ERROR CONTROL CODING WITH APPLICATIONS FOR INFOPAD

by

Yuming Kathy Lu

Memorandum No. UCB/ERL M94/80

10 October 1994

ERROR CONTROL CODING WITH APPLICATIONS FOR INFOPAD

by

Yuming Kathy Lu

Memorandum No. UCB/ERL M94/80

10 October 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ERROR CONTROL CODING WITH APPLICATIONS FOR INFOPAD

by

Yuming Kathy Lu

Memorandum No. UCB/ERL M94/80

10 October 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Table of Contents:

1 Introduction

1.1 The need of error correction for indoor wireless communications

With the growing demand for wireless computing, and its associated noisy transmission environment, the need for effective error correction coding is increasing. The goal of this project is to design an effective and robust coding scheme that provides a means for overcoming a fading and noisy environment; while meeting the data throughput requirements ranging from 5 kbit/sec for raw pen data to 1 Mbit/sec for compressed video data. The result of this work will be applied to the UC Berkeley InfoPad project, which is the motivation of this project. As different data types including video, speech, text-graphic have different bit error rate requirements, it is reasonable to apply a separate error correction algorithm to different applications. For instance, acceptable pen data requires a bit error rate (BER) of 10^{-5}, which implies error correction with high amount of redundancy; whereas video data with a BER of 10^{-3} has negligible detrimental visual effect.

1.2 Indoor Wireless Communication Environment

The common types of errors that are experienced in indoor wireless communications are:

a) Errors caused by multiuser interference together with multipath fading channel. The InfoPad base-station to pad radio link uses a spread spectrum CDMA communication scheme. In this scheme, the user data is encoded in a Walsh code, which is a perfectly orthogonal code. The high data rate at 1 MHz with 64x spread factor implies a 64 MHz actual transmission rate over the wireless link, which can result in severe intersymbol interference (ISI), and thus, causes bit errors. The bit error rate for a typical indoor environment varies from 0.1 to 10^{-6} [sheng]. In a situation when a large number of users are present, for instance, 30 users operating InfoPad within the same room, the interference can be closely modeled as a Gaussian distribution or a binomial distribution by the central limit theorem (CLT). On the other hand, it is important to notice that the transmitted data is correlated to its reflection off barriers and the transmitted data from other users due to the nature of the Walsh code. Therefore, the resulting interference is hardly white.

b) Another type of errors which are commonly seen are burst errors caused by obstacles between the transmitter and receiver which block the main transmitting path. The obstacles can be a piece of lab equipment, furniture, or even people. The duration of blockage of the main transmitting path can vary from hours to a fraction of a second, and errors in this case appear in bursts.

1.3 Report Organization

This report is composed of six chapters. Chapter 2 describes the theoretical background for error correction for both convolutional and block codes. The simulation environment in Ptolemy and results for various error correction algorithms are presented in chapter 3. In chapter 4, bit error rate analysis is performed for various error correction and error detection/retransmission schemes. Moreover, the performance and trade-offs of two approaches to error control are discussed. Chapter 5 presents a custom low power design of BCH(15, 5, 7) triple error correction code which will be used for the InfoPad. Finally, chapter 6 presents a system level view on how error correction and error detection can be combined to provide a reliable transmission over wireless channel while achieving bandwidth efficiency.

2

Background

2.1 Convolutional Code

The convolutional encoder functions as a Markov-type finite state machine. A simple encoder is shown below:

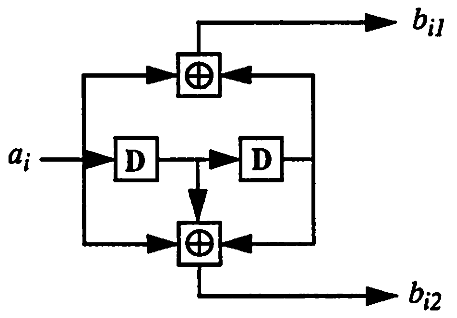
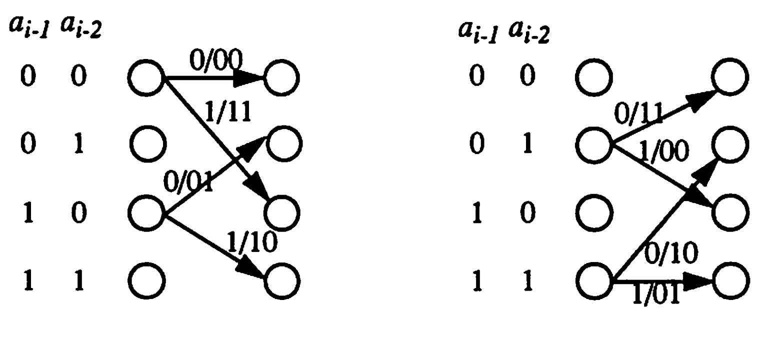


Fig 1. Rate-1/2, Constraint Length 2 Convolutional Encoder

Some salient features of the above encoder include: (1) for each input bit, two output bits are created; therefore, it is called a rate-1/2 coder (2) each input bit is stored for two clock cycles; therefore, the constraint length K is said to be two. It is intuitively obvious that a larger K yields better performance because it stores more elements from the past into the registers as redundancy for the current element.

For each possible state and each possible input bit, we need to know which output bits result so maximum-likelihood decoding can be accomplished. Such transitions are easily represented by a trellis diagram and are shown in Fig. 3 for K = 2.



Typical branch is labelled as $\xrightarrow{a_i/b_{i1}b_{i2}}$

Fig 2. Trellis representation of rate-1/2, K = 2 convolutional encoder,

For the K = 4 case, the trellis involves a rather large 16 states which is similar to Fig 2. Output bits b_i are trivially obtained via $b = aG$ where G is a generator polynomial. The G matrices are shown below for the two cases considered: K = 2 and K = 4, both of which are rate 1/2 code. They were chosen because they are optimal for rate-1/2 in the sense of maximal free distance. For K = 2, the free distance is 5 and for K = 4, the free distance is 7 [Lin].

