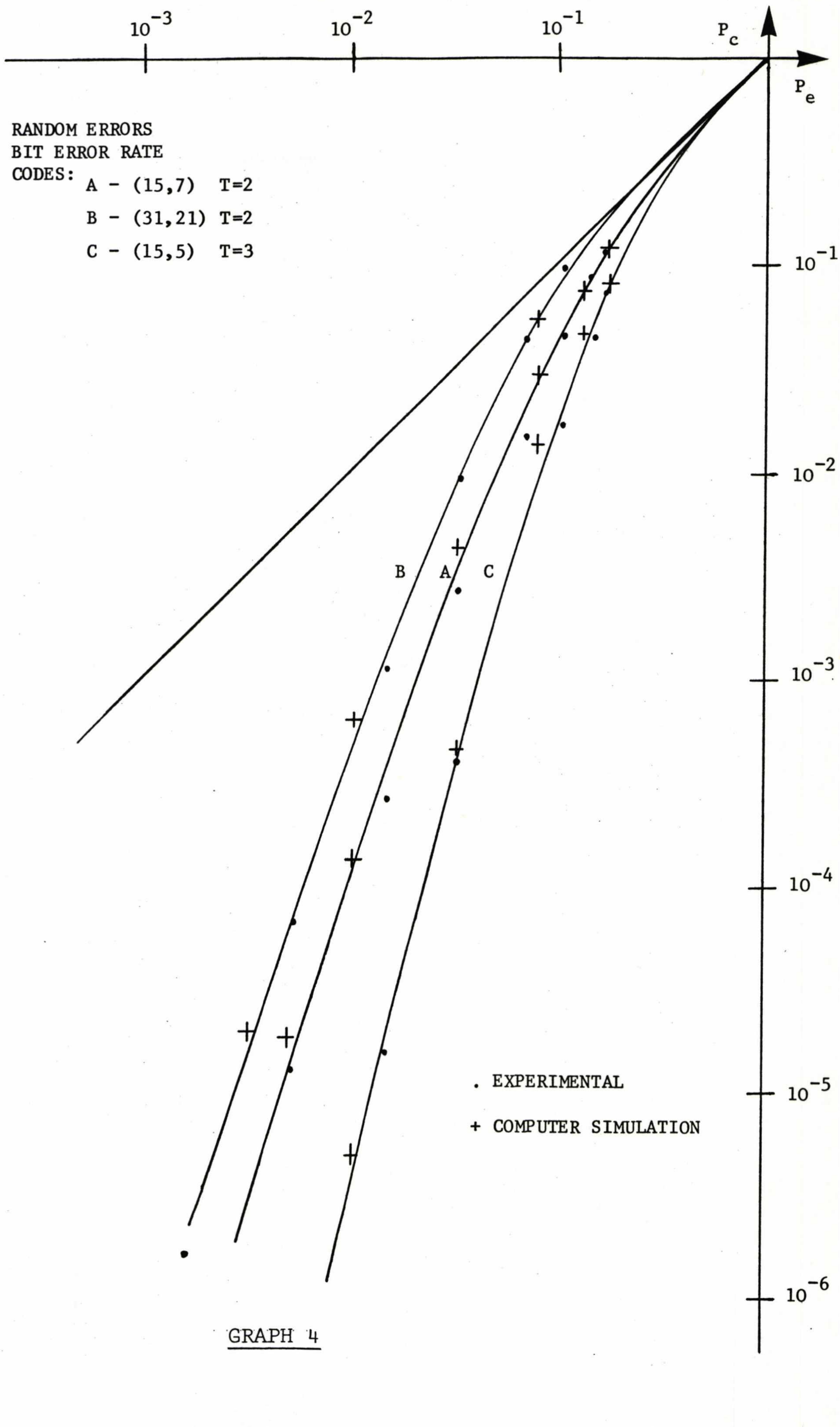
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
T=1
CODES: A - (7,4)
B - (15,11)
C - (31,26)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
C
B
A
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 2

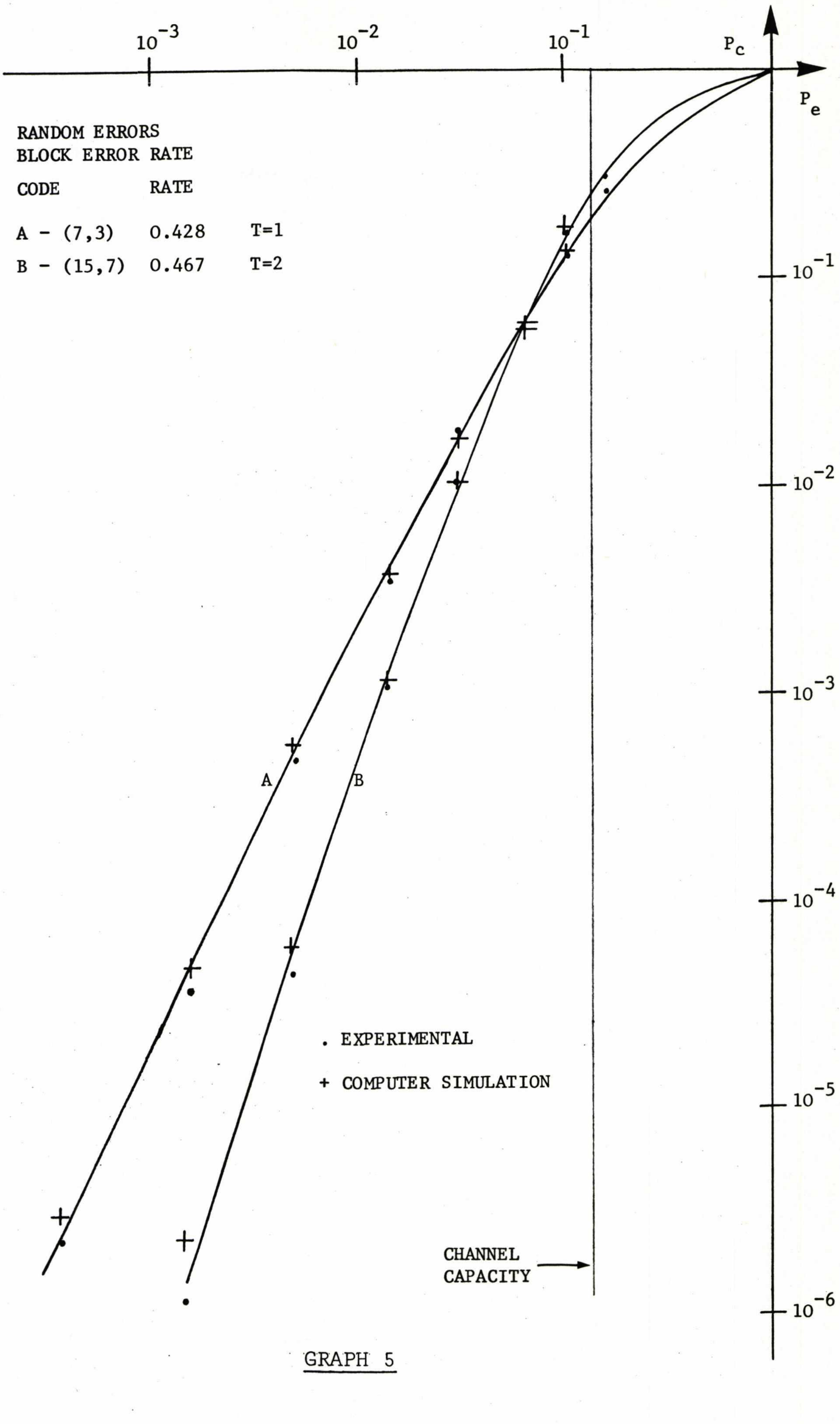


10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODES: A - (15,7) T=2
B - (31,21) T=2
C - (15,5) T=3
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
B
A
C
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 3

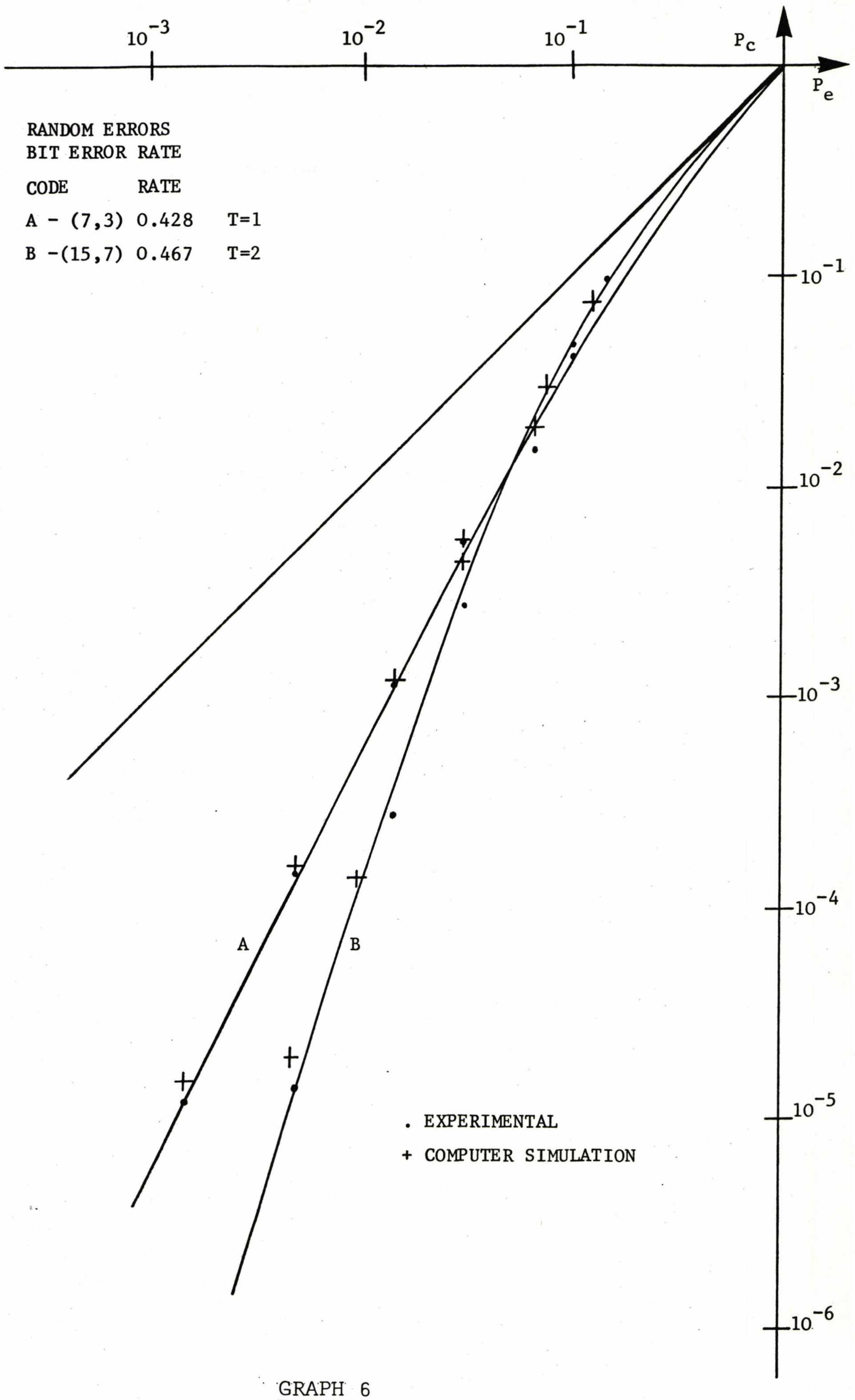

10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
CODES: A - (15,7) T=2
B - (31,21) T=2
C - (15,5) T=3
10^{-1}
10^{-2}
B
A
C
10^{-3}
10^{-4}
. EXPERIMENTAL
+ COMPUTER SIMULATION
10^{-5}
10^{-6}

GRAPH 4



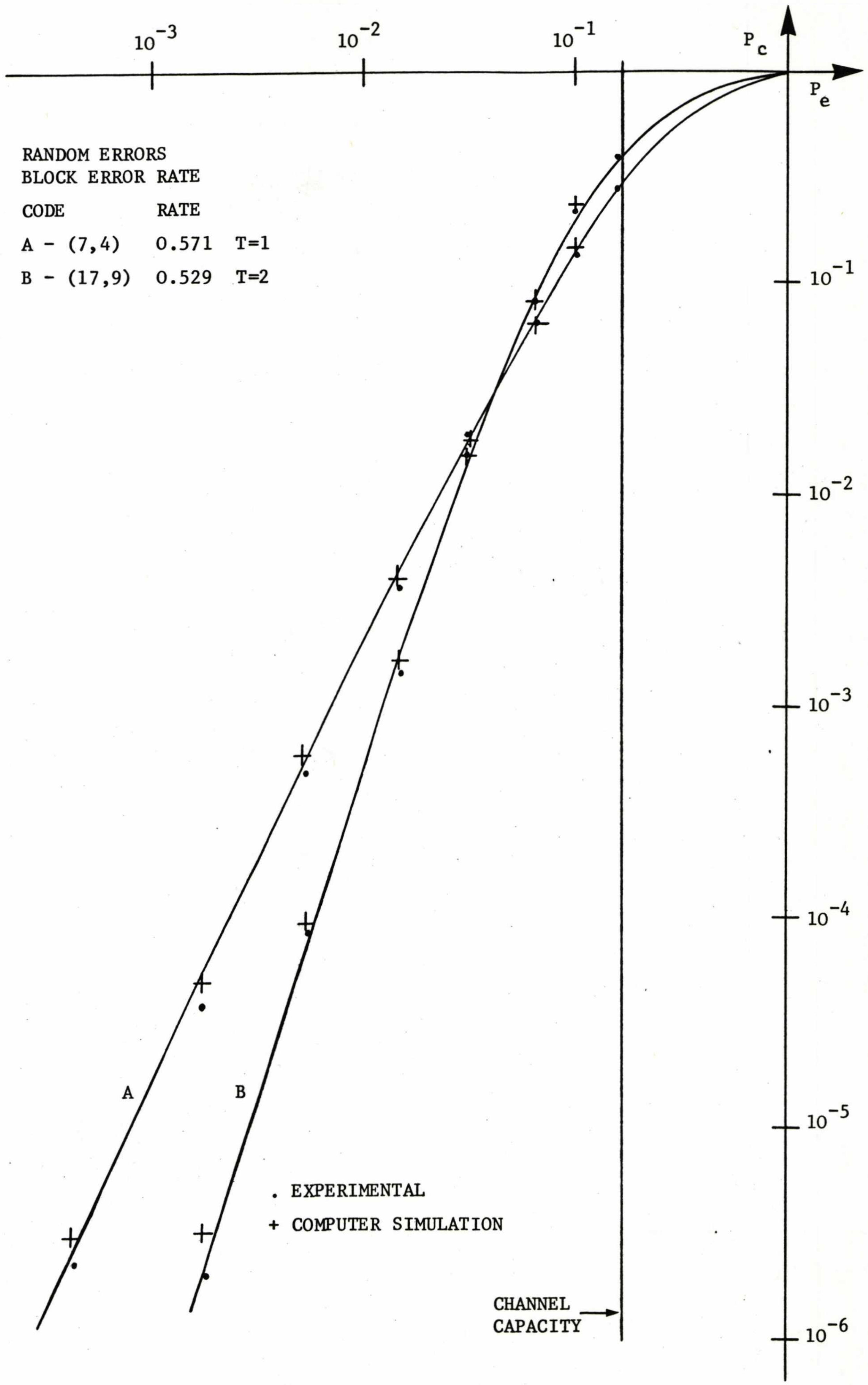
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODE RATE
A - (7,3) 0.428 T=1
B - (15,7) 0.467 T=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION
CHANNEL CAPACITY