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}^t, K = 2$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}^t, K = 4$$

A very popular way to decode convolutional codes is by maximum-likelihood sequence estimation (MLSE) using the Viterbi algorithm. The VA (dynamic programming) is the optimum symbol-by-symbol maximum-likelihood decision method for a bandlimited multipath channel with intersymbol interference. The overall system diagram is shown below in Fig 3.

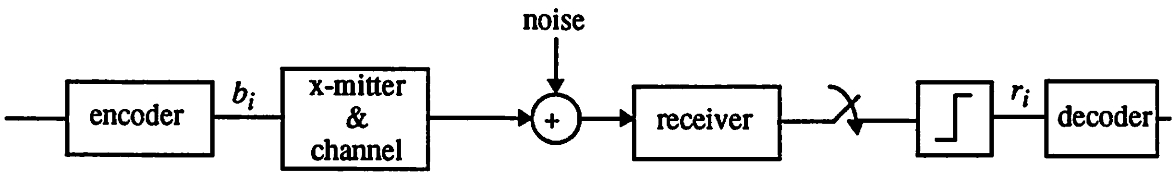


Fig 3. Block diagram of communication system showing convolutional encoder and decoder.

For clarity and simplicity, we demonstrate in detail the rate-1/2, $K = 2$ case. The objective of our VA is to recreate the trellis of Fig 2. Note that each node has two paths entering and exiting. Figure 4shows the two entering paths for the "00" node at some time n.

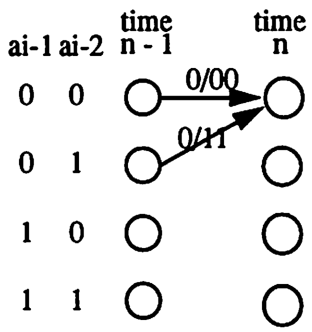


Fig 4. Showing two input paths into node "00" at time n.

At time n -1, the accumulated metric up to that time and the corresponding path back for that metric is stored in each of the four nodes. We then form partial path metrics for the two branches coming into node "00", given by

$$\rho_{i,0} = |r_{i1}| + |r_{i2}|$$

$$\rho_{i,1} = |r_{i1} \oplus 1| + |r_{i2} \oplus 1|$$

where r_i are the hard decisions out of the slicer in Fig 4. Then, at time n, we consider all incoming paths to a node. For each path, we form the sum of the accumulated metric up to time n-1 and the partial path metric. The path with the smallest sum is retained while the others are discarded - the essence of the VA is to keep the amount of information reasonable and still provide maximum-likelihood solutions. After repeating this for M steps, we look at all four nodes and pick the smallest accumulated metric, backtrace M steps and output resulting bits. The final consideration in implementing the VA is how long to wait before making a final decision. We consider the best path coming into each node and choose a truncation depth M, meaning we will look back M time steps to obtain the transmitted bit sent at that time. Ideally for maximum likelihood decoding, we would only start decoding when all the bits are received, this implies M is infinite. But to avoid high latency in practice, M can be set to a few integer multiples of the number of states.

Even for the rate-1/2, $K = 4$ case, which functions analogous to the one described, the process starts to become unwieldy. This is one disadvantage of a full-blown MLSE on a general channel - its complexity is just too large. Methods of mitigating this effect include truncating the overall channel impulse response [Falconer] or a more recent idea of combining equalization and the MLSE[Eyuboglu].

2.2 Block Code

This project is mostly concentrated on analysis and implementation of block codes. Unlike convolutional codes, the block of n code digits generated by an (n, k) block code encoder in any particular time unit depends only on the block of k input message digits within that time unit; whereas in a convolutional code, the block of n code digits depends not only on the block of k message digits within that time unit, but also message digits in the previous time units.

2.3 Linear Block Code

An (n, k) linear binary block code is a set of all linear combinations of k independent vectors in a vector space V with dimension n that maps k bits to n bits (Fig 5), where redundancy has been introduced in n-k bits.

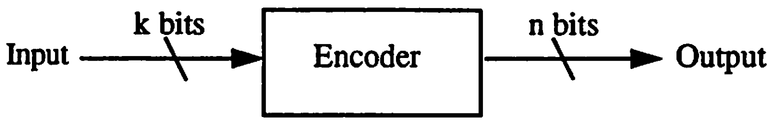


Fig 5. Block Encoder

The k input bits are called *information bits*, whereas n-k bits are called *parity-check bits*. Based on an encoding algorithm together with the number of redundancy bits, a block code can correct various numbers of bit errors within a block. For example, the Hamming(7, 4) code corrects a single bit error out of a block of 7 bits, within which 4 are message bits, 3 are parity check bits. The Golay(23, 12) code can correct up to 3 bit errors in any order within a block of 12 message bits together with 11 parity check bits.

An (n, k) binary code has 2^k codewords in it. Since a code is a vector subspace, it can be given a basis. The matrix whose rows are the basis vectors is called the *generator matrix*. Since a vector space has more than one basis, a code can have more than one generator matrix. For example, a Hamming(7, 4) code has the generator matrix

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

Given a generator matrix, the encoding process of a message x is $y = x \cdot G$, where y is the encoded message with n bits that shall be transmitted over a communications channel. It is worth mentioning here that there has not yet been a systematic way to find a generator matrix with a given error correction specification.

At the receiving side for the Hamming (7, 4) code, the decoding algorithm requires three decoding vectors [Pless]:

u = [0 0 0 1 1 1 1]
v = [0 1 1 0 0 1 1]

w =[1 0 1 0 1 0 1]

The following example will demonstrate the complete decoding process of the Hamming (7, 4) code. Suppose the transmitted message is x=1000; the encoded message is therefore xG=1000011. Assume an error occurred on the third bit during the transmission, so the actual received sequence is y=1010011. Then we take the inner products $y \cdot u$, $y \cdot v$, and $y \cdot w$ defined as $x \cdot y = \sum_{i=1}^{n} x_i \cdot y_i \pmod 2$; the result is $y \cdot u = 0$, $y \cdot v = 1$, $y \cdot w = 1$. Reading with this order, 011 is the number "3" represented in binary, which implies the third bit is an error. To correct the error, the receiver would then flip the third received bit. This decoding algorithm seems to work quite magically; in order to have a thorough understanding of it, we need more definitions.