GRAPH 5



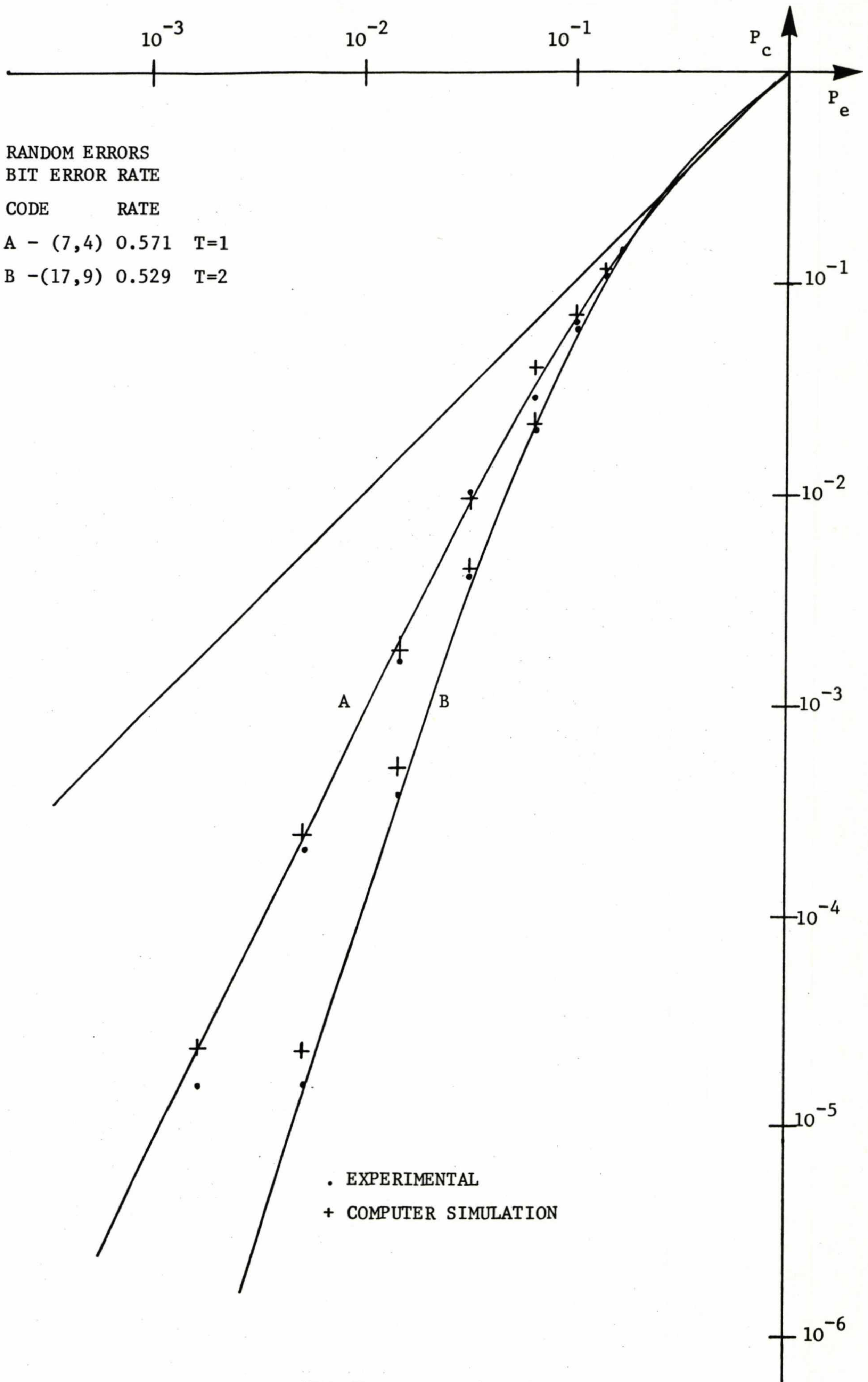
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
CODE RATE
A - (7,3) 0.428 T=1
B -(15,7) 0.467 T=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 6



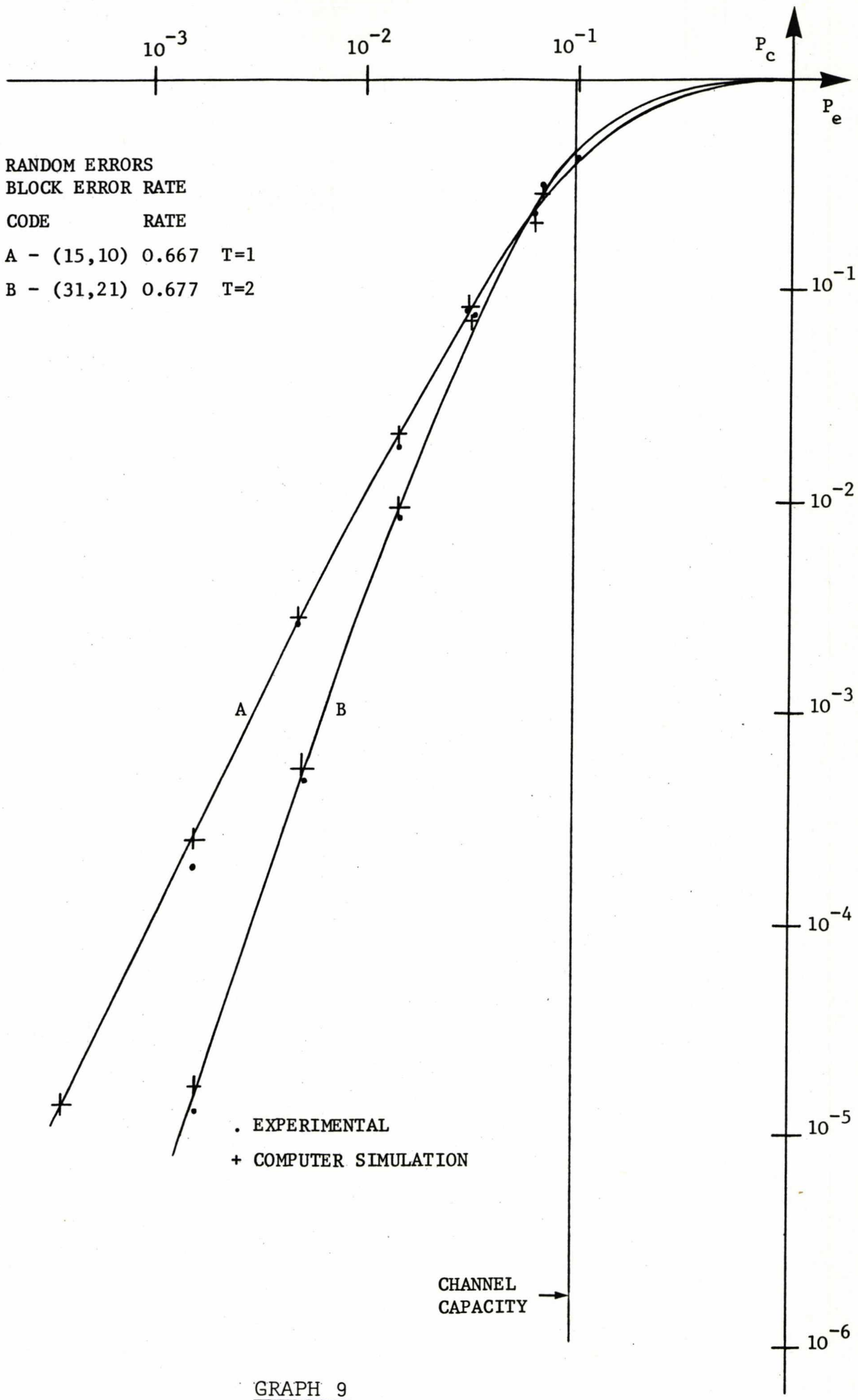
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODE RATE
A - (7,4) 0.571 T=1
B - (17,9) 0.529 T=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION
CHANNEL CAPACITY

GRAPH 7

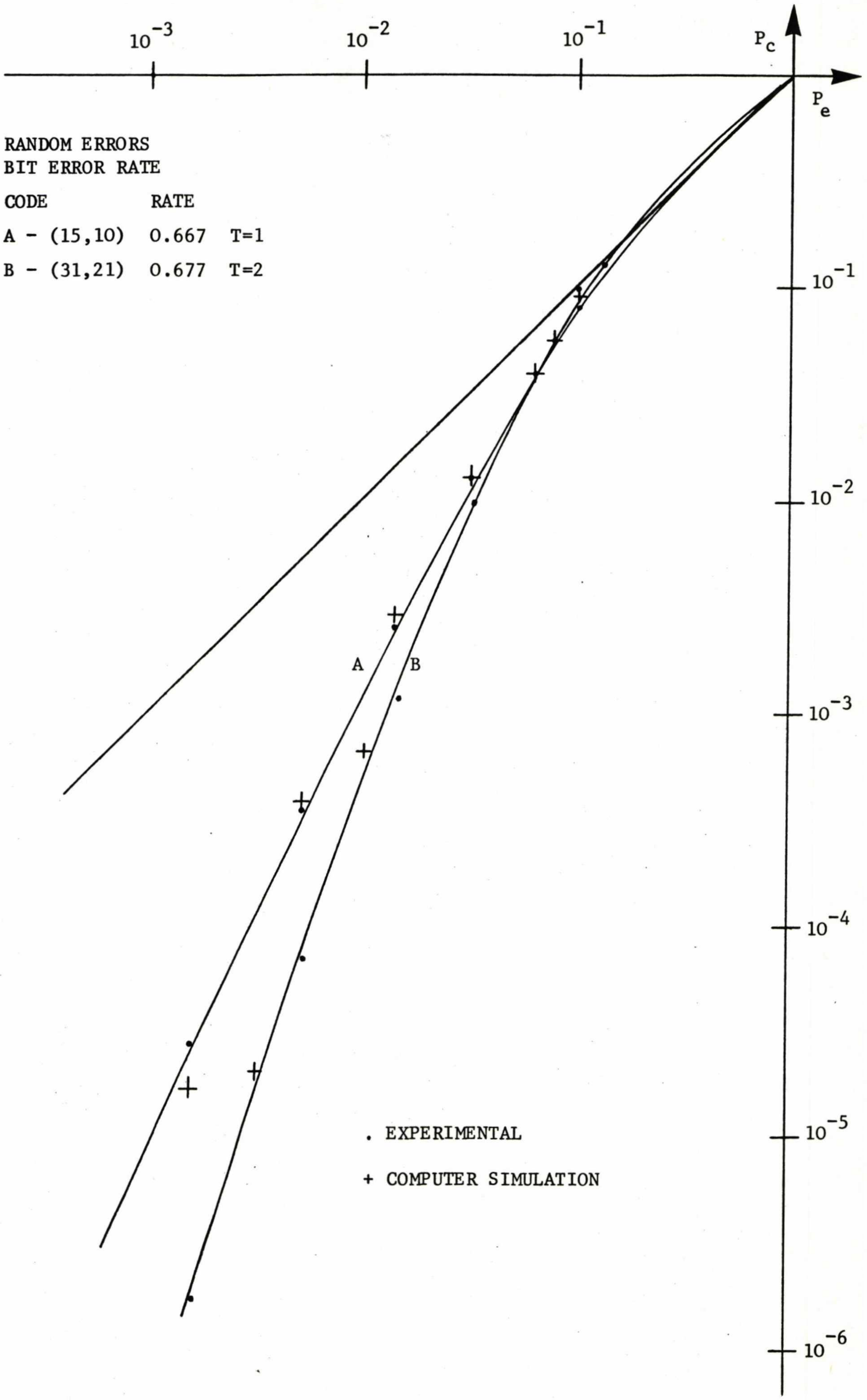


10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
CODE RATE
A - (7,4) 0.571 T=1
B -(17,9) 0.529 T=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 8

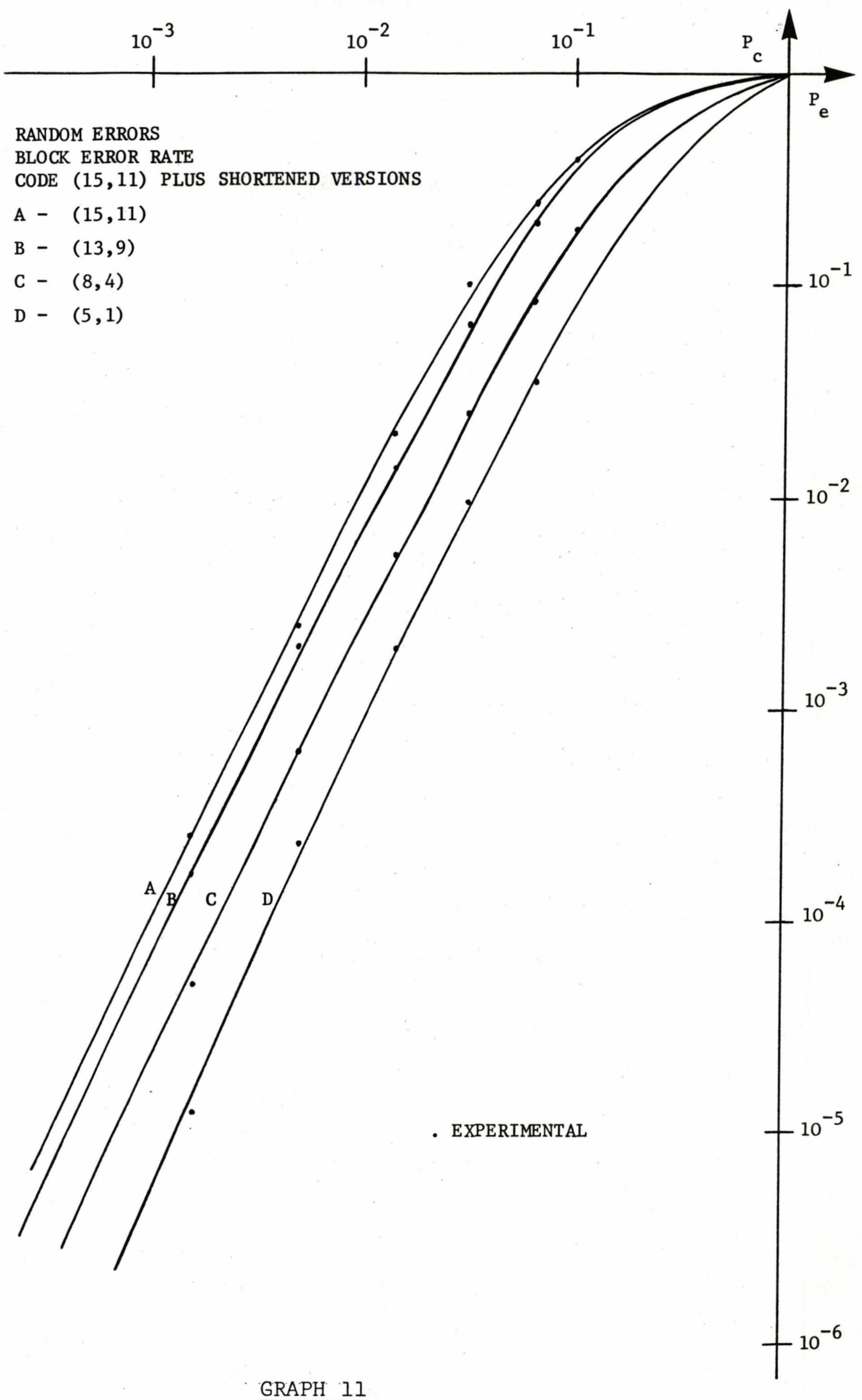

10^-3
10^-2
10^-1
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODE RATE
A - (15,10) 0.667 T=1
B - (31,21) 0.677 T=2
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION
CHANNEL CAPACITY

GRAPH 9



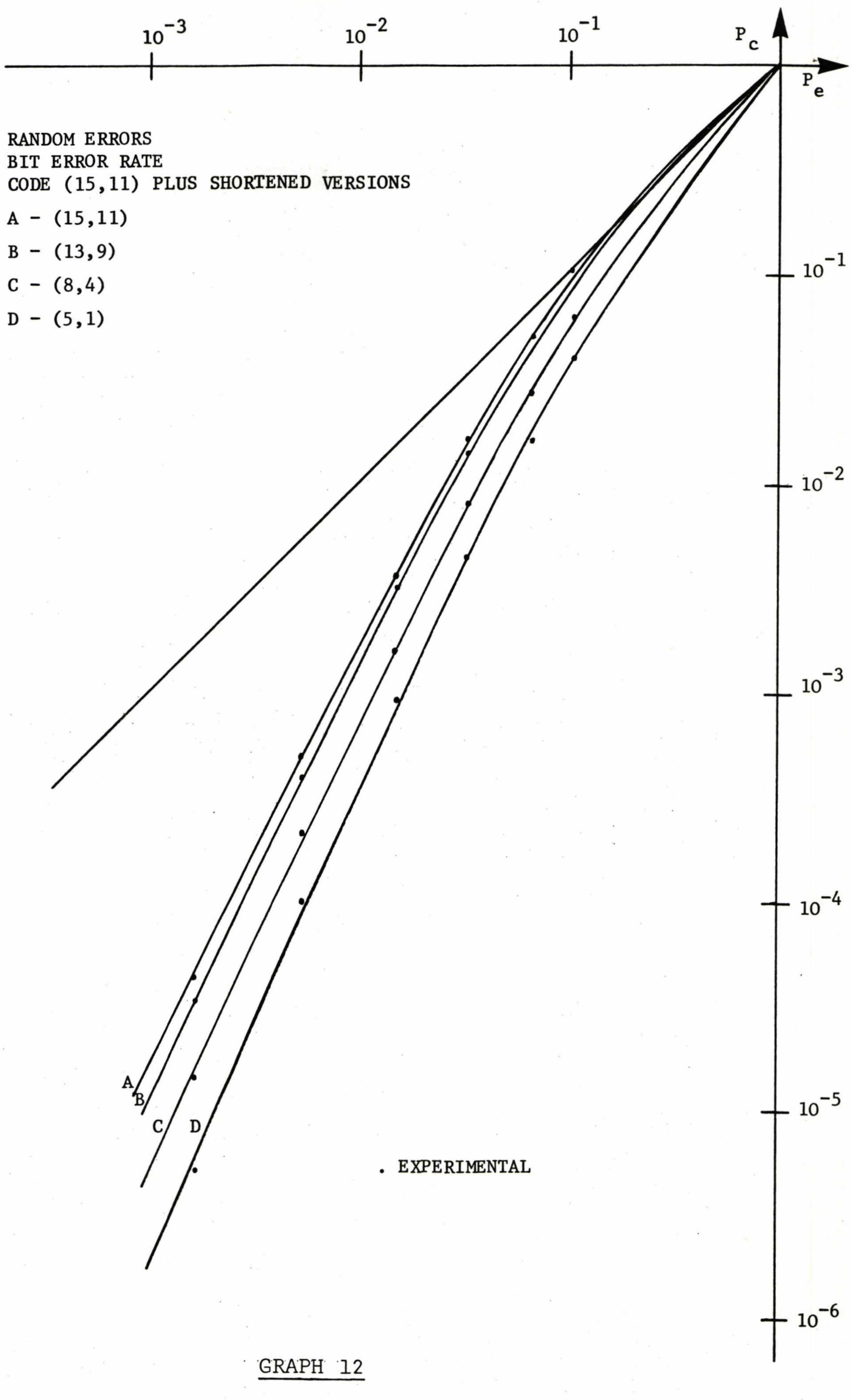
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
CODE RATE
A - (15,10) 0.667 T=1
B - (31,21) 0.677 T=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 10