A set of equations that gives the redundancy positions in terms of the information positions are called parity check equations. If the generator matrix G has dimension k for an (n, k) block code, the subspace generated by the redundancy bits would have dimension n-k and can be generated by an $(n-k) \times n$ matrix H called the *parity check matrix*. It is easy to prove that H is orthogonal to generator matrix G within a given block code. For every (n, k) block code, after performing matrix row operations, we can rewrite G in a standard form G=(I, A), where I is a $k \times k$ identity matrix, and A is a $k \times (n-k)$ matrix. Notice that the generator matrix for the Hamming code given previously is written in the standard form. It can be proved that if an (n, k) code has a generator matrix G=(I, A), then the parity check matrix is H=(-A', I), where A' is the transpose of A; as a result, H is an $(n-k) \times n$ matrix.

For an encoded message y, $y = x \cdot G$ for information x, then $y \cdot H' = (x \cdot G) \cdot H' = x \cdot (G \cdot H') = 0$. This is the same as saying if the received vector y is one of the valid codewords, i.e. $y = x \cdot G$ for some x, then $y \cdot H' = 0$. This observation can be applied to every received message to check if it is a valid codeword. Suppose the channel has introduced arbitrary errors, so the actual received signal is $\tilde{y} = y \oplus e$, where e corresponds to the error sequence. Applying the decoding algorithm we get $\tilde{y} \cdot H' = (y \oplus e) \cdot H' = e \cdot H'$. By cleverly performing matrix row operations (or column operations) on H, we can have H rearranged in such a way that the result of $e \cdot H^t$ is a function of error positions. This decoding algorithm is called syndrome decoding, and the syndrome of y is defined as $Syn\ (y) = y \cdot H^t$.

The (7,4) Hamming code can correct 1 bit out of 7 transmitted bits. Some more elaborate codes such as BCH(15, 7) code can correct up to 2 bit errors from any position within a block of 15 bits. This is different from considering two Hamming codes sitting next to each other as one double error correction code; for two Hamming codes to correct two errors, each error has to reside precisely within each block of 7 bits. The difficulty in correcting multiple consecutive errors is the reason it took so long to develop a multiple-error-correction algorithm after the discovery of a single error correction algorithm. The generator matrix for BCH(15, 7) code is a 7×15 matrix, thus the parity check matrix has dimension 8×15. To decode BCH(15, 7) codes, we apply the syndrome decoding algorithm.

We start with a parity check matrix of 8 rows with the following form:

$$H = \begin{bmatrix} 1 & 2 & \cdot & \cdot & 15 \\ f(1) & f(2) & \cdot & \cdot & f(15) \end{bmatrix}$$

where the numbers 1, 2, $\cdot \quad \cdot$, 15 are 4-tuples in binary, and f is a function which has yet to be determined. If y is the received vector with 2 errors occurring in the i th and j th columns, the syndrome of y is $Syn\ (y) = \begin{bmatrix} i+j \\ f(i) + f(j) \end{bmatrix} = \begin{bmatrix} y_1 \\ y_3 \end{bmatrix}$, which corresponds to two equations with two unknowns. Before being able to solve these equations, we have to determine the function f. We know that f can not be the identity function, since $y_1 = y_3 = i + j$ will not give a unique solution for i and j. Next we try $f(i) = i^2$. Since the space of possible codes is defined on a binary field, $i^2 + j^2 = (i + j)^2$, again, we have the same problem as we had with the identity function. Let us try $f(i) = (i)^3$, The syndrome of y gives

$$Syn\ (y) = \begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} i+j \\ i^3 + j^3 \end{bmatrix}$$

and $y_3 = i^3 + j^3 = (i+j)(i^2+ij+j^2)$ so that $y_3/y_1 = i^2+ij+j^2 = y_1^2+ij$. Noticing that $i+j = y_1$, and $ij = -y_1^2+y_3/y_1 = y_1^2+y_3/y_1$, we see that i and j are roots of the equation $(x+i)(x+j) = x^2+(i+j)x+ij = x^2+y_1x+(y_3/y_1+y_1^2)$. We know the coefficients of this quadratic equation, so we can solve for its roots. Once again, the solution for i and j will correspond to the bit error positions.

It is worthwhile to state that if a codeword C has minimum Hamming distance ***d***, where Hamming distance is defined as the number of bits by which two distinct code vectors differ, then C can correct $t = [(d-1)/2]$ or fewer errors. Therefore, when constructing a code, we should always aim for the greatest minimum distance to achieve best performance.

2.4 Cyclic Code

One very useful subclass of linear block codes is cyclic code. Cyclic code not only inherits the algebraic structure of the linear block code, its encoding process can also be achieved easily with a shift register implementation.

A cyclic code is defined as follows: if $v = (v_0, v_1, \ldots, v_n)$ is code vector of C, then the shifted v, $v^{(1)} = (v_{n-1}, v_0, \ldots, v_{n-2})$ is also a code vector. Consequently, $v^{(i)} = (v_{n-i}, v_{n-i+1}, \ldots, v_0, \ldots, v_{n-i-1})$ is a code vector. To explore the algebraic structure of cyclic code, we shall write the components of a code vector as coefficients of a polynomial as follows:

$$v(X) = v_0 + v_1X + \ldots + v_{n-1}X^{n-1}$$

therefore,

$$v^{(i)}(X) = v_{n-i} + v_{n-i+1}X + \ldots + v_{n-i-1}X^{n-1}$$

With this polynomial representation, the *generator polynomial* of an (n, k) cyclic code is defined as:

$$g(X) = 1 + g_1X + g_2X^2 + \dots + g_{n-k-1}X^{n-k-1} + X^{n-k}$$

such that every code polynomial is a multiple of g(X), and moreover, every polynomial of degree n-1 or less which is a multiple of g(X) must be a code polynomial. Using this definition, every code polynomial v(X) in an (n, k) cyclic code can be expressed as

$$v(X) = m(X)\,g(X) = (m_0 + m_1X + \dots + m_{k-1}X^{k-1})\,g(X)$$

where $(m_0, m_1, \dots, m_{k-1})$ are the k information digits and v(X) = m(X) g(X) is the corresponding code polynomial. Thus, the encoding of message m(X) is equivalent to multiplying the message m(X) by g(X), and an (n, k) code is completely specified by the generator polynomial g(X).