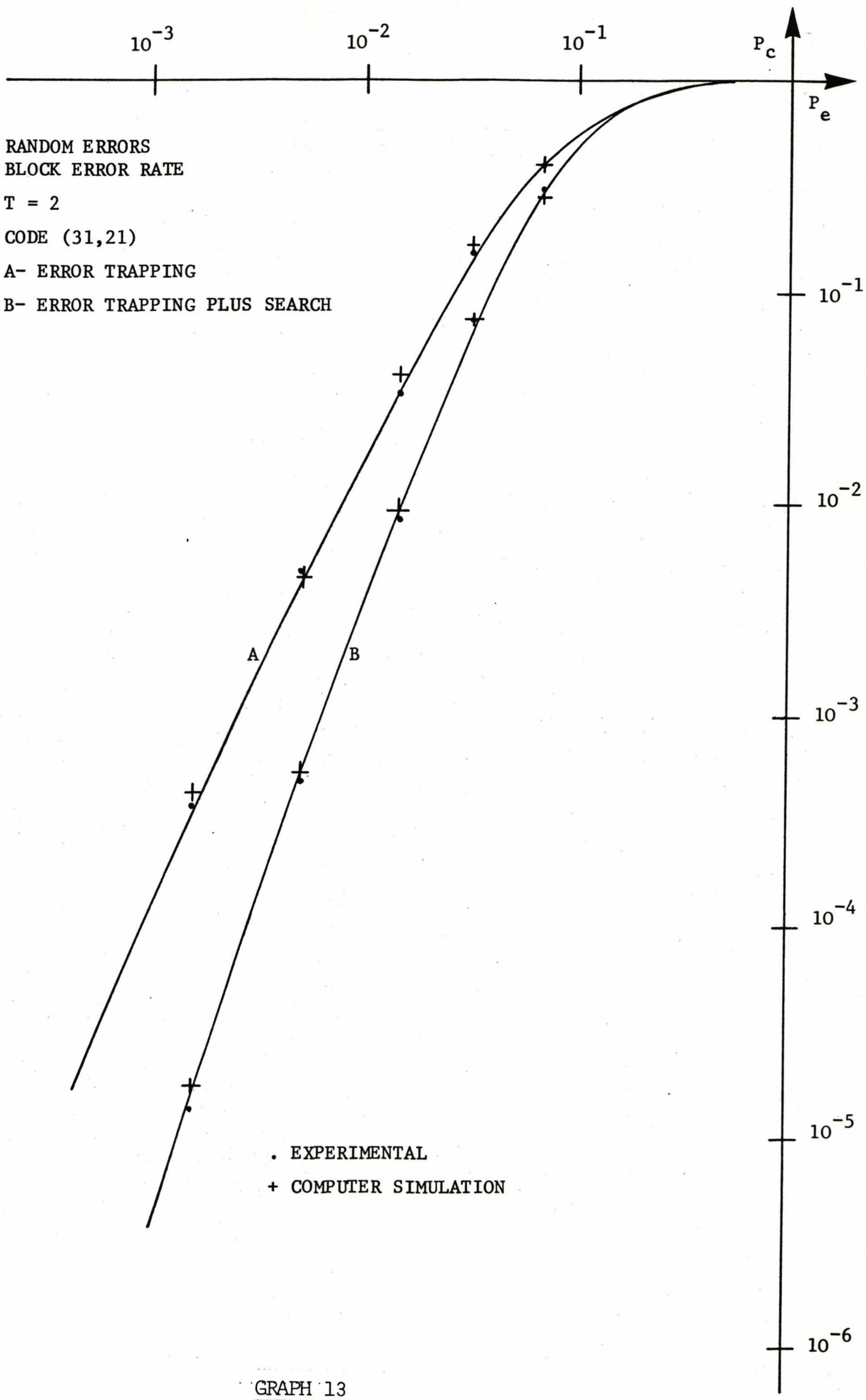
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODE (15,11) PLUS SHORTENED VERSIONS
A - (15,11)
B - (13,9)
C - (8,4)
D - (5,1)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
C
D
. EXPERIMENTAL

GRAPH 11



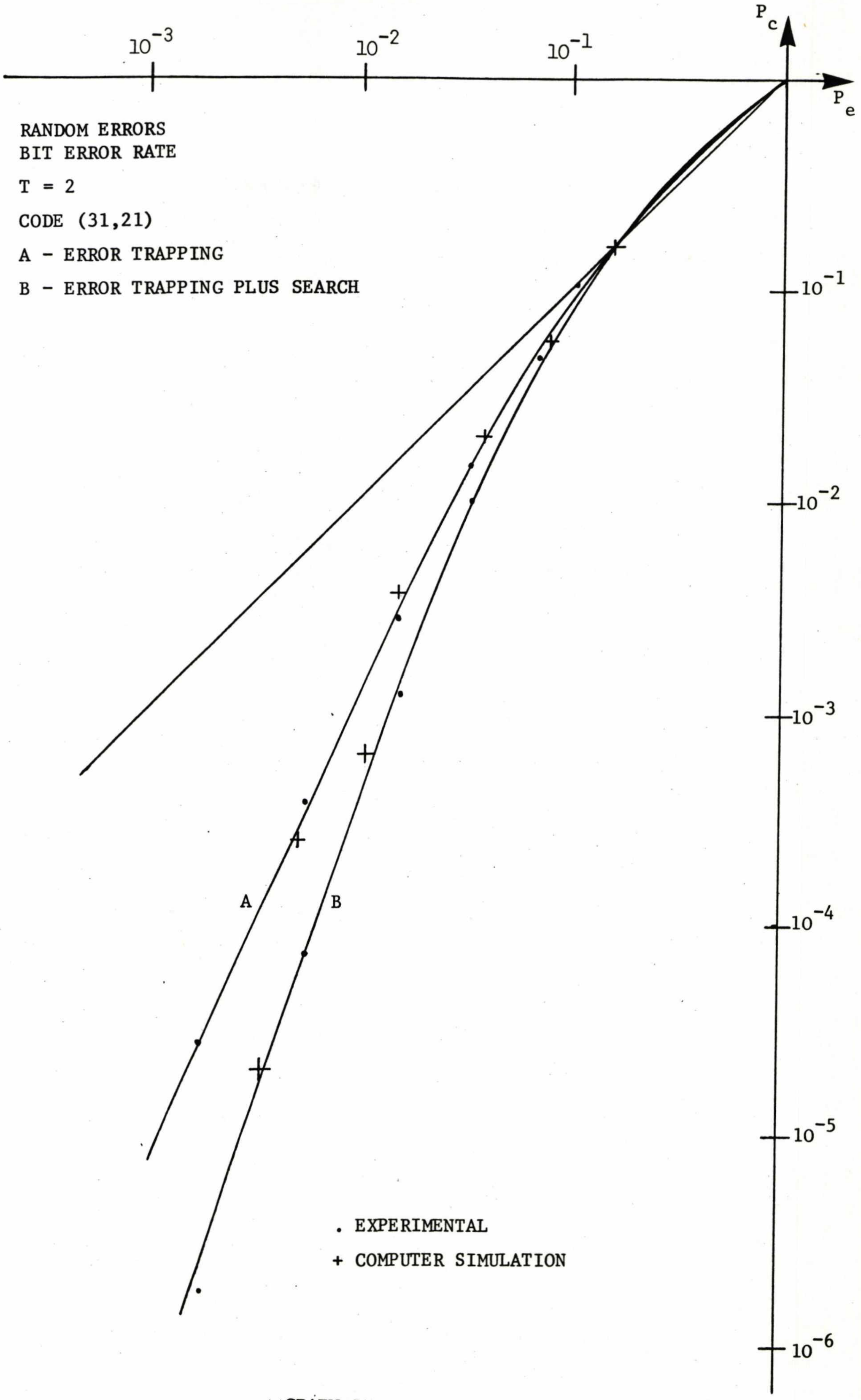
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BIT ERROR RATE
CODE (15,11) PLUS SHORTENED VERSIONS
A - (15,11)
B - (13,9)
C - (8,4)
D - (5,1)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
C
D
. EXPERIMENTAL

GRAPH 12



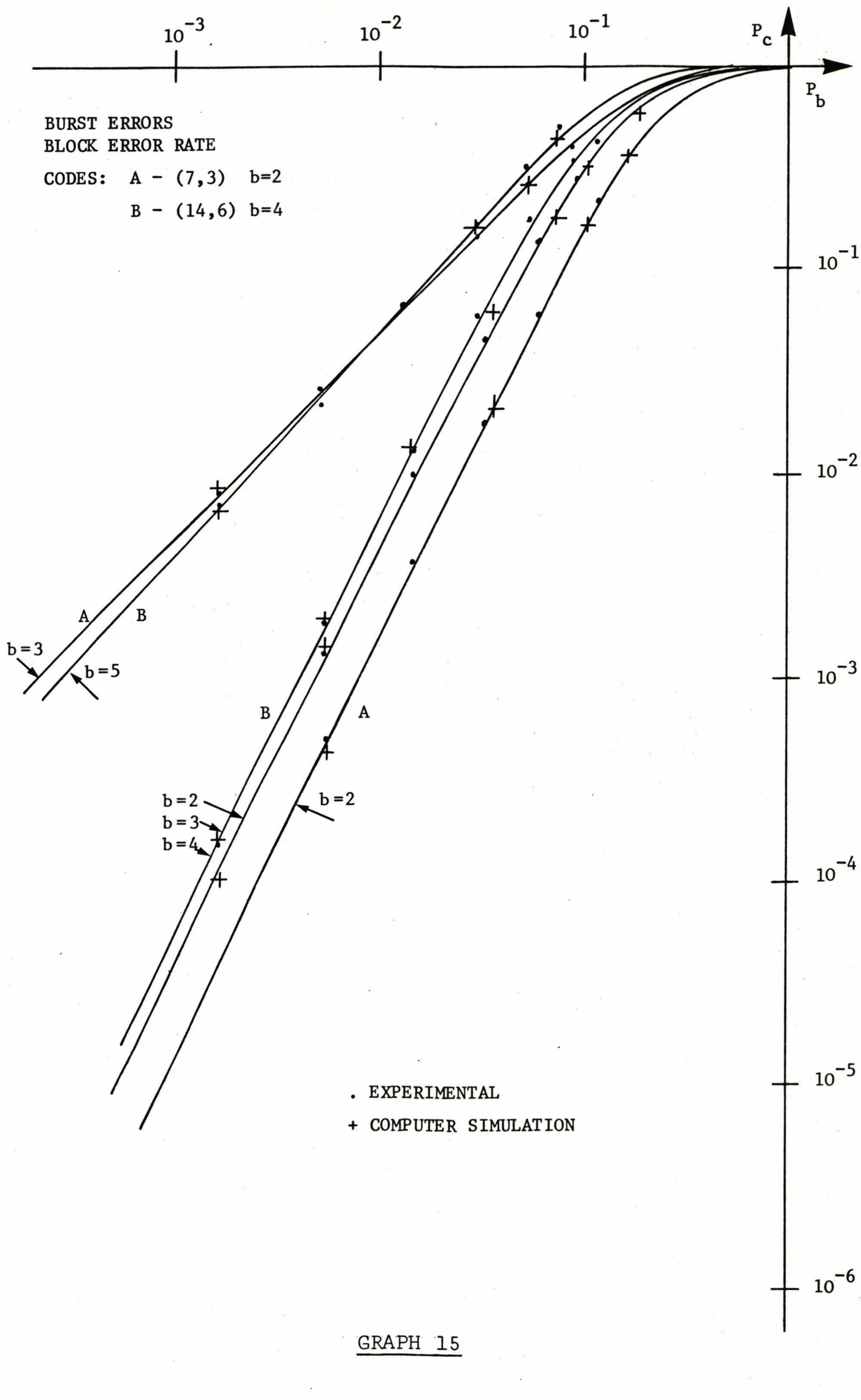
10^{-3}
10^{-2}
10^{-1}
P_c
P_e
RANDOM ERRORS
BLOCK ERROR RATE
T = 2
CODE (31,21)
A- ERROR TRAPPING
B- ERROR TRAPPING PLUS SEARCH
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 13



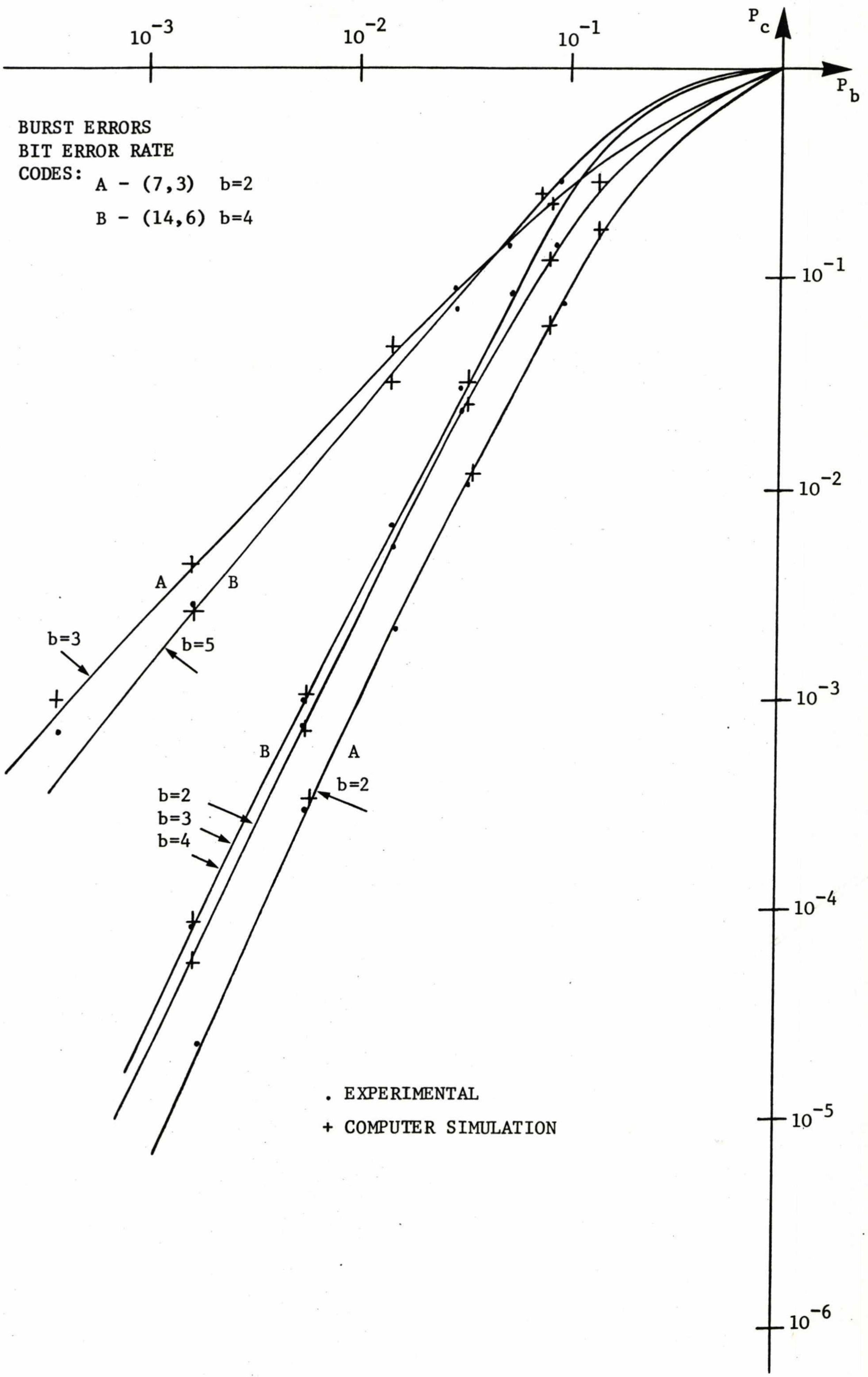
P_c
P_e
10^{-3}
10^{-2}
10^{-1}
RANDOM ERRORS
BIT ERROR RATE
T = 2
CODE (31,21)
A - ERROR TRAPPING
B - ERROR TRAPPING PLUS SEARCH
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 14