It can be proved that the generator polynomial g(X) of an (n, k) cyclic is a factor of $x^n + 1$, which means

$$x^n + 1 = g(X)\,h(X) \qquad \text{for some } h(X).$$

h(X) is call the *parity polynomial* of cyclic code generated by g(X). With the above relation, we can answer the question: for any n and k, does there exist an (n, k) cyclic code? The answer is, if an irreducible polynomial g(X) (irreducible means g(X) can not be factored as a product of two polynomials) of degree n-k is a factor of x^n+1, then g(X) generates (n, k) cyclic code. For example, $x^7+1 = (1 + X + X^3)(1 + X + X^2 + X^4)$, so the generator polynomial $g(X) = 1 + X + X^3$ generates a (7, 4) code of $d_{min} = 3$. This (7, 4) code is the Hamming(7, 4) code used in the earlier example. Using g(X) to encode message 1100, we would have

$$(1+X)(1 + X + X^3) = 1 + X + X^3 + X + X^2 + X^4 = 1 + X^2 + X^3 + X^4,$$

where 1+X corresponds to the message polynomial, the resulting code word is therefore 1011100.

Given that $x^n + 1 = g(X)\,h(X)$ for some h(X), we see an (n, k) cyclic code is also completely determined by parity polynomial h(X). Let $v(X) = v_0 + v_1X + v_2X^2 + \dots + v_{n-1}X^{n-1}$ be a code polynomial, where $v_{n-k}, v_{n-k+1}, \dots, v_{n-1}$ are the k information digits and $v_0, v_1, \dots, v_{n-k-1}$ are the n-k parity check digits. Then v(X) can be written as v(X) = p(X) q(X). Multiplying v(X) by h(X), we have

$$\begin{aligned} v(X) &= q(X)\, g(X)\, h(X) \\ &= q(X)\, [X^n + 1] \\ &= X^n\, q(X) + q(X) \end{aligned}$$

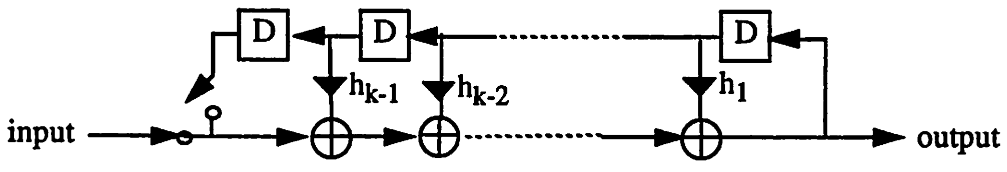
Since the degree of q(X) is k-1 or less, the powers of $X^k, X^{k+1}, ..., X^{n-1}$ do not appear in X^n q(X) + q(X). That is to say, in the expansion of v(X) h(X), the coefficients of $X^k, X^{k+1}, ..., X^{n-1}$ must be zero. Writing this in equation form, we have:

$$\sum_{i=0}^{k} h_i v_{n-i-j} = 0 \qquad \text{for } i<=j<=n-k$$

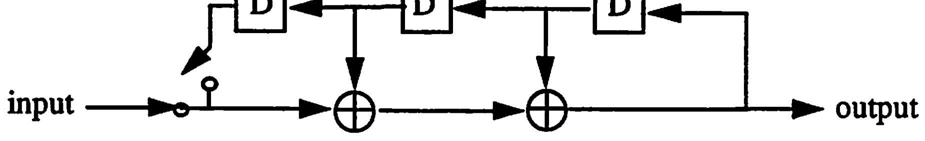
then the first term of v is given by

$$v_{n-k-j} = \sum_{i=0}^{k-1} h_i v_{n-i-j}, \text{ for } h_k = 1$$

Given the k information digits $v_{n-1}, v_{n-2}, ..., v_{n-k}$, the above equation is a rule to determine the n-k parity check digits $v_{n-k-1}, v_{n-k-2}, ..., v_0$. Thus the (n, k) cyclic code generated by g(X) is also completely specified by h(X) = (X^n + 1) / g(X). The encoding circuit based on this relation is shown below:



where all the register Ds are initialized to zero. The (7, 4) Hamming code will then have the encoding structure:



At receiver side, the received polynomial can be expressed as

$$\begin{aligned} r(X) &= v(X) + e(X) \\ &= m(X)\, g(X) + e(X), \end{aligned}$$

where v(X) is the code polynomial and e(X) is the error polynomial representing the errors during the transmission. r(X) can also be expressed as

$$r(X) = p(X)\, g(X) + s(X),$$

where s(X) is the remainder of r(X) divided by the code generator polynomial g(X). Combine these two equations, we would get the result:

$$e(X) = [p(X) + m(X)]\, g(X) + s(X),$$

Clearly, s(X) equals to the remainder of e(X) divided by g(X).

If the errors of e(X) are confined to n-k parity check positions, 1, X, ..., X^{n-k-1}, then e(X) is a polynomial of degree n-k-1 or less, thus e(X) = s(X). Therefore, the error correction can achieved simply by adding (modulo 2) the syndrome to the n-k received parity check bits. In practice, the message digits are the portion of a block that need protection. Suppose that the errors are confined to X^i, X^{i+1}, ..., $X^{(n-k)+i-1}$; after n-i cyclic shifts of r(X), the errors will be shifted to the n-k parity check positions of the cyclically shifted received vector $r^{(n-i)}(X)$ and errors can be corrected the same way as in the earlier case.