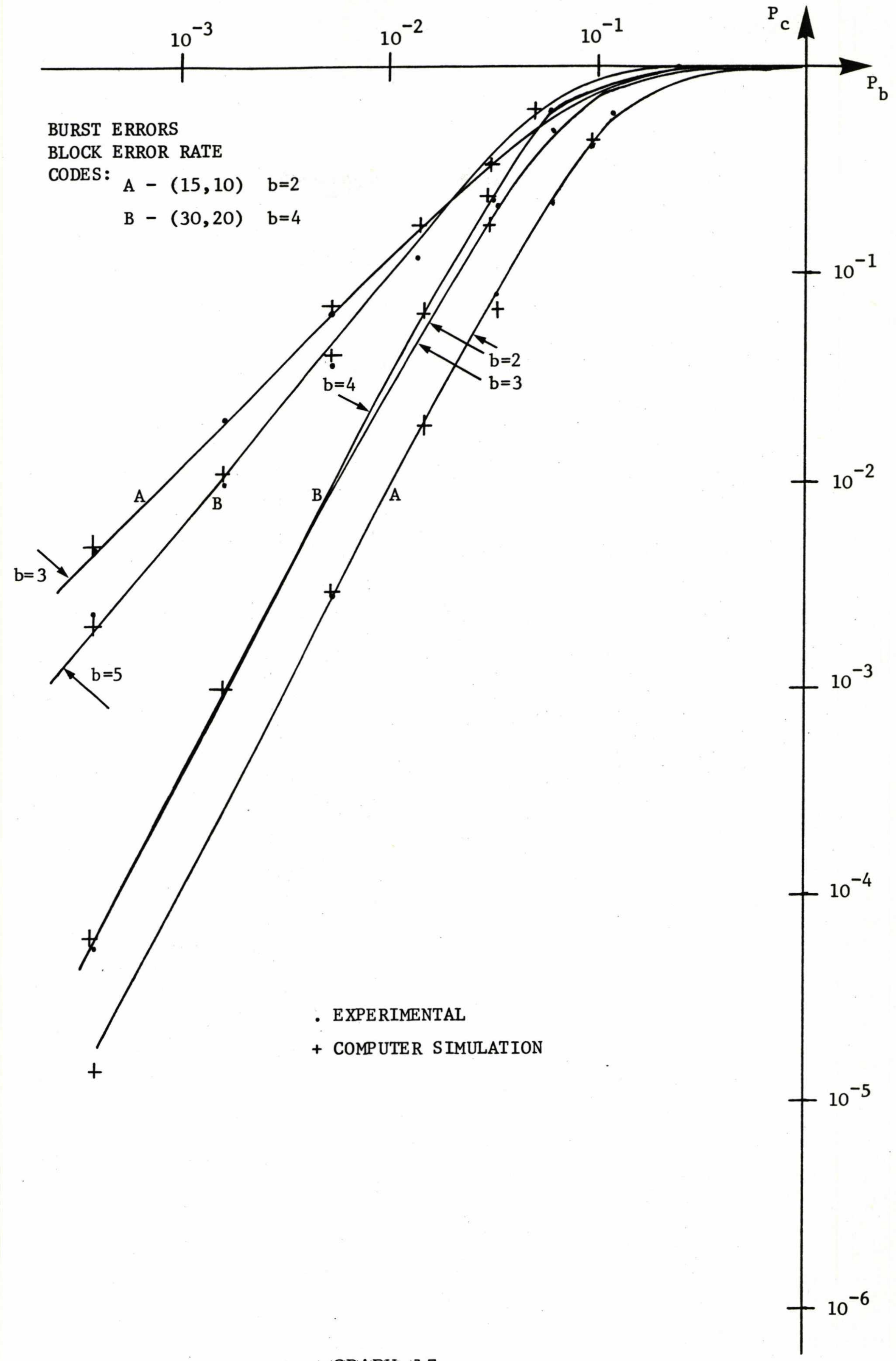
10^{-3}
10^{-2}
10^{-1}
P_c
P_b
BURST ERRORS
BLOCK ERROR RATE
CODES: A - (7,3) b=2
B - (14,6) b=4
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
A
B
b=3
b=5
B
A
b=2
b=3
b=4
b=2
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 15



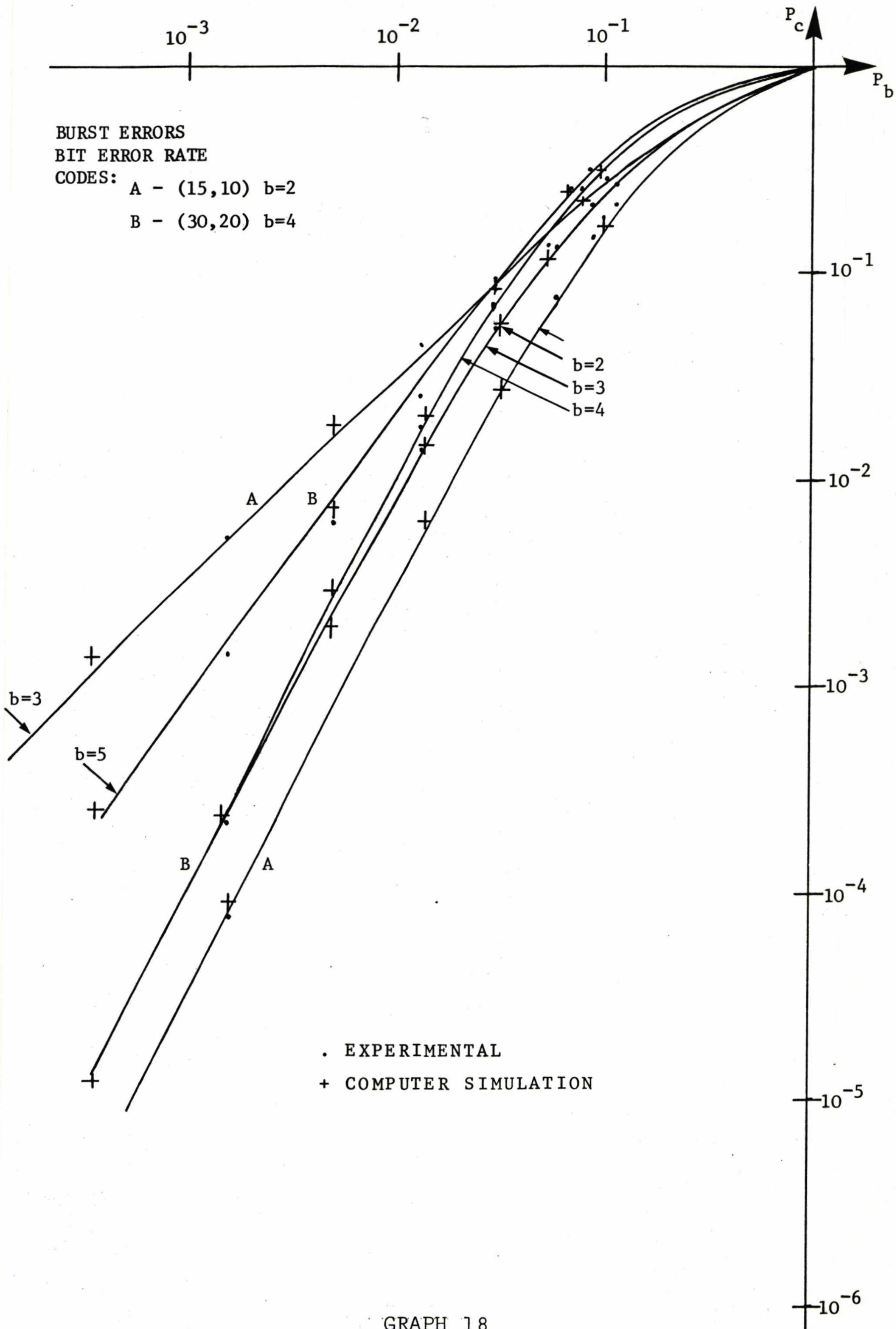
P_c
P_b
10^{-3}
10^{-2}
10^{-1}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
BURST ERRORS
BIT ERROR RATE
CODES: A - (7,3) b=2
B - (14,6) b=4
A
B
b=3
b=5
B
A
b=2
b=3
b=4
b=2
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 16



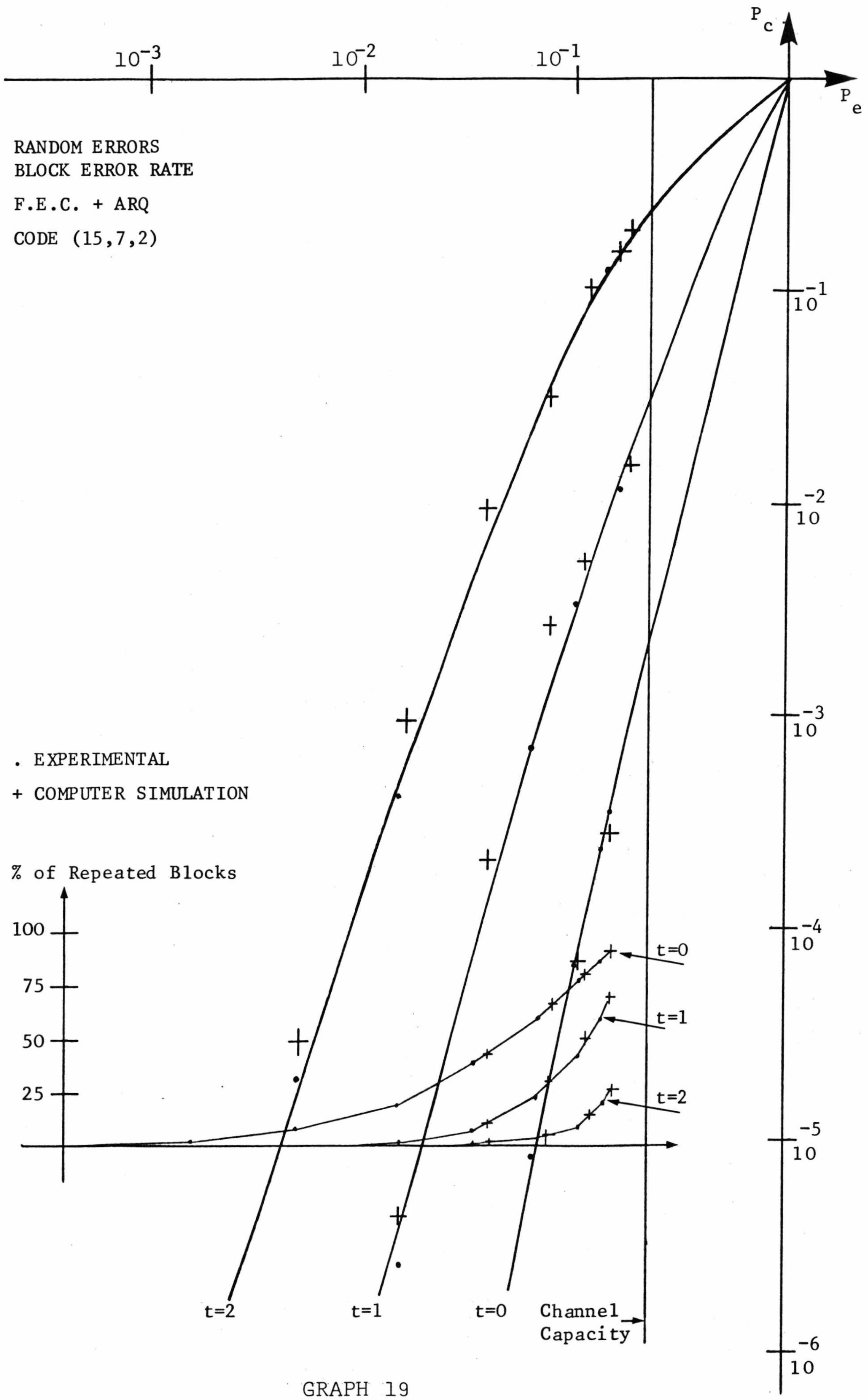
10^{-3}
10^{-2}
10^{-1}
P_c
P_b
BURST ERRORS
BLOCK ERROR RATE
CODES: A - (15,10) b=2
B - (30,20) b=4
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
b=2
b=3
b=4
A
B
B
A
b=3
b=5
. EXPERIMENTAL
+ COMPUTER SIMULATION

GRAPH 17



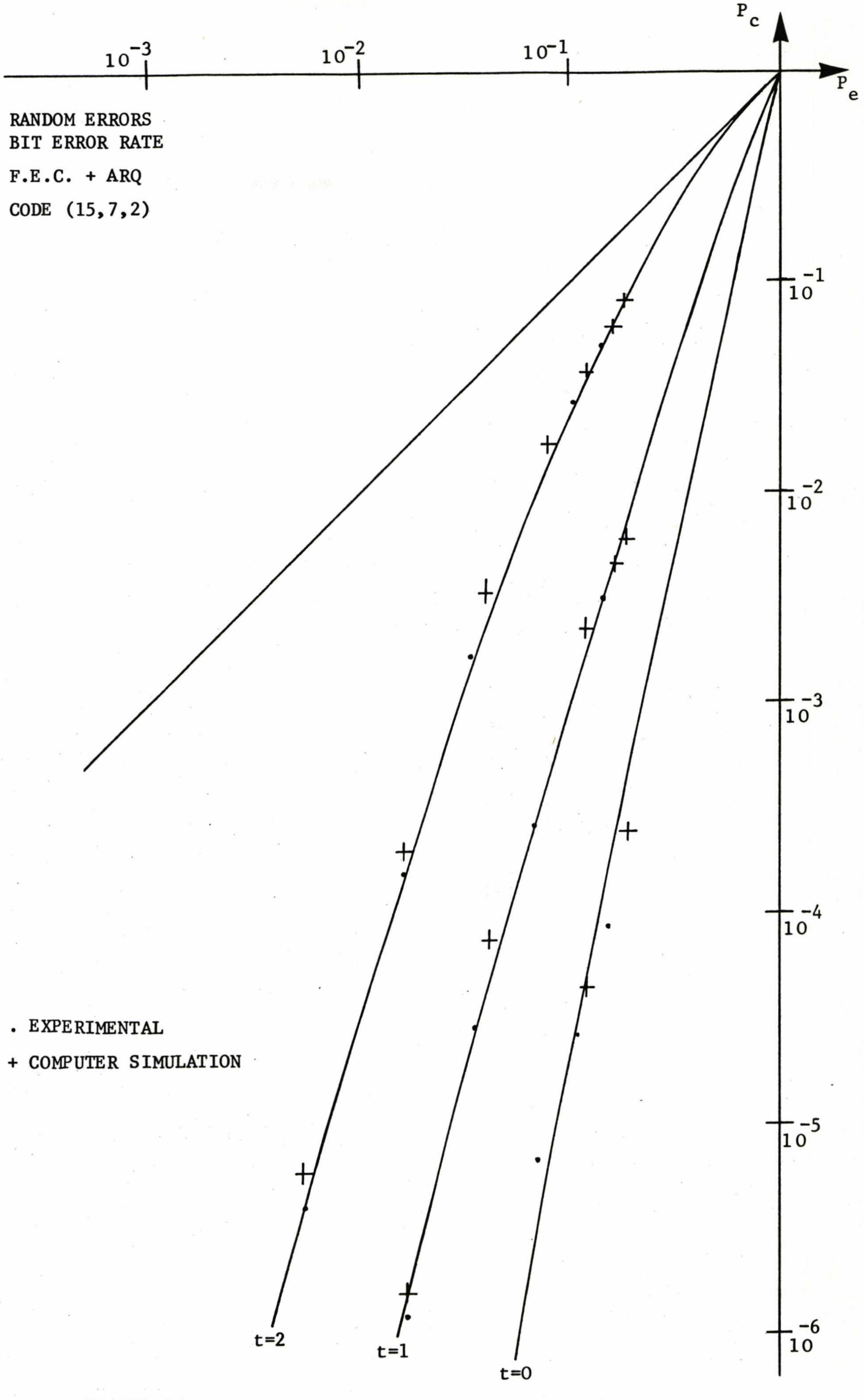
10^{-3}
10^{-2}
10^{-1}
P_c
P_b
BURST ERRORS
BIT ERROR RATE
CODES: A - (15,10) b=2
B - (30,20) b=4
10^{-1}
b=2
b=3
b=4
A
B
10^{-2}
10^{-3}
b=3
b=5
B
A
10^{-4}
. EXPERIMENTAL
+ COMPUTER SIMULATION
10^{-5}
10^{-6}

GRAPH 18



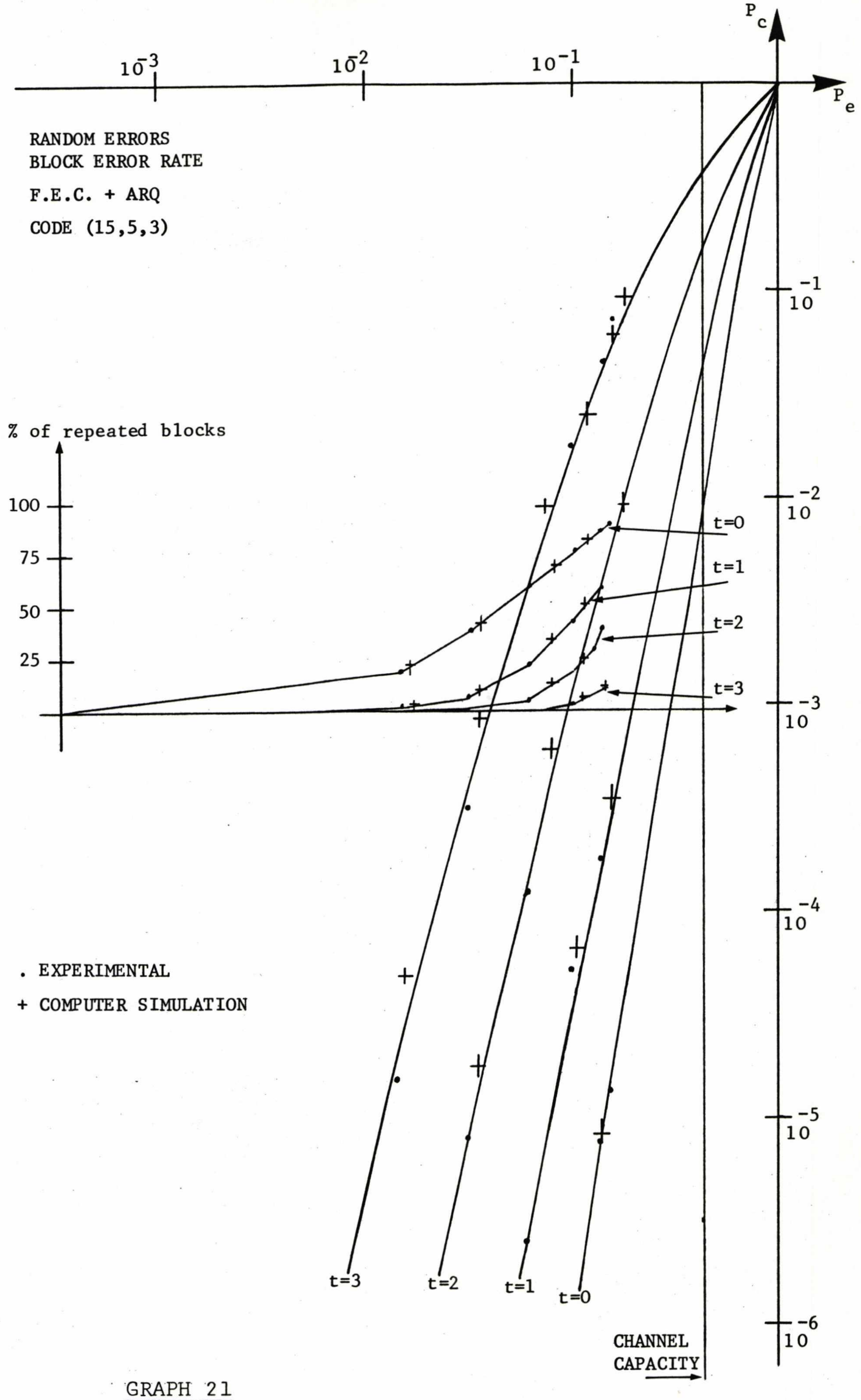
P_c
P_e
10^{-3}
10^{-2}
10^{-1}
RANDOM ERRORS
BLOCK ERROR RATE
F.E.C. + ARQ
CODE (15,7,2)
. EXPERIMENTAL
+ COMPUTER SIMULATION
% of Repeated Blocks
100
75
50
25
t=0
t=1
t=2
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
t=2
t=1
t=0
Channel Capacity

GRAPH 19



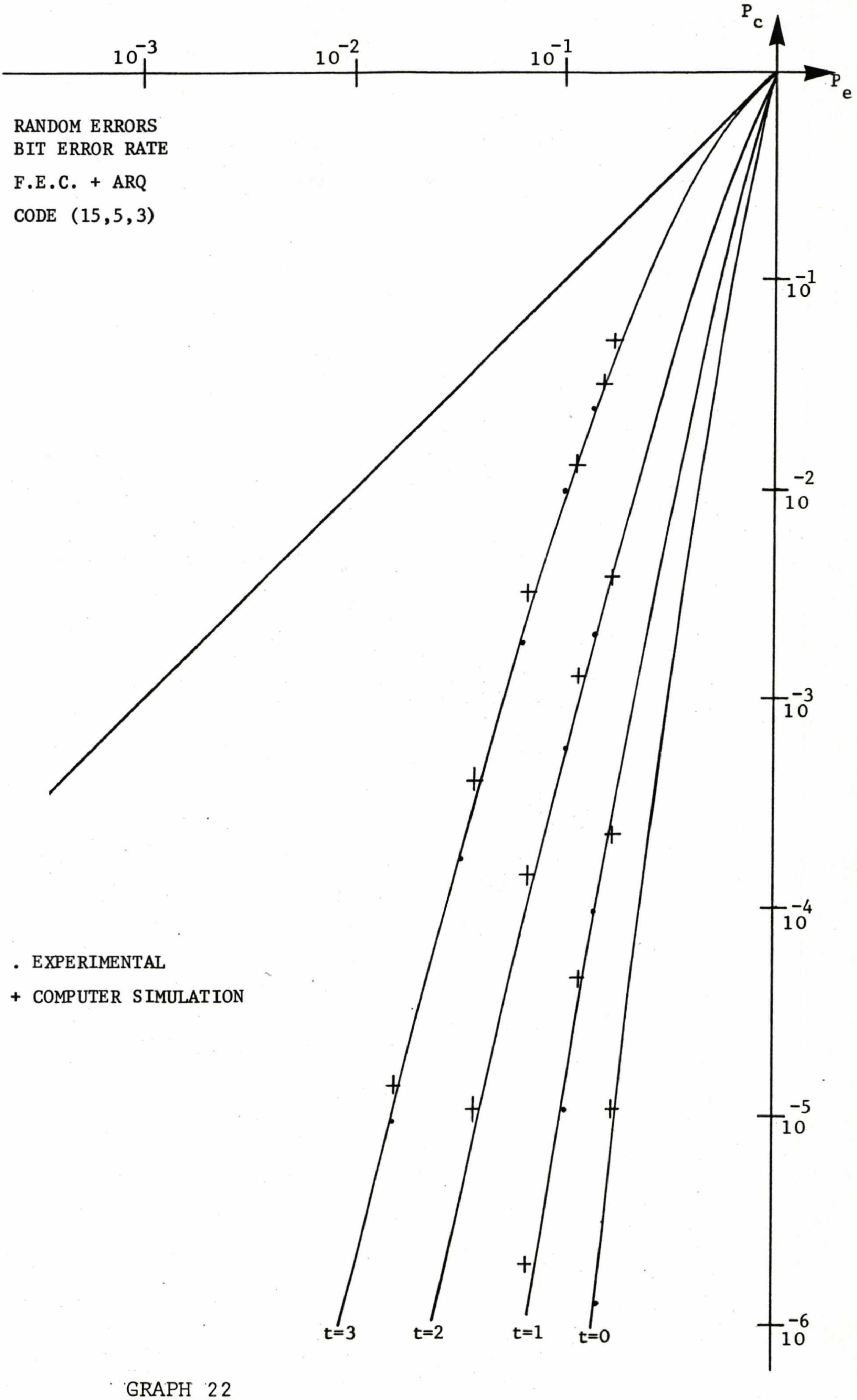
P_c
P_e
10^{-3}
10^{-2}
10^{-1}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
RANDOM ERRORS
BIT ERROR RATE
F.E.C. + ARQ
CODE (15,7,2)
. EXPERIMENTAL
+ COMPUTER SIMULATION
t=2
t=1
t=0

GRAPH 20



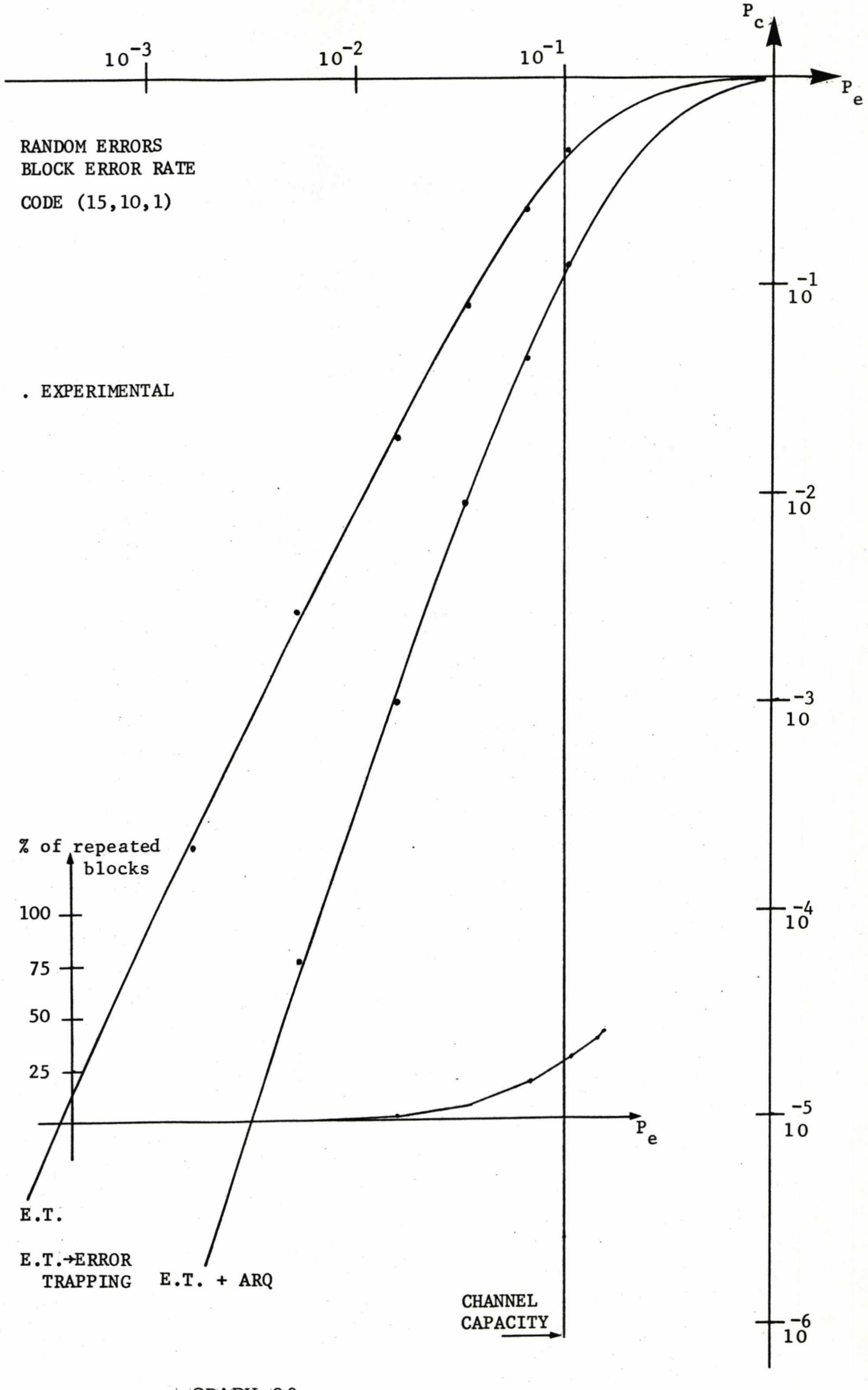
P_c
P_e
10^{-3}
10^{-2}
10^{-1}
RANDOM ERRORS
BLOCK ERROR RATE
F.E.C. + ARQ
CODE (15,5,3)
% of repeated blocks
100
75
50
25
t=0
t=1
t=2
t=3
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
. EXPERIMENTAL
+ COMPUTER SIMULATION
t=3
t=2
t=1
t=0
CHANNEL
CAPACITY

GRAPH 21



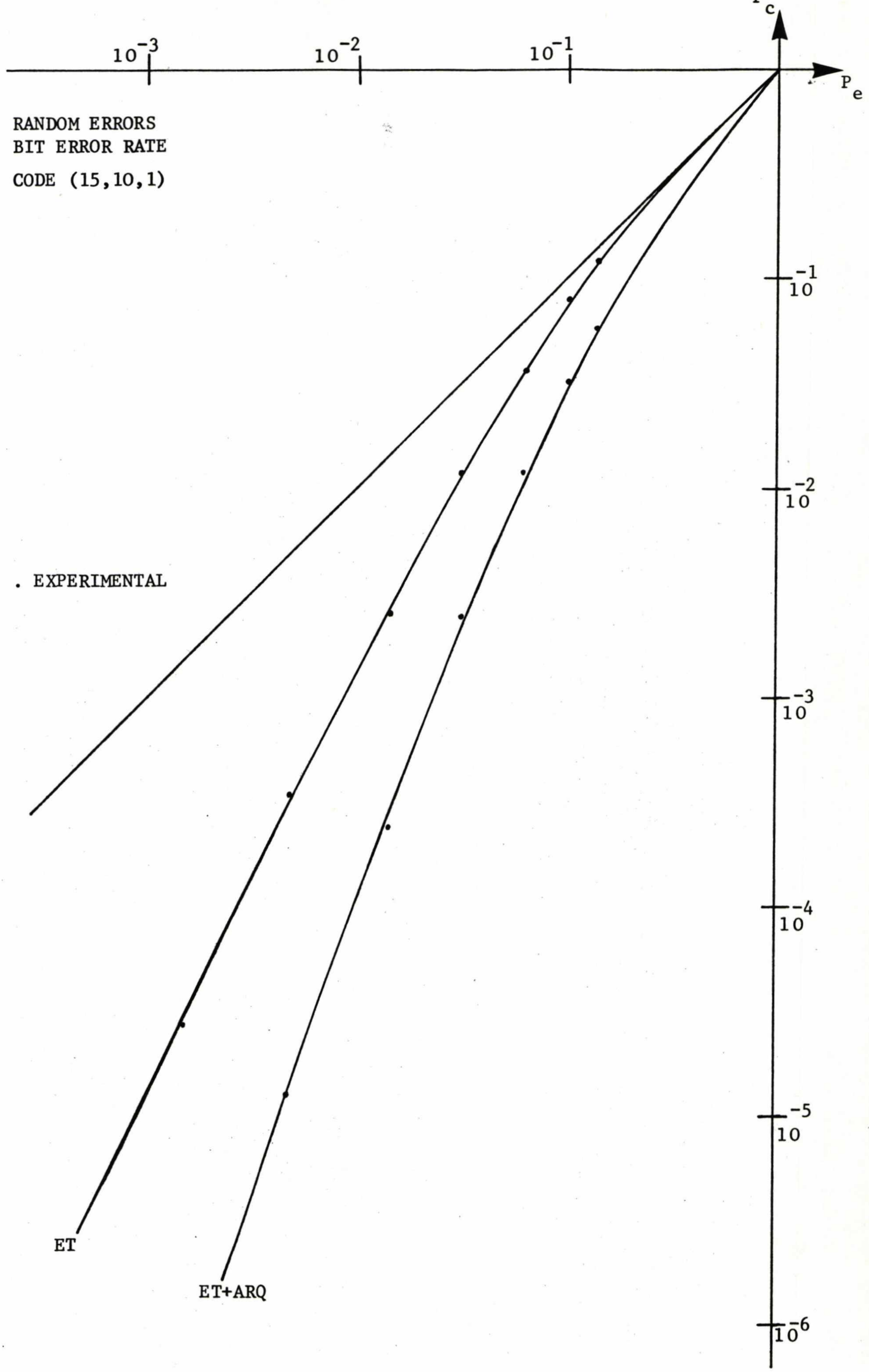
P_c
P_e
10^{-3}
10^{-2}
10^{-1}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
RANDOM ERRORS
BIT ERROR RATE
F.E.C. + ARQ
CODE (15,5,3)
. EXPERIMENTAL
+ COMPUTER SIMULATION
t=3
t=2
t=1
t=0