2.4.1 The BCH (15, 5, 7) Code

BCH(15, 5, 7) is a triple error correction code that is capable of correcting 3 errors in any combination within a block of 15 bits. In this section, readers are assumed to have a certain familiarity with basic finite field theory. A couple of theorems on finite fields will be restated in this section for the purpose of understanding of the BCH(15, 5, 7) code.

In a finite field of q elements, GF(q), there is a primitive element α, which is an element of order q-1 such that every nonzero element of GF(q) can be expressed as a power of α. That is to say, the multiplicative group of GF(q) is cyclic. Let α be a root of an irreducible polynomial $g(X) = X^4 + X + 1$, and therefore, α is a primitive element of $GF(2^4)$. $GF(2^4)$ is a finite field of 16 elements, and the presentation of the 15 nonzero field elements is given in Table1; notice that every element is expressed as a power of α.

α^0	1				1 0 0 0
α^1		α			0 1 0 0
α^2			α^2		0 0 1 0
α^3				α^3	0 0 0 1
α^4	1+	α+			1 1 0 0
α^5		α+	α^2		0 1 1 0
α^6			α^2+	α^3	0 0 1 1
α^7	1+	α+		α^3	1 1 0 1
α^8	1+		α^2		1 0 1 0
α^9		α+		α^3	0 1 0 1
α^{10}	1+	α+	α^2		1 1 1 0
α^{11}		α+	α^2+	α^3	0 1 1 1
α^{12}	1+	α+	α^2+	α^3	1 1 1 1
α^{13}	1+		α^2+	α^3	1 0 1 1
α^{14}	1+			α^3	1 0 0 1
α^{15}	1=	α^0			

Table 1: Representation of finite field elements

The BCH(15, 5, 7) code is defined by its parity check matrix,

$$M = \begin{bmatrix} 1 & 1 & 1 \\ \alpha & \alpha^3 & \alpha^5 \\ \alpha^2 & \alpha^6 & \alpha^{10} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \alpha^{14} & (\alpha^3)^{14} & (\alpha^5)^{14} \end{bmatrix} \qquad \text{Eqn 1}$$

Since $GF(2^4)$ is a cyclic code, we have $\alpha^{15} = \alpha^0 = 1$, terms such as $(\alpha^3)^{14}$ can be simplified as $(\alpha^3)^{14} = \alpha^{42} = \alpha^{12}$. Referring to Table 1 and Eqn 1, the binary representation of the parity check matrix is:

$$M = \begin{bmatrix} 1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&1&0&1&1&0 \\ 0&0&1&0&0&0&1&1&1&1&1&0 \\ \cdot&\cdot&&&&&&&&&&\cdot \\ \dot{1}&\dot{0}&0&1&1&1&1&1&1&1&1&\dot{0} \end{bmatrix} \qquad \text{Eqn 2}$$

For a receiving vector $r = (r_0, r_1, ..., r_{14})$, the polynomial representation is $r(X) = r_0 + r_1X + r_2X^2 + ... + r_{14}X^{14}$. Multiplying r with the matrix M in Eqn 1, $rM = [r(\alpha), r(\alpha^3), r(\alpha^5)]$, where the first component $r(\alpha)$ is:

$$r(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + ... + r_{14}\alpha^{14}$$

Suppose errors occur, which is the same as saying $r = y + e$, where y is a code vector. Written in terms of a polynomial, we would have $r(X) = y(X) + e(X)$. Therefore, the result of parity check calculation *rM* is

$$rM = [y(\alpha)+e(\alpha), y(\alpha^3) + e(\alpha^3), y(\alpha^5) + e(\alpha^5)] = [e(\alpha), e(\alpha^3), e(\alpha^5)] \qquad \text{Eqn 3}$$

because α is a root of the generator polynomial and y is a code polynomial, so that $y(\alpha) = 0$. The result of rM is called the syndrome of a received code; from Eqn 3, we see it is a nonlinear function of error location. The standard way of expressing this is $rM = [S_1, S_3, S_5] = [e(\alpha), e(\alpha^3), e(\alpha^5)]$. Existing theorems in error correction showed that:

$$\begin{aligned} &S_1 - \sigma_1 = 0 \\ &S_2 - S_1\sigma_1 + 2\sigma_2 = 0 \\ &S_3 - S_2\sigma_1 + S_1\sigma_2 - 3\sigma_3 = 0 \\ &S_4 - S_3\sigma_1 + S_2\sigma_2 - S_1\sigma_3 + 4\sigma_4 = 0 \\ &S_5 - S_4\sigma_1 + S_3\sigma_2 - S_2\sigma_3 + S_1\sigma_4 - 5\sigma_5 = 0 \end{aligned} \qquad \text{Eqn 4 [Peterson]}$$

where σ_is are called the *elementary symmetric functions.*

Assuming there are three or less errors in a block of 15 bits, it has been shown that if errors occur in the received vector, the error position numbers X, $X \in \{\alpha^0, \alpha^1, \alpha^2, \cdot, \cdot, \ldots, \alpha^{14}\}$ must satisfy the equation:

$$f(X) = X^3 - \sigma_1 X^2 + \sigma_2 X - \sigma_3 = 0 \qquad \text{Eqn 5}$$

For example, if the 2nd bit has an error in it, substitute α^2 into Eqn 5, then $f(\alpha^2) = (\alpha^2)^3 - \sigma_1(\alpha^2)^2 + \sigma_2(\alpha^2) - \sigma 3 = 0$. In BCH(15, 5, 7) code, all operations are performed in the binary field, so addition and subtraction are functionally equivalent. Moreover, because of the binary field, it can be shown that $S_2 = S_1^2$. As this point, we conclude that if we can compute the elementary symmetric functions, the error locations can be found by substituting each bit position into Eqn 5 and testing if $f(\alpha^i)=0$, for i=0, 1, .., 14. To correct errors, we would only need to flip the appropriate bit which satisfies $f(X)=0$. Thus, the problem of error correction is completely focused on how to compute the elementary symmetric functions σ_i's.