GRAPH 22

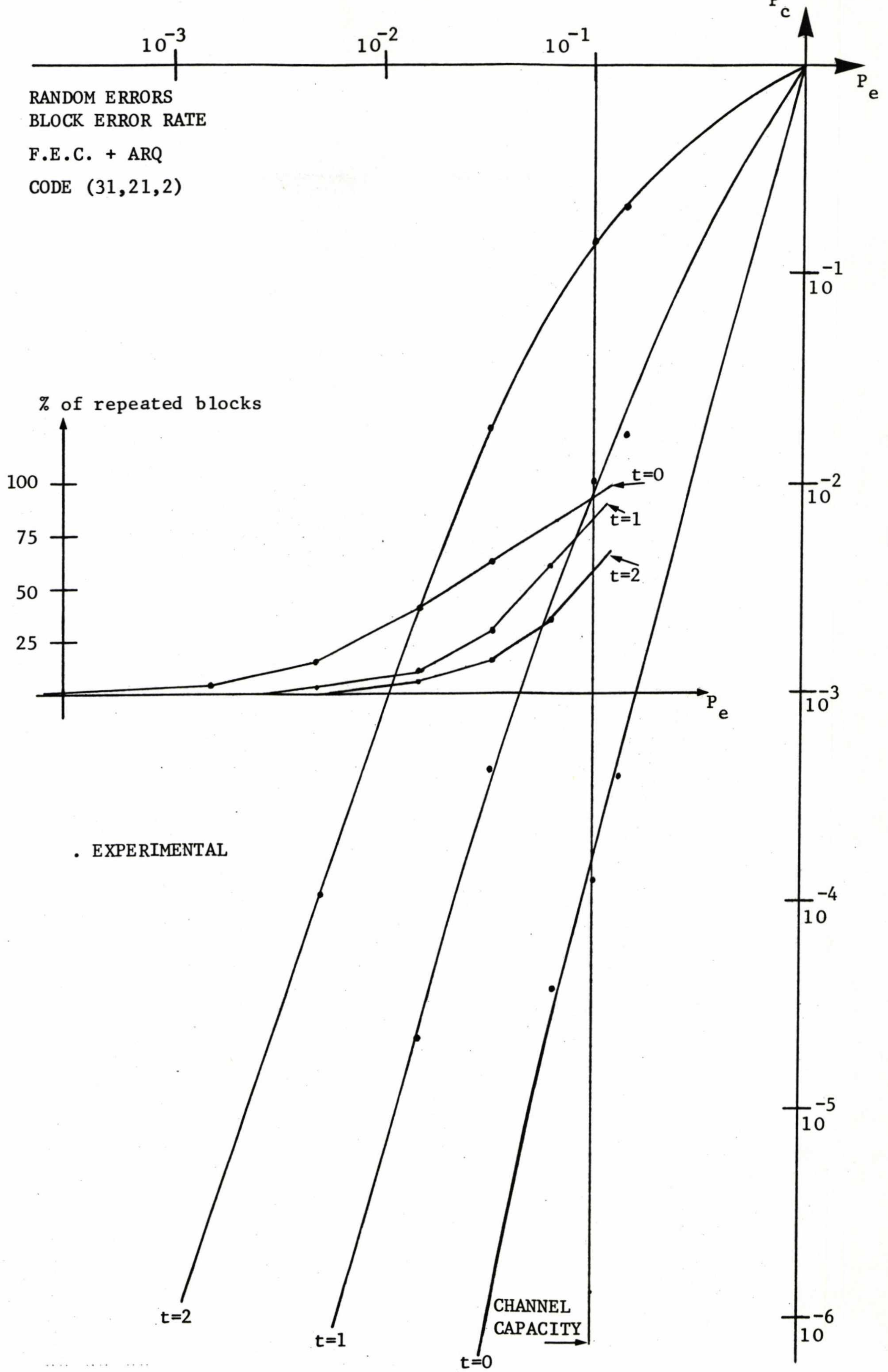


P_c
10^{-3}
10^{-2}
10^{-1}
P_e
RANDOM ERRORS
BLOCK ERROR RATE
CODE (15,10,1)
. EXPERIMENTAL
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
% of repeated blocks
100
75
50
25
P_e
E.T.
E.T.→ERROR TRAPPING
E.T. + ARQ
CHANNEL CAPACITY

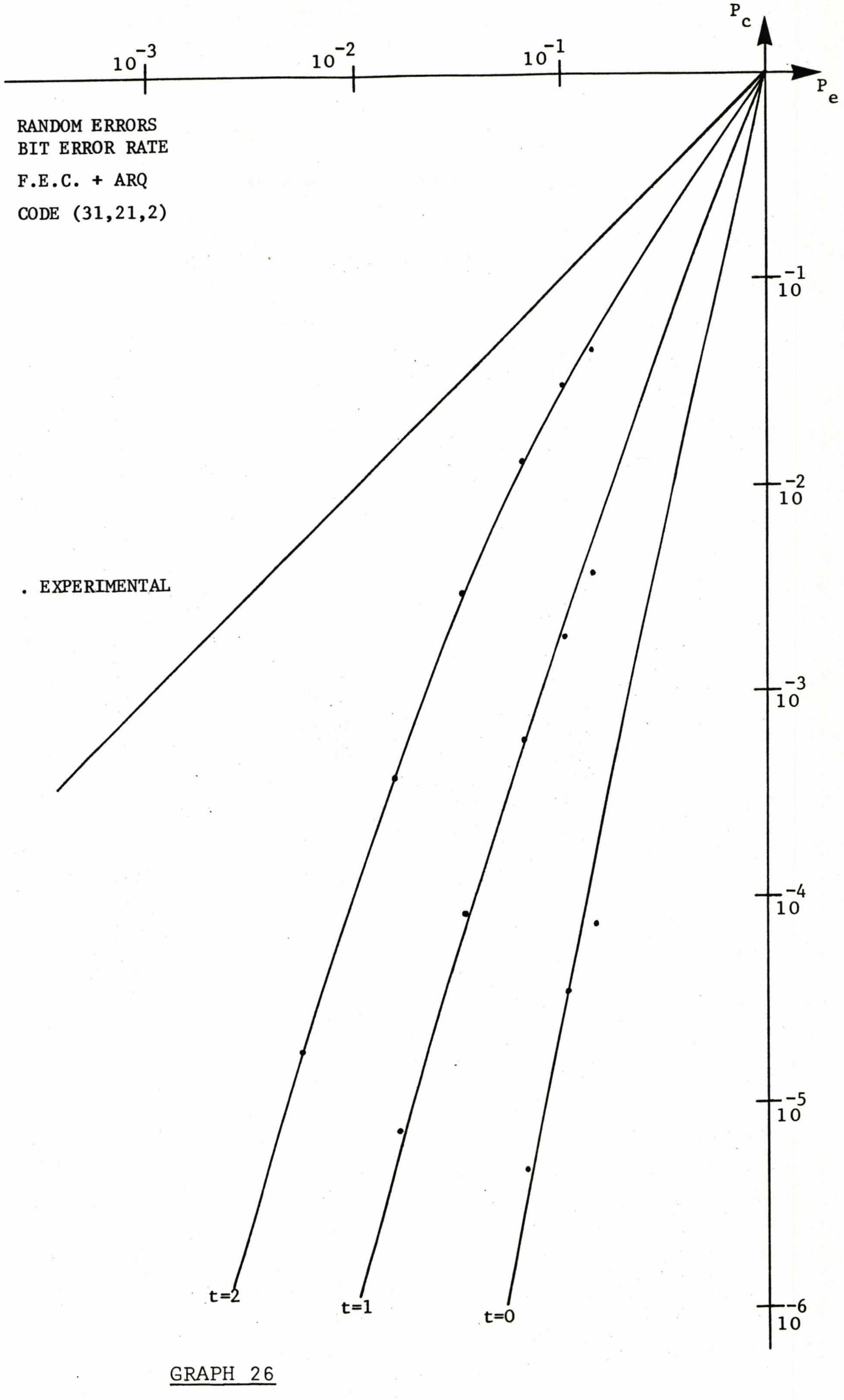
GRAPH 23


P_c
P_e
10^{-3}
10^{-2}
10^{-1}
RANDOM ERRORS
BIT ERROR RATE
CODE (15,10,1)
. EXPERIMENTAL
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
ET
ET+ARQ

GRAPH 24


P_c
10^{-3}
10^{-2}
10^{-1}
P_e
RANDOM ERRORS
BLOCK ERROR RATE
F.E.C. + ARQ
CODE (31,21,2)
10^{-1}
% of repeated blocks
100
75
50
25
t=0
t=1
t=2
10^{-2}
P_e
10^{-3}
. EXPERIMENTAL
10^{-4}
10^{-5}
t=2
t=1
CHANNEL CAPACITY
t=0
10^{-6}

GRAPH 25



P_c
P_e
10^{-3}
10^{-2}
10^{-1}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
RANDOM ERRORS
BIT ERROR RATE
F.E.C. + ARQ
CODE (31,21,2)
. EXPERIMENTAL
t=2
t=1
t=0

GRAPH 26

CHAPTER 6

Conclusion

The main objective of the research described in the previous chapters was the investigation of the practical performance of block error correcting codes as a means of reducing the error rate in communication systems. This study necessarily involved an analysis and a discussion of various coding techniques and their associated decoding problems. For practical reasons, the sub-class of linear block codes called cyclic codes was used for the investigation.

In Chapter 1 the general ideas related to the use of error correcting codes in communication systems were presented. Also, the Coding Theorem and the consequent coding problem were introduced.

Chapters 2 and 3 dealt with the theory of linear block codes, the study of cyclic codes, and some important coding techniques which were reviewed and compared. From this study became apparent the practical advantages to be gained by using linear cyclic codes rather than simply linear codes. Chapter 4 described the hardware used in the experimental investigation which comprised a general encoder/decoder for cyclic codes, a noise generator and the associated circuits for counting errors. The practical performance of many cyclic codes was assessed using the hardware constructed, and the results obtained were checked by computer simulation. The construction of similar hardware to study linear codes would have involved an extremely complicated decoder for the reasons given in Chapter 2. Chapter 5 presented and discussed the results of error measurements obtained with the experimental system and by a computer simulation. In all cases there was always close agreement between both sets of results.

From the various graphs shown in Chapter 5 it is easy to see how the slopes of the curves plotted are related to the error-correcting power of the corresponding codes. For a random t error-correcting code a slope of t+1 is obtained, in the linear region of the graph, when the decoder used can correct all errors up to t per codeword. In this case only patterns containing t+1 or more errors will cause the decoder to decode wrongly. For a sufficiently small P_e the effect of patterns with more than t+1 errors in the system error rate is negligible when compared with those containing t+1 errors. In situations where error-trapping is not sufficient to correct all patterns with t errors, the curves obtained will fall somewhere between lines of slope t and t+1, and will tend asymptotically to a slope of t. Also, from the graphs shown, it can be seen that for codes with the same rate, shorter codes apparently perform better than longer codes at high error rates. However, this may not be a true advantage because when it happens the codes have already deteriorated in performance considerably.

As mentioned before, most real channels are difficult to model and the use of simple hardware, like the one constructed, makes possible quick and easy measurement of code performance. As far as practical applications of coding are concerned, this procedure is very effective whenever it is feasible. Even for channels like the binary symmetric channel the exact calculation of bit error rate after decoding is normally very complicated and impractical (Turner, 1976).

In practical applications of coding, the use of a powerful code, able to cope with a worst case situation, normally implies a very low throughput for the system even during periods of relatively low noise. If, alternatively, a code is selected which is efficient in a quiet channel, it will often have a poor performance during noisy periods. In this context the so called fixed redundancy schemes are inefficient. However, in conjunction with the problem of efficient channel utilisation, the engineers are also faced with the problem of designing the decoding equipment. So, in many cases, the two requirements of simple decoding and reasonable efficiency may indicate the choice of a weak code. In order to make more efficient use of the channel it may be better in such cases to consider a variable redundancy system (i.e. to use a number of codes with a wide range of rates and error correcting power, selecting them adaptively according to varying channel conditions). The advantages of having a relatively simple general encoder/decoder are obvious in this case. With very little modification the hardware constructed provides a simple encoder/decoder for applications requiring variable redundancy.

If a situation arises where only one fairly long and powerful random error-correcting code must be used, then a BCH code can be a strong candidate because of its convenient decoding algorithm. Of course, this might not be the final answer to the problem because it is always a function of other parameters as well like cost, complexity, maximum tolerable decoding delay, and the error rate after decoding. Since all types of decoders, proposed so far, grow very fast in complexity with code length, it is of practical importance to use relatively short codes. Many powerful coding schemes result from a combination of short, easy to decode, codes. If a feedback channel is available, then by using A.R.Q. (perhaps in combination with forward error correction) a simple decoder results. A.R.Q. systems normally have a high throughput but during high noise periods become inefficient due to the many repeat requests; the addition of forward error correction in these cases may be an acceptable compromise because it helps to decrease the number of repeat requests at the cost of a reduction in throughput.

Recommendations for future research

The experimental system built proved very reliable, as was expected, due to the use of integrated circuit components. The only limiting factor, when larger values of n are required, is the decoder complexity. This clearly shows the need for more efficient decoding algorithms for cyclic codes, perhaps aiming to be optimum only for certain types of channel. Efficient decoding will possibly involve more sophisticated ways of manipulating the code redundancy, by exploring some properties of the channel e.g., the fact that most real channels are not memoryless. In the area of burst error correction there is always a need for more sophisticated interleaving techniques. Also, the construction of codes which meet the bound

$$R \leq \frac{g}{g+b}$$

(Gallager, 1968), rather than the more common

$$R \leq \frac{g-b}{g+b} \quad ,$$

deserves more attention. Burst correcting codes are normally designed to combat bursts on a burst length basis; it may be interesting to analyse the results of codes designed to combat bursts on a burst density basis.

An interesting extension of the research described here is the study of codes and coding schemes to operate under very noisy conditions, with error rates in the range of 0.5 to 0.1. Systems of this type may be of interest for use with channels subject to severe fading and multipath effects (e.g. HF links), or operating at very low receiver signal-to-noise ratios (e.g. spread-spectrum systems; Farrell & Munday, 1976).

Finally, the advent of microprocessors represent an important step forward for the implementation of experimental coding schemes. The use of software programmes, instead of hard wired logic, makes a system more versatile and easier to modify. Though, at the moment, the price-speed factor is not very favourable to microprocessors, as compared to systems built using TTL for example, it is possible to have cheaper microprocessors combined in parallel operation to improve speed.

APPENDIX I

Modern Algebra and Vector Spaces

The definitions presented here can be found in texts like Peterson (1972), Lin (1970), Berlekamp (1968) and Herstein (1964).

GROUP: A set of elements obeying the following postulates. If a, b and c are elements of the set then:

(1) $a+b$ is in the set, where (+) denotes an operation.

(2) $(a+b)+c = a+(b+c)$.

(3) There is a unique element 0 such that $0+a = a$.

(4) There is a unique element $-a$ such that $a+(-a) = 0$.

SUB GROUP: A subset of elements of a group which satisfies all the properties of a group and so forms itself a group with respect to the operation used for the group (+).

ABELIAN GROUP: A group with elements which satisfy a fifth postulate, namely:

(5) $a+b = b+a$.

RING: A set of elements which form an Abelian group under the operation (+) and satisfy the following postulates with respect to another operation (·).

(1) a.b is in the set.

(2) a.(b.c) = (a.b).c

(3) a.(b+c) = a.b + a.c

(4) (b+c).a = b.a + c.a

The ring is commutative if a.b = b.a

FIELD: A set of elements which form a ring, and the non-zero elements form an Abelian group with respect to the operation (·).

GALOIS FIELDS: GF(q)

It can be shown that for $q=p^k$, where p is a prime number, there exists a finite field containing q elements. This is called a Galois field of q elements. Those q elements will be the integers 0 to q-1 if and only if q is prime.

POLYNOMIAL RINGS:

The set of all polynomials in a single unknown X, with coefficients from GF(q), forms a commutative ring which is called a ring of polynomials over GF(q).

IDEALS:

The set of all polynomials with coefficients in GF(q) of the form g(x).p(x), where p(x) is any polynomial, is called the ideal generated by g(x).

RESIDUE CLASSES:

An ideal can be thought of as a subset of a ring of polynomials. The elements of this ring, which are not in the ideal, form what is called residue classes.

The elements of each residue class are characterised by the fact that subtraction between any pair of its members gives an element in the ideal.

<u>RESIDUE CLASS RING</u>:

The set of residue classes, as defined above, form a ring called a residue class ring with respect to an ideal.

<u>VECTOR SPACES</u>:

The sequence $[V] = [V_1, V_2, V_3, \ldots, V_n]$, where the components are elements from GF(2), i.e. either 0 or 1, is called an n-tuple over GF(2). There are 2^n different n-tuples, for a given n, due to the binary nature of the components V_i. <u>Addition</u> of two n-tuples $[U]$ and $[V]$ is defined as follows:

$$[U] = [U_1, U_2, U_3, \ldots, U_n]$$

$$[V] = [V_1, V_2, V_3, \ldots, V_n]$$

$$[U] + [V] = [(U_1+V_1), (U_2+V_2), \ldots, (U_n+V_n)]$$

where U_i+V_i represents the modulo-2 addition of U_i and V_i. <u>Scalar multiplication</u> of a binary n-tuple by an element a of GF(2) is defined as:

$$a[V_1, V_2, V_3, \ldots, V_n] = [aV_1, aV_2, aV_3, \ldots, aV_n]$$

The inner product of two n-tuples U and V is defined as:

$$[U] \cdot [V] = U_1V_1 + U_2V_2 + U_3V_3 + \ldots + U_nV_n$$

where addition and multiplication are taken modulo-2. A vector space V_n over GF(2) is defined as the set of all possible binary n-tuples. A subspace S_n of V_n is defined as a subset of V_n which contains the all zero n-tuple and such that the addition of any two n-tuples in S_n always results in a third n-tuple belonging to S_n. Given i n-tuples $[V_1],[V_2],[V_3],\ldots,[V_i]$, a linear combination of them is defined as:

$$[U] = C_1[V_1] + C_2[V_2] + C_3[V_3] + \ldots + C_i[V_i]$$

where the coefficients C_i are taken from GF(2). If it is possible to find scalars C_i from GF(2), not all zero, such that:

$$C_1[V_1] + C_2[V_2] + \ldots + C_i[V_i] = 0 ,$$

then the set $\{[V_1],[V_2],[V_3],\ldots,[V_i]\}$ is said to be linearly dependent. If the only set of scalars from GF(2) satisfying this condition is $C_1=C_2=\ldots=C_i=0$, then the set $\{[V_1],[V_2],\ldots,[V_i]\}$ is said to be linearly independent. In any vector space or subspace there is at least one set of linearly independent vectors (n-tuples) that generate, by linear combinations, all other vectors in the space or sub-space. This set is called a basis of the vector space or subspace. The dimension of a space is the number of vectors in its basis.

APPENDIX II

Computer Programmes

Random Error Correction

```
DIM PAT D(31,1),PAT D1(31,1),PAT X(31,1),PAT X1(31,1)
DIM PAT E(31,1),PAT BL(31,1),PAT C(31,1),PAT G(31,1)
DIM MAP SF(31,31),MAP S2(31,31),PAT S1(31,1),PAT II(31,1)
DIM PAT ZO(20,1),MAP FS(20,20),MAP FB(20,20)
DIM PAT C2(20,1),PAT D2(20,1),PAT X2(20,1)
1  PRINT "SR1 :UP FOR TT,DOWN FOR VT"
2  PRINT "GIVE N,K,T,M,THRESHOLD FOR PROB. OF ERROR"
3  INPUT N,K,T1,M,T4
4  IF SR2 GOTO 11
5  DRSPEC I3 DC T 10
7  INPUT #3 MAP FS,MAP FB
8  INPUT #3 MAP SF,MAP S2,PAT G,PAT D
9  INPUT #3 PAT S1,PAT II
10 DRSPEC
11 LET I5=0
12 DIM V(10),U(10)
13 PRINT "DO YOU NEED SEARCH? TYPE 1 FOR YES,0 FOR NO"
14 INPUT F
15 IF SR1 GOTO 20
17 DRSPEC TO TT
20 FOR I=1 TO 10
22 LET V(I)=0
24 NEXT I
26 LET PAT C2 BE NT PAT ZO
28 IF SR3 GOTO 800
30 FOR I3=1 TO 10
35 FOR J1=1 TO M
37 LET I2=0
40 LET PAT X1 BE PAT D
45 LET PAT E BE PAT BL
50 FOR J=1 TO N
55 LET PAT X BE PAT X1 BY MAP SF
60 LET PAT X1 BE PAT X                  Generation of random errors
62 LET PAT X2 BE PAT C2 BY MAP FS
64 LET PAT D2 BE PAT C2 BY MAP FB
66 LET PAT C2 BE PAT X2 EX PAT D2
68 LET S=SUM PAT C2
70 IF S<>T4 THEN GOTO 80
75 LET PAT E BE PAT E OR PAT X1
80 NEXT J
85 LET S=SUM PAT E                      Avoid running through the
90 IF S=0 THEN GOTO 505                 programme when there are
95 LET PAT D1 BE PAT E                  no errors
100 GOSUB 600
```

```
110 LET B=SUM PAT D1
115 IF B<=T1 THEN GOTO 230
120 LET I2=I2+1
125 IF I2=N+1 THEN GOTO 300
130 LET PAT X BE PAT D1 BY MAP SF
135 LET PAT D1 BE PAT X
140 LET PAT X BE PAT D1 AN PAT D          Error trapping
145 LET S=SUM PAT X
150 IF S=0 THEN GOTO 120
155 LET PAT D1 BE PAT D1 EX PAT G
160 GOTO 110
230 LET PAT C BE PAT D1 BY MAP S2
240 IF I2=0 THEN GOTO 490
250 FOR I=1 TO N-I2
255 LET PAT X BE PAT C BY MAP SF
260 LET PAT C BE PAT X
265 NEXT I
270 GOTO 490
300 REM SEARCH TO COMPLEMENT ERROR TRAPPING
302 IF F=0 THEN GOTO 480
304 LET PAT X1 BE PAT D1
305 LET PAT C BE PAT S1
310 LET PAT D1 BE PAT C
315 GOSUB 600
320 LET I2=0
325 IF I2=N+1 THEN GOTO 450
330 LET PAT X BE PAT X1 EX PAT D1
335 LET S=SUM PAT X
340 IF S=0 THEN GOTO 485
345 LET I2=I2+1
350 LET PAT X BE PAT D1 BY MAP SF
355 LET PAT D1 BE PAT X
360 LET PAT X BE PAT C BY MAP SF
365 LET PAT C BE PAT X
370 LET PAT X BE PAT D1 AN PAT D
375 LET S=SUM PAT X
380 IF S=0 THEN GOTO 325
385 LET PAT D1 BE PAT D1 EX PAT G
390 GOTO 325
450 LET I5=I5+1
455 IF I5=6 THEN GOTO 480
460 LET PAT C BE PAT C EX PAT II
465 LET PAT D1 BE PAT C
470 GOTO 315
480 LET PAT C BE PAT BL
485 LET I5=0
490 LET PAT C BE PAT C EX PAT E
495 LET S=SUM PAT C                     Error correction and bit and block
496 IF S=0 THEN GOTO 505                error count
500 LET V(I3)=V(I3)+S
502 LET U(I3)=U(I3)+1
505 NEXT J1
510 PRINT V(I3)P
515 NEXT I3
520 PRINT "     "
525 END
600 REM DIVISION OF N-TUPLE BY G(X)
605 LET I1=0
```

```
610 LET I1=I1+1
615 LET PAT X BE PAT D1 AN PAT D
620 LET S=SUM PAT X
625 IF S=0 THEN GOTO 635
630 LET PAT D1 BE PAT D1 EX PAT G
635 IF I1=K THEN GOTO 700
640 LET PAT X BE PAT D1 BY MAP SF
645 LET PAT D1 BE PAT X
660 GOTO 610
700 RETURN
800 REM INITIALISE M-SEQUENCE
810 INPUT K1
820 FOR I=1 TO K1
830 LET PAT X2 BE PAT C2 BY MAP FS
840 LET PAT D2 BE PAT C2 BY MAP FB
850 LET PAT C2 BE PAT D2 EX PAT X2
860 NEXT I
870 GOTO 30
```

<u>Burst Error Correction</u>

```
DIM PAT Z0(20,1),PAT C2(20,1),PAT X2(20,1)
DIM MAP FS(20,20),MAP FB(20,20),PAT D2(20,1)
5  DIM T(31),X(31),G(31),L(31),V(11),U(10),W(10)
20 FOR I=1 TO 11
25 INPUT A3
30 LET V(I)=A3
35 NEXT I
36 LET PAT C2 BE NT PAT Z0
37 INPUT N,K,M
40 DRSPEC I3 DC T 10
42 INPUT #3 MAP FS,MAP FB
44 DRSPEC
45 IF SR1 GOTO 47
46 DRSPEC TO TT
47 FOR B=2 TO 5
50 FOR T4=10 TO 16
55 PRINT "BLOCK","BIT","LOG.AVERAGES:BLOCK AND BIT"
80 FOR I3=1 TO 5
82 FOR J3=1 TO M
83 LET J1=0
84 LET I5=0
85 LET I2=0
86 FOR I=1 TO N
90 LET PAT X2 BE PAT C2 BY MAP FS           Burst error generation
95 LET PAT D2 BE PAT C2 BY MAP FB
```

```
100 LET PAT C2 BE PAT X2 EX PAT D2
105 LET S=SUM PAT C2
107 LET L(I)=0
108 LET G(I)=0
110 IF S<>T4 THEN GOTO 180
115 LET J1=J1+1
120 LET T(I+B-1)=1                          Burst error generation
125 LET T(I)=1
130 IF B=2 THEN GOTO 165
135 FOR J=I+1 TO I+B-2
140 IF RND(0)>.5 THEN GOTO 155
145 LET T(J)=1
150 GOTO 160
155 LET T(J)=0
160 NEXT J
165 FOR J=I TO I+B-1
170 LET L(J)=T(J)
175 NEXT J
177 LET I=I+B-1
180 NEXT I
185 IF J1=0 THEN GOTO 485
200 GOSUB 600
205 REM STORE SYNDROME
210 FOR I=K+1 TO N
215 LET X(I)=T(I)
220 NEXT I
240 REM SHIFT SYNDROME
245 LET I5=I5+1
250 LET C=T(1)
260 FOR J=2 TO N
270 LET T(J-1)=T(J)
280 NEXT J
290 LET T(N)=C
300 GOSUB 600
310 REM COMPARE SHIFTED WITH STORED SYND.
320 FOR I=K+1 TO N
330 IF T(I)>X(I) THEN GOTO 240
335 IF T(I)<X(I) THEN GOTO 352
340 NEXT I
351 GOTO 360
352 LET I2=I5
354 FOR J=K+1 TO N
356 LET X(J)=T(J)
358 NEXT J
360 IF I5=N+I2 THEN GOTO 372
370 GOTO 240
372 IF I2=N THEN GOTO 450
380 FOR I=1 TO N-I2
390 LET C=X(1)
400 FOR J=2 TO N
410 LET X(J-1)=X(J)
420 NEXT J
430 LET X(N)=C
440 NEXT I
450 FOR I=1 TO N
460 LET T(I)=X(I)+L(I)
465 NEXT I
470 GOSUB 700
```

```
471 LET A=0
472 FOR I=1 TO N
473 LET A1=T(I)
474 LET X(I)=0
475 LET A=A+A1
477 LET T(I)=0
480 NEXT I
482 IF A=0 THEN GOTO 485
483 LET U(I3)=U(I3)+1              Bit and block error count
484 LET W(I3)=W(I3)+A
485 NEXT J3
486 PRINT "  "
487 PRINT U(I3),W(I3)
488 PRINT LOG((U(I3)/M)+1E-20)/LOG(10),LOG((W(I3)/(N*M))+1E-20)/LOG(10)
489 LET U(I3)=0
490 LET W(I3)=0
491 NEXT I3
492 PRINT "THRESHOLD T4="PT4,"BURST LENGTH B="PB,"USED ABOVE"
495 NEXT T4
497 NEXT B
500 STOP
600 REM DIVISION OF N-TUPLE BY G(X)
601 LET I1=0
602 FOR I=1 TO 11
603 LET G(I)=V(I)
604 NEXT I
606 LET I1=I1+1
610 LET A=T(I1)+G(I1)
615 IF A=2*INT(A/*) THEN GOTO 625
620 GOTO 638
625 FOR I=1 TO N
630 LET T(I)=T(I)+G(I)
635 NEXT I
638 IF I1=K THEN GOTO 700
640 FOR I=N TO 2 STEP -1
645 LET G(I)=G(I-1)
650 NEXT I
655 LET G(1)=0
665 GOTO 606
700 FOR I=1 TO N
705 IF T(I)=2*INT(T(I)/2) THEN GOTO 720
710 LET T(I)=1
715 GOTO 735
720 LET T(I)=0
735 NEXT I
740 RETURN
```

Forward Error Correction with A.R.Q.

```
DIM PAT D(15,1),PAT D1(15,1),PAT X(15,1),PAT X1(15,1)
DIM PAT E(15,1),PAT BL(15,1),PAT C(15,1),PAT G(15,1)
DIM MAP SF(15,15),MAP S2(15,15)
DIM PAT ZO(20,1),MAP FS(20,20),MAP FB(20,20)
DIM PAT C2(20,1),PAT D2(20,1),PAT X2(20,1)
1  PRINT "SR1 :UP FOR TT,DOWN FOR VT"
2  PRINT "GIVE N,K,T,M"
3  INPUT N,K,T1,M
4  IF SR2 GOTO 11
5  DRSPEC I3 DC T 10
7  INPUT #3 MAP FS,MAP FB
8  INPUT #3 MAP SF,MAP S2,PAT G,PAT D
10 DRSPEC
11 DIM V(10),R(10),U(10)
15 IF SR1 GOTO 20
17 DRSPEC TO TT
20 LET PAT C2 BE NT PAT ZO
22 IF SR3 GOTO 800
24 FOR T4=10 TO 16
26 FOR I=1 TO 10
27 LET V(I)=0
28 LET R(I)=0
29 LET U(I)=0
30 NEXT I
31 PRINT "T4="PT4
32 PRINT "BLOCK"," BIT ","LOG. AVERAGES:BLOCK AND BIT,"P
33 PRINT "REPEATED BLOCKS"
34 FOR I3=1 TO 5
35 FOR J1=1 TO M
37 LET I2=0
40 LET PAT X1 BE PAT D
45 LET PAT E BE PAT BL
50 FOR J=1 TO N
55 LET PAT X BE PAT X1 BY MAP SF
60 LET PAT X1 BE PAT X
62 LET PAT X2 BE PAT C2 BY MAP FS
64 LET PAT D2 BE PAT C2 BY MAP FB          Random error generation
66 LET PAT C2 BE PAT X2 EX PAT D2
68 LET S=SUM PAT C2
70 IF S<>T4 THEN GOTO 80
75 LET PAT E BE PAT E OR PAT X1
80 NEXT J
85 LET S=SUM PAT E
90 IF S=0 THEN GOTO 505
95 LET PAT D1 BE PAT E
100 GOSUB 600
110 LET B=SUM PAT D1
115 IF B<=T1 THEN GOTO 230
120 LET I2=I2+1                             Error trapping
125 IF I2=N+1 THEN GOTO 300
130 LET PAT X BE PAT D1 BY MAP SF
135 LET PAT D1 BE PAT X
140 LET PAT X BE PAT D1 AN PAT D
145 LET S=SUM PAT X
```

```
150 IF S=0 THEN GOTO 120
155 LET PAT D1 BE PAT D1 EX PAT G
160 GOTO 110
230 LET PAT C BE PAT D1 BY MAP S2
240 IF I2=0 THEN GOTO 490                   Error trapping
250 FOR I=1 TO N-I2
255 LET PAT X BE PAT C BY MAP SF
260 LET PAT C BE PAT X
265 NEXT I
270 GOTO 490
300 REM ARQ BEING USED
305 LET R(I3)=R(I3)+1                  ←  Count number of repeated blocks
310 GOTO 505
490 LET PAT C BE PAT C EX PAT E
495 LET S=SUM PAT C
497 IF S=0 THEN GOTO 505
500 LET U(I3)=U(I3)+S
502 LET V(I3)=V(I3)+1                       Count bit and block errors
505 NEXT J1
510 PRINT V(I3),U(I3),LOG((V(I3)/M)+1E-20)/LOG(10)
512 PRINT LOG((U(I3)/(N*M))+1E-20)/LOG(10)
513 PRINT R(I3)/M
515 NEXT I3
520 PRINT "             "
522 NEXT T4
525 END
600 REM DIVISION OF N-TUPLE BY G(X)
605 LET I1=0
610 LET I1=I1+1
615 LET PAT X BE PAT D1 AN PAT D
620 LET S=SUM PAT X
625 IF S=0 THEN GOTO 635
630 LET PAT D1 BE PAT D1 EX PAT G
635 IF I1=K THEN GOTO 700
640 LET PAT X BE PAT D1 BY MAP SF
645 LET PAT D1 BE PAT X
660 GOTO 610
700 RETURN
800 REM INITIALISE M-SEQUENCE
810 INPUT K1
820 FOR I=1 TO K1
830 LET PAT X2 BE PAT C2 BY MAP FS
840 LET PAT D2 BE PAT C2 BY MAP FB
850 LET PAT C2 BE PAT D2 EX PAT X2
860 NEXT I
870 GOTO 24
```

APPENDIX III

A list of the integrated circuits used in the construction of the hardware described in Chapter 4 is given below. These integrated circuits are all from the Transistor-Transistor Logic (TTL) family, series 74.