From Eqn 4 in this section, the syndrome (computed from the received vector) is related to σ_i by

$$S_1 - \sigma_1 = 0,$$
$$S_2 - S_1\sigma_1 + 2\sigma_2 = 0,$$
etc.

Notice this is a system of linear equations with variables σ_i's which it can be solved as:

$$\sigma_1 = S_1$$
$$\sigma_2 = (S_1^2 S_3 + S_5)/(S_1^2 + S_3) \qquad \text{Eqn 6}$$
$$\sigma_3 = (S_1 S_5 + S_3^2 + S_1^3 S_3 + S_1^6)/(S_1^3 + S_3)$$

provided that $S_1^2 + S_3$ does not equal to zero.

As a numerical example, suppose the vector of all zeros is transmitted, and that errors occur in the 3rd, 4th, and 9th positions. Then

$r = (0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0)$

$rM = (S_1, S_3, S_5) = (1\,0\,0\,0, 0\,1\,0\,1, 0\,1\,1\,0)$

where $S_1 = (1\ 0\ 0\ 0)$, $S_3 = (0\ 1\ 0\ 1)$, and $S_5 = (0\ 1\ 1\ 0)$

Referring to Table 1, $S_1 = 1$, $S_3 = \alpha^9$, $S_5 = \alpha^5$. Substitute syndromes into Eqn 3, then

$\sigma_1 = 1$
$\sigma_2 = (S_1{}^2S_3 + S_5)/(S_1{}^2 + S_3) = (\alpha^9 + \alpha^5)/\alpha^9 = 1 + \alpha^{-4} = 1 + \alpha^{11} = \alpha^{12} = 1111$
$\sigma_3 = (S_1S_5 + S_3{}^2 + S_1{}^3S_3 + S_1{}^6)/(S_1{}^3 + S_3) = \alpha^{14} = 1001.$

Putting these coefficients into f(X), it is then easy to verify that $f(X) = X^3 + X^2 + \alpha^{12}X + \alpha^{14} = 0$ is satisfied for α^3, α^4, and α^9.

The BCH(15, 5, 7) triple error correction code provides very good protection over data especially in an indoor wireless communication environment where errors occur frequently. A custom VLSI implementation of the BCH(15, 5, 7) code will be shown in chapter 5, and this design will be used in the Berkeley InfoPad project.

3

Simulation Using Ptolemy

3.1 Theoretical Analysis of Hamming Code over a Fading Channel

The applications we consider in this project are generally in the area of indoor communications, where we can assume the amplitude of the spectrum is constant over a block duration of 7 bits, which is the block size of Hamming code. In an indoor environment, the receiver such as the InfoPad will not move a significant amount during one block time, so Doppler shifts which usually occur in wireless communication is not a significant issue throughout our analysis. For the (7,4) Hamming code over a fading channel using OOK modulation that we are investigating, an error occurs if there is more than one error in each block of seven bits. Therefore the probability of correct detection is given by

$$P_c = \binom{7}{0}(1 - Q(\sqrt{SNR}))^7 + \binom{7}{1}(1 - Q(\sqrt{SNR}))^6 (Q(\sqrt{SNR})) \quad \text{Eqn 1}$$

thus, the probability of error is

$$P_e = 1 - P_c \quad \text{Eqn 2}$$

where SNR is given by

SNR = instantaneous power / (noise power + ISI power) Eqn 3

and the ISI is assumed to be Gaussian. This development can also be derived by from the Rician channel model [Linnartz] by letting Rician K factor equal to zero. The Rician equation is therefore reduced to:

$$P_e = \int_0^{\infty} \frac{e^{-\frac{p_0}{\overline{p_0}}}}{\overline{p_0}} \sum_{m=0}^{1} \binom{7}{m} \left(1 - Q\left(\sqrt{\frac{p_0 T_b}{N_0 + \bar{p}_t T_b}}\right)\right)^{7-m} \left(Q\left(\sqrt{\frac{p_0 T_b}{N_0 + \bar{p}_t T_b}}\right)\right)^{m} dp_0 \qquad \text{Eqn 4}$$

where p_0 is the instantaneous power, $\overline{p_0}$ is the local mean power, and $\bar{p}_t$ is the variance of the ISI, which we assume is Gaussian. Note the term

$$\frac{p_0 T_b}{N_0 + p_t T_b} \qquad \text{Eqn 5}$$

equals the SNR term of equation 2; therefore, this expression of P_e agrees exactly with Eqn 1. For the simulations, we consider the impulse response of Fig. 7; for simulation efficiency, the discrete equivalent (shown in Fig. 8) of the combination of transmitter, the channel, and receiver is used. We included a gain factor to make the discrete equivalent channel unit-energy which is similar as the normalization term $\frac{1}{\overline{p_0}}$ in Eqn 4

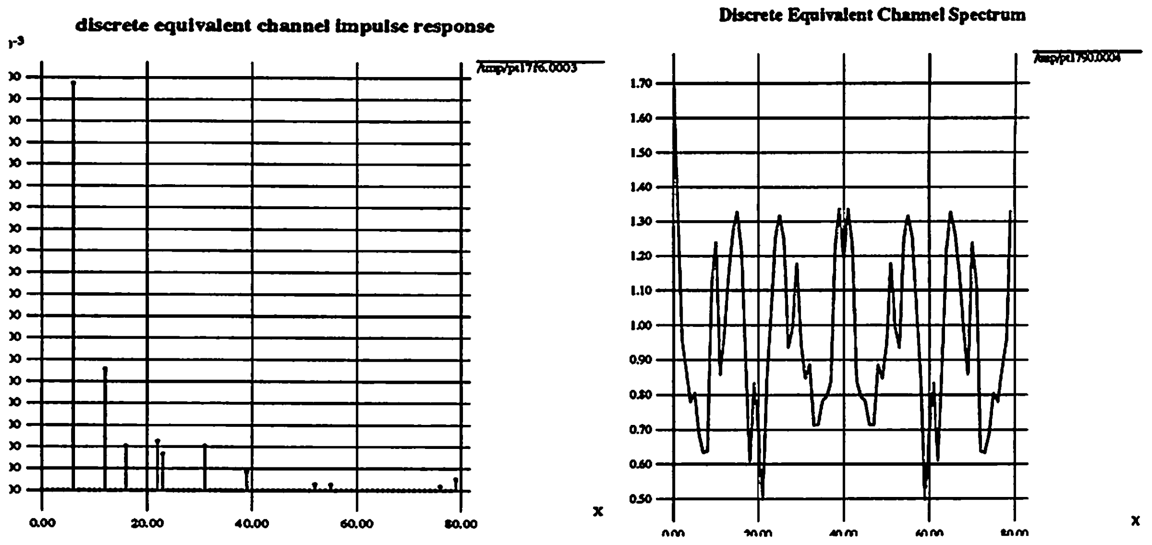


Fig 1. Impulse and spectrum of the discrete equivalent channel

In the next section, we varied the noise assuming some fixed instantaneous power. Noting equation 3, we see that the SNR varies as the carrier frequency changes, which will occur in mobile communications applications since different terminals will communicate with differing carrier frequencies. The spectrum of Fig. 8 shows that the SNR varies by about ±3dB over the various choices of carrier frequency one may choose for this channel.

The noise variance is chosen to be 0.005, which is at the higher end of noise power used in the simulation, with respect to a signal power of 0.5. We found the ISI power via the following relation:

$$\sigma^2_{isi} = \sum_{n \neq 0} h^2(n) \qquad \text{Eqn 6}$$

Then, from (Eq 3), we derive

$$SNR = \frac{0.5|H(\omega)|^2}{\sigma^2_{isi} + N_0} \qquad \text{Eqn 7}$$

For each sample taken uniformly across the spectrum of Fig 9 (each of which could be considered a different carrier frequency), the probability of error is computed according to Eq 1. The result of this computation is displayed in Fig. 9.

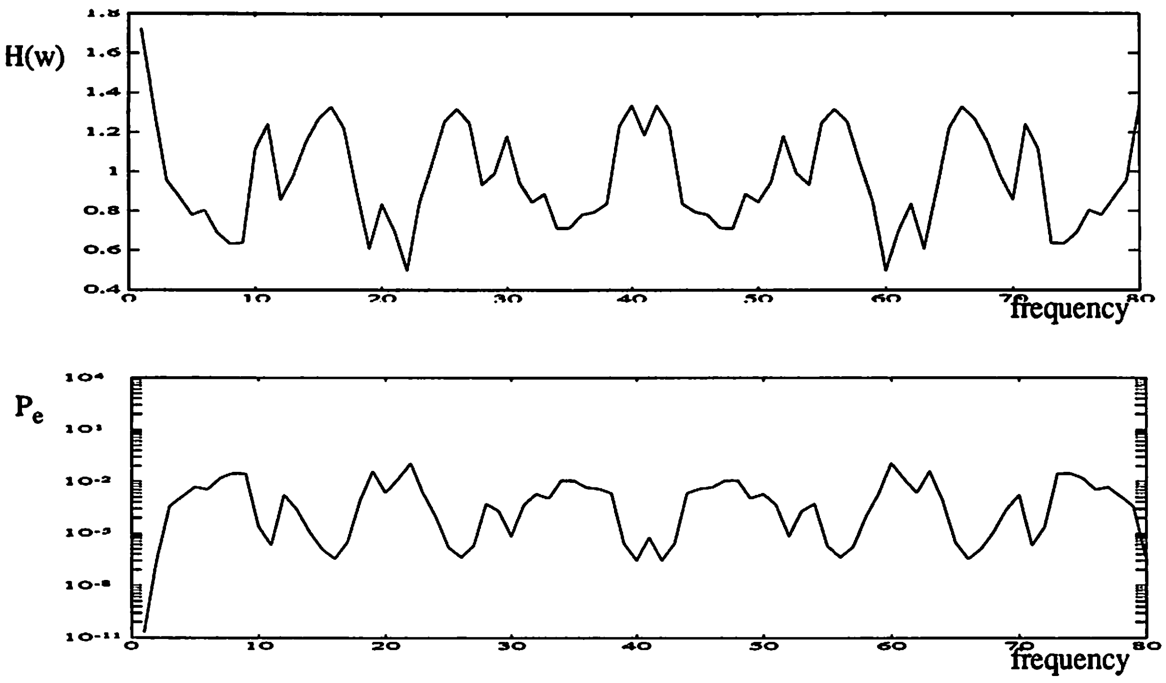


Fig 2. Theoretical probability of error over spectrum of multipath channel

As expected, at the frequencies that correspond to local maxima (peaks) of the spectrum which imply high SNR, a low BER is observed. And likewise, at the "nulls" of the spectrum, we observe a very high BER. Furthermore, we see that probability of error varies in the range of 10-2 to 10-6 which matches our simulation results shown in the later sections. The low-end noise power (i.e. $N_0 = 0.0005$) is also used to test the model. The possible inaccuracies introduced by this analytical model is the assumption that ISI terms from our multipath channel follow a Gaussian distribution. Since our result agrees with the simulation, we believe that our model is accurate. Moreover, as probability of error varies from 10^{-2} to 10^{-6} at different frequencies, therefore it is a good strategy to use spread spectrum in a multipath environment for an overall low bit error rate, and this indeed has been done in real implementations.

3.2 Simulations Setup

To simulate the entire system in Ptolemy, the basic building blocks include a random bit generator, an encoder, a noise source, a multipath fading channel, and a decoder. The communication system we simulated also consists of a transmitter filter and a receiver filter, both of which are raised cosine filters with 100% excess bandwidth. The multipath fading channel is generated from a Ptolemy simulation of an indoor channel as might be found in an application such as the InfoPad. Throughout the entire simulation, the sampling rate is set to 3 GHz and the carrier frequency is set to 1 GHz. The impulse response of the fading channel and its resulting spectrum are shown in Fig. 10 - notice it has Rayleigh characteristics of deep fades.