SN 7400	Quad 2 input NAND
SN 7402	Quad 2 input NOR
SN 7404	Hex inverter
SN 7408	Quad 2 input AND
SN 7410	Triple 3 input NAND
SN 7413	Dual 4 input NAND Schmitt trigger
SN 7420	Dual 4 input NAND
SN 7427	Triple 3 input NOR
SN 7432	Quad 2 input OR
SN 7474	Dual D-type flip-flop
SN 7482	2 bit binary full adder
SN 7483A	4 bit binary full adder
SN 7485	4 bit word magnitude comparator
SN 7486	Quad 2 input exclusive-OR
SN 7496	5 bit shift register PIPO
SN 74107	Dual J-K flip-flop
SN 74121	Monostable multivibrator
SN 74123	Dual monostable multivibrator
SN 74150	16 bit data selector
SN 74157	Quad 2 to 1 line selector
SN 74163	4 bit synchronous binary counter
SN 74164	8 bit shift register SIPO
SN 74174	Hex D-type flip-flop
SN 74H183	Dual carry save full adder
SN 74221	Dual monostable multivibrator
SN 74265	Quad delay equaliser
SN 74LS266	Quad 2 input exclusive-NOR O/C

Bibliography

The following abbreviations have been used in the bibliography:

BSTJ	Bell System Technical Journal
IBM	International Business Machines
IEE	Institution of Electrical Engineers
IEEE	Institute of Electrical & Electronics Engineers
IRE	Institute of Radio Engineers
MIT	Massachusetts Institute of Technology
SIAM	Society of Industrial & Applied Mathematics

IEEE Transactions:

COM	Communication Technology
CS	Communication Systems
EC	Electronic Computers
IT	Information Theory

N.M. Abramson (1959) — A Class of Systematic Codes for Non-Independent Errors. IRE Trans., IT-5, No.4, p.150, Dec.

N.M. Abramson (1961) — Error-Correcting Codes from Linear Sequential Circuits. The 4th London Symposium on Information Theory, Editor C. Cherry, Butterworths, p.26

N.M. Abramson (1968) — Cascade Decoding of Cyclic Product Codes. IEEE Trans., COM-16, No.3, p.398, June.

B.G. Bajoga & W.J. Walbesser (1973) — Decoder Complexity for BCH Codes. Proc. IEE, Vol.120, No.4, p.429, April.

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

E.R. Berlekamp (1973) — Goppa Codes. IEEE Trans., IT-19, No.5, p.590, Sept.

E.R. Berlekamp & O. Moreno (1973) — Extended Double-Error-Correcting Binary Goppa Codes are Cyclic. IEEE Trans., IT-19, No.6, p.817, Nov.

R.C. Bose & D.K. Ray-Chaudhuri (1960)	On a Class of Error Correcting Binary Group Codes. Info. & Control, Vol.3, p.68, March.
K. Brayer (1971)	Error Correction Code Performance on HF, Troposcatter, and Satellite Channels. IEEE Trans., COM-19, No.5, p.781, Oct.
D.T. Brown & W.W. Peterson (1961)	Cyclic Codes for Error Detection. Proc. IRE, Vol.49, No.1, p.228, Jan.
H.O. Burton & D.D. Sullivan (1972)	Errors and Error Control. Proc. IEEE, Vol.60, No.11, p.1293, Nov.
H.O. Burton & E.J. Weldon, Jr. (1965)	Cyclic Product Codes. IEEE Trans., IT-11, No.3, p.433, July.
R.D. Carmichael (1937)	Introduction to the Theory of Groups of Finite Order. Ginn & Company, New York.
R.T. Chien (1964)	Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes. IEEE Trans., IT-10, p.357, Oct.

R.T. Chien (1969)	Burst-Correcting Codes with High Speed Decoding. IEEE Trans., IT-15, No.1, p.109, Jan.
R.T. Chien (1971)	Block-Coding Techniques for Reliable Data Transmission. IEEE Trans., COM-19, No.5, p.743, Oct.
A.C. Davies (1967)	Probability Distributions of Pseudo-random Waveforms Obtained from m-sequences. Electronics Letters, Vol.3, p.115.
H.C.A. van Duuren (1961)	Error Probability and Transmission Speed on Circuits Using Error Detection and Automatic Repetition of Signals. IRE Trans., CS-9, No.1, p.38, March.
P. Elias (1954)	Error-Free Coding. IRE Trans., IT-4, p.29, Sept.
P. Elias (1955)	Coding for Noisy Channels. IRE Nat. Convention Record, Part 4, p.37.
R.M. Fano (1961)	Lectures on Communication System Theory, Chapter 23. Editor Elie J. Baghdady. McGraw-Hill Book Company.

P.G. Farrell (1969)	Coding for Noisy Data Links. Ph.D Thesis, University of Cambridge.
P.G. Farrell & E. Munday (1976)	Economical Practical Realisation of Minimum Distance Soft-Decision Decoding for Data Transmission. Proc. of the International Zurich Seminar on Digital Communications.
P. Fire (1959)	A Class of Multiple-Error-Correcting Binary Codes for Non-Independent Errors. Silvania Electric Products Inc., Report No.RSL-E-2, March.
G.D. Forney (1966a)	Generalised Minimum Distance Decoding. IEEE Trans., IT-12, No.2, p.125, April.
G.D. Forney (1966b)	Concatenated Codes. The MIT Press, Cambridge, Massachusetts.
G.D. Forney (1971)	Burst-Correcting Codes for the Classic Bursty Channel. IEEE Trans., COM-19, No.5, p.772, Oct.
R.G. Gallager (1963)	Low-Density Parity-Check Codes. The MIT Press, Cambridge, Massachusetts.
R.G. Gallager (1968)	Information Theory and Reliable Communication. New York, John Wiley & Sons.

E.N. Gilbert (1952)	A Comparison of Signalling Alphabets. BSTJ, Vol.31, p.504, May.
R. Glover (1974)	Private Communication.
J.M. Goethals (1967)	Factorisation of Cyclic Codes. IEEE Trans., IT-13, p.242, April.
J.M. Goethals & P. Delsarte (1968)	On a Class of Majority-Logic Decodable Cyclic Codes. IEEE Trans., IT-14, No.2, p.182, March.
M.J.E. Golay (1949)	Notes on Digital Coding. Proc. IRE, Vol.37, No.6, p.657, Jan.
M.J.E. Golay (1959)	Note on Binary Decoding. Proc. IRE, Vol.47, p.996, May.
S.W. Golomb (1967)	Shift-Register Sequences. Holden-Day Inc., San Franciso, California.
R.M.F. Goodman (1975)	Variable Redundancy Coding for Adaptive Error Control. Ph.D Thesis, University of Kent at Canterbury, June.
D. Gorenstein & N. Zierler (1961)	A Class of Cyclic Linear Error-Correcting Codes in p^m Symbols. Journal SIAM, Vol.9, p.207.

E. Gorog (1963)	Some Classes of Cyclic Codes Used for Burst-Error Correction. IBM Journal, Vol.7, No.2, p.102, April.
J.H. Green, Jr. & R.L. San Saucie (1958)	An Error-Correcting Encoder and Decoder of High Efficiency. Proc. IRE, Vol.46, No.7, p.1741, Oct.
D.H. Green & I.S. Taylor (1974)	Irreducible Polynomials over Composite Galois Fields and Their Applications in Coding Techniques. Proc. IEE, Vol.121, No.9, p.935, Sept.
R.W. Hamming (1950)	Error Detecting and Error Correcting Codes. BSTJ, Vol.29, p.147, April.
C.R.P. Hartmann & K.K. Tzeng (1973)	On Some Classes of Cyclic Codes of Composite Length. IEEE Trans., IT-19, No.6, p.820, Nov.
C.R.P. Hartmann & K.K. Tzeng (1974)	Decoding Beyond the BCH Bound Using Multiple Sets of Syndrome Sequences. IEEE Trans., IT-20, No.2, p.292, March.
A.A. Hashim & A.G. Constantinides (1974)	Class of Linear Binary Codes. Proc. IEE, Vol.121, No.7, p.555, July.
H.J. Helgert & R.D. Stinaff (1973a)	Minimum Distance Bounds for Binary Linear Codes. IEEE Trans., IT-19, No.3, p.344, May.

H.J. Helgert & R.D. Stinaff (1973b)	Shortened BCH Codes. IEEE Trans., IT-19, No.6, p.818, Nov.
I.N. Herstein (1964)	Topics in Algebra. Ginn and Company. Xerox College Publishing. Waltham, Mass.-Toronto.
A. Hocquenghem (1959)	Codes Correcteurs d'Erreurs. Chiffres, Vol.2, p.147.
H.T. Hsu, T. Kasami & R.T. Chien (1968)	Error-Correcting Codes for a Compound Channel. IEEE Trans., IT-14, No.1, p.135, Jan.
D.A. Huffman (1960)	On Decoding Linear Error Correcting Codes - Part I. IRE Trans., IT-6, p.450, Sept.
W.J. Hurd (1974)	Efficient Generation of Statistically Good Pseudonoise by Linearly Interconnected Shift Registers. IEEE Trans., C-23, No.2, p.146, Feb.
I.M. Jacobs (1974)	Practical Applications of Coding. IEEE Trans., IT-20, No.3, p.305, May.

J.R. Juroshek, R.T. Matheson & M. Nesenbergs (1971) — Interleaved Block Coding Tests over VHF and HF Channels. IEEE Trans., COM-19, No.5, p.790, Oct.

T. Kasami (1964) — A Decoding Procedure for Multiple Error-Correcting Cyclic Codes. IEEE Trans., IT-10, p.134, April.

T. Kasami, S. Lin & W.W. Peterson (1968) — Polynomial Codes. IEEE Trans., IT-14, No.6, p.807, Nov.

W.H. Kautz & K.N. Levitt (1969) — A Survey of Progress in Coding Theory in the Soviet Union. IEEE Trans., IT-15, No.1, p.197, Jan.

S.Y. Kuba & R.B. Lowry (1971) — Estimating the Performance of Error-Correcting Codes on the HF Channel. IEEE Trans., COM-19, No.5, p.802, Oct.

C.Y. Lee (1958) — Some Properties of Non-Binary Error Correcting Codes. IEEE Trans., IT-4, No.2, p.77, June.

D.J.H. Lewis & M. Fukada (1973) — A Note on Burst-Error Correction Using the Check Polynomial. IEEE Trans., IT-19, No.2, p.246, March.

S. Lin (1967)	Some Codes Which are Invariant Under a Transitive Permutation Group and Their Connection with Balanced Incomplete Block Designs. Proc. of the Conference on Combinational Mathematics and its Applications, Chapel Hill, N.C.: University of North Carolina Press.
S. Lin (1968)	On a Class of Cyclic Codes. Chapter 7 in Error-Correcting Codes, Editor H.B. Mann. John Wiley & Sons, Inc.
S. Lin (1970)	Introduction to Error-Correcting Codes. Englewood Cliffs, N.J.: Prentice-Hall, Inc.
J.H. van Lint (1970)	Nonexistence Theorems for Perfect Error-Correcting Codes. Computers in Algebra and Number Theory, SIAM AMS Proc., Vol.45, G. Birkhoff and M. Hall, Jr., Editors, p.89.
R.W. Lucky, J. Salz & E.J. Weldon, Jr. (1968)	Principles of Data Communication. New York : McGraw-Hill Book Company.
F.J. MacWilliams (1964)	Permutation Decoding of Systematic Codes. BSTJ, Vol.43, p.484, Jan.

H.B. Mann (Editor) (1968)	Error Correcting Codes. Proceedings of a Symposium organised by the Mathematics Research Center, U.S. Army, University of Wisconsin, New York: John Wiley & Sons, Inc.
J.L. Massey (1963)	Threshold Decoding. The MIT Press, Cambridge, Massachusetts.
J.L. Massey (1969)	Shift Register Synthesis and BCH Decoding. IEEE Trans., IT-15, No.1, p.122, Jan.
H.F. Mattson & G. Solomon (1961)	A New Treatment of Bose-Chaudhuri Codes. Journal SIAM, Vol.9, No.4, p.654.
E.J. McCluskey, Jr. (1965)	Introduction to the Theory of Switching Circuits. New York: McGraw-Hill Book Company.
C.M. Melas (1960)	A Cyclic Code for Double Error-Correction. IBM Journal, Vol.4, No.3, p.364, July.
D.E. Muller (1954)	Application of Boolean Algebra to Switching Circuit Design and Error Detection. IRE Trans., EC-3, p.6, Sept.

A.W. Nordstrom & J.P. Robinson (1967)	An Optimum Nonlinear Code. Info. and Control, Vol.11, p.613.
J.K. Omura (1969)	A Probabilistic Decoding Algorithm for Binary Group Codes. Presented at the IEEE International Symposium on Information Theory.
W.K. Pehlert, Jr. (1971)	Design and Evaluation of a Generalised Burst-Trapping Error Control System. IEEE Trans., COM-19, No.5, p.863, Oct.
W.W. Peterson (1960)	Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes. IRE Trans., IT-6, p.459, Sept.
W.W. Peterson (1961)	Error-Correcting Codes. The MIT Press, Cambridge, Massachusetts.
W.W. Peterson & E.J. Weldon, Jr. (1972)	Error-Correcting Codes. 2nd Edition. The MIT Press, Cambridge, Massachusetts.
M. Plotkin (1960)	Binary Codes with Specified Minimum Distance. IRE Trans., IT-6, No.3, p.445, Sept.

E. Prange (1957)	Cyclic Error-Correcting Codes in Two Symbols. AFCRC-TN-57-103, Air Force Cambridge Research Center, Cambridge, Massachusetts, Sept.
E. Prange (1958)	Some Cyclic Error-Correcting Codes with Simple Decoding Algorithms. AFCRC-TN-58-156, Air Force Cambridge Research Center, Bedford, Mass., April.
E. Prange (1959)	The Use of Coset Equivalence in the Analysis and Decoding of Group Codes. AFCRC-TR-59-164, Air Force Cambridge Research Center, Cambridge, Mass., June.
F.P. Preparata (1968)	A Class of Optimum Nonlinear Double Error Correcting Codes. Info. and Control, Vol.13, p.378.
I.S. Reed (1954)	A Class of Multiple-Error-Correcting Codes and the Decoding Scheme. IRE Trans., IT-4, p.38, Sept.
I.S. Reed & G. Solomon (1960)	Polynomial Codes over Certain Finite Fields. Journal SIAM, Vol.8, p.300.

S.H. Reiger (1960)	Codes for the Correction of "Clustered" Errors. IRE Trans., IT-6, No.1, p.16, March.
G.I. Riley (1975)	Error Control for Data Multiplex Systems. Ph.D Thesis, University of Kent, Canterbury.
V.C. Rocha (1975)	The Decoding of Cyclic Codes Using P.R.O.M.s. Internal Report, Electronics Labs., University of Kent at Canterbury, Sept.
L.D. Rudolph (1967)	A Class of Majority-Logic Decodable Codes. IEEE Trans., IT-13, No.2, p.305, April.
L.D. Rudolph & C.R.P. Hartmann (1973)	Decoding by Sequential Code Reduction. IEEE Trans., IT-19, No.4, p.549, July.
L.D. Rudolph & M.E. Mitchell (1964)	Implementation of Decoders for Cyclic Codes. IEEE Trans., IT-10, No.3, p.259, July.
R.A. Scholtz (1966)	Codes with Synchronisation Capability. IEEE Trans., IT-12, No.2, p.135, April.

R.A. Scholtz (1969)	Maximal and Variable Word-Length Comma-Free Codes. IEEE Trans., IT-15, No.2, p.300, March.
C.E. Shannon (1948)	A Mathematical Theory of Communication. BSTJ, Vol.27, p.379, July & p.623, Oct.
D. Slepian (1956a)	A Class of Binary Signalling Alphabets. BSTJ, Vol.35, p.203.
D. Slepian (1956b)	A Note on Two Binary Signalling Alphabets. IRE Trans., IT-2, p.84.
D. Slepian (1960)	Some Further Theory of Group Codes. BSTJ, Vol.39, p.1219.
D. Slepian (1973)	Information Theory in The Fifties. IEEE Trans., IT-19, No.2, p.145, March.
J.J. Stifler (1971)	Theory of Synchronous Communication. Englewood Cliffs, N.J.: Prentice-Hall, Inc.
S.E. Tavares, P.E. Allard S.G.S. Shiva (1970)	Decomposition of Cyclic Codes into Cyclic Classes and Applications. International Symposium on Info. Theory, Noordwijk, The Netherlands, June.

A. Tietäväinen (1973)	On the Non-existence of Perfect Codes Over Finite Fields. Journal SIAM, Vol.24, p.88, Jan.
R.L. Townsend & E.J. Weldon, Jr. (1967)	Self-orthogonal Quasi-cyclic Codes. IEEE Trans., IT-13, No.2, p.183, April.
L.F. Turner & O.O. Olanyan (1976)	On the Error-Correcting Capability of Optimum Linear Block Codes. Proc. IEE, Vol.123, No.1, p.26, Jan.
R.R. Varshamov (1957)	Estimate of the Number of Signals in Error-Correcting Codes. Translated from Dokl.Akad.Nauk.SSSR, Vol.117, p.739.
A.J. Viterbi (1973)	Information Theory in the Sixties. IEEE Trans., IT-19, No.3, p.257, May.
E.J. Weldon, Jr. (1966)	Difference-Set Cyclic Codes. BSTJ, Vol.45, p.1045.
E.J. Weldon, Jr. (1967)	Euclidean Geometry Cyclic Codes. Proc. of a Symposium of Combinational Mathematics at the University of North Carolina, Chapel Hill, N.C.

J.K. Wolf (1973) A Survey of Coding Theory. IEEE Trans., IT-19, No.4, p.381, July.

The Engineering Staff of Texas Instruments (1972) The TTL Data Book for Design Engineers. Texas Instruments Components Group.

Index