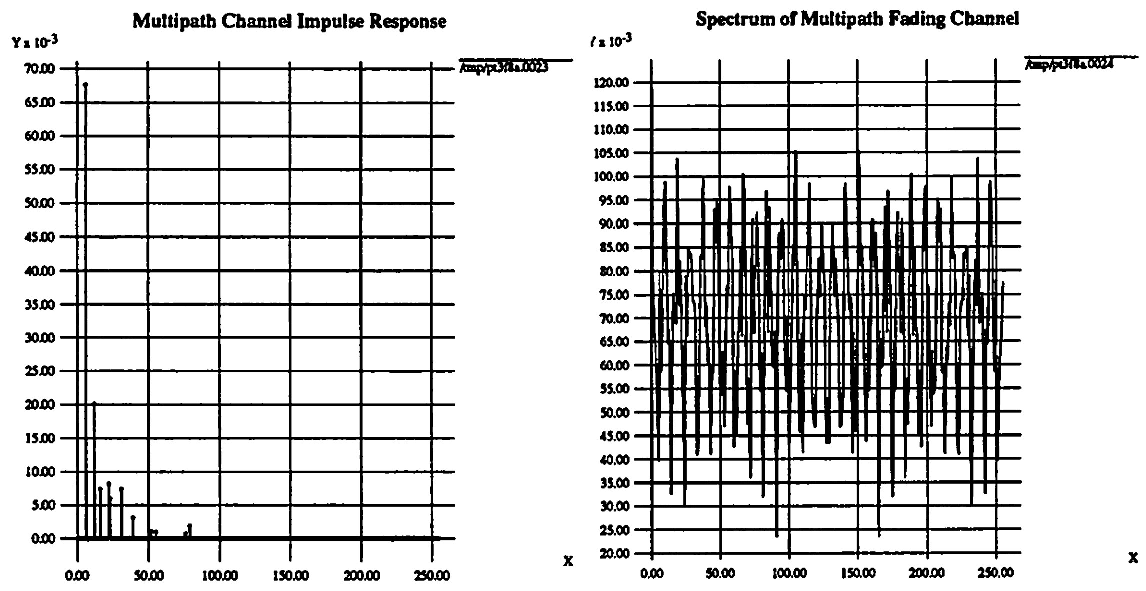


Fig 3. Impulse response of a fading channel with its spectrum

We wish to simulate the system as close to continuous time as possible. However, Ptolemy only simulates discrete systems. Therefore, the tactic of using several samples per baud is employed to make the system look like it is continuous time. Due to its large overhead, Ptolemy is not able to execute enough iterations in a reasonable time when these conditions are used. Fortunately, since the result of the receiver filter is sampled once per baud, the computation of the convolution of transmitter filter, multipath channel, and receiver filter can be condensed into a discrete-time equivalent channel. By making this discrete-equivalent response, simulation time is shortened dramatically and we can make meaningful BER tests on the order of 10 hours. Two other relevant issues that must be considered before running large simulations are timing and delay. When making the discrete-equivalent channel, we carefully located the maximum point of the output of the receiver filter and used that as the sampling point. In other words, we are doing perfect timing recovery for the system. Also, when checking to see if a particular transmitted bit is received properly, care must be taken to align transmitted bits and received bits properly. For example, our convolutional code requires 2 clock cycles before an input bit is correctly received.

Several "custom" stars are programmed in C++ for use in the Ptolemy environment. The code for these stars is included in the Appendix. These include encoders and decoders for each of the following:

Hamming (7,4) block code
rate 1/2 K = 2 convolutional code
rate 1/4 K = 4 convolutional code

3.3 Results

As an initial baseline, we tested a system with an ideal (no multipath) channel. In this simple case, analytical results are simply known and we easily checked that all was working properly. Table 1 shows the BER vs. signal-to-noise ratio (SNR) for the Rayleigh fading channel we described in Section 3.1. Figure 4 graphically displays the results. We used on-off keying modulation.

SNR	Uncoded	Hamming(7,4)	Conv(1/2, 2)	Conv(1/2, 4)
13.7	3.50×10^{-2}	1.63×10^{-2}		9.87×10^{-3}
14.1	3.30×10^{-2}	1.42×10^{-2}		7.66×10^{-3}
14.6	3.02×10^{-2}	1.17×10^{-2}		5.66×10^{-3}
14.9	2.79×10^{-2}	1.04×10^{-2}		5.42×10^{-3}
15.4	2.51×10^{-2}	9.11×10^{-3}		4.32×10^{-3}
16.4	1.99×10^{-2}	5.77×10^{-3}		2.64×10^{-3}
17.0	1.73×10^{-2}	4.33×10^{-3}		2.21×10^{-3}
18.5	1.11×10^{-2}	1.78×10^{-3}		9.8×10^{-4}
20.0	7.29×10^{-3}	6.91×10^{-4}	2.29×10^{-3}	5.18×10^{-4}
22.2	3.14×10^{-3}	1.14×10^{-4}	1.92×10^{-3}	1.65×10^{-4}
25.2	4.70×10^{-4}	3.75×10^{-6}	3.8×10^{-4}	2.85×10^{-5}
27.0	1.82×10^{-4}	8.3×10^{-7}	5.8×10^{-5}	6.5×10^{-6}
30.0	1.23×10^{-5}		4.25×10^{-6}	1.5×10^{-6}

Table 1: Simulation result for bit error rate vs. SNR

At high SNR, we can see the noticeable improvement of data after applying the Hamming(7, 4) code; however, at low SNR, the Hamming code actually made the data look worse! This makes sense after careful thinking about the capability of Hamming codes. The Hamming code is a single error correction code. If multiple bit errors occurred, the decoder would then flip a bit randomly, which in some cases, causes more errors. This is exactly the effect we observed at low SNR.

As expected, the performance improved as the constraint length of the convolutional code increased from 2 to 4. If the constraint length were made longer, we would be able to see some further improvement, this is what Qualcomm had actually implemented in their system where they used constraint length 7. On the decoding side, the truncation depth (how far to look back) of the Viterbi search algorithm may also be extended to obtain the theoretically predicted coding gain.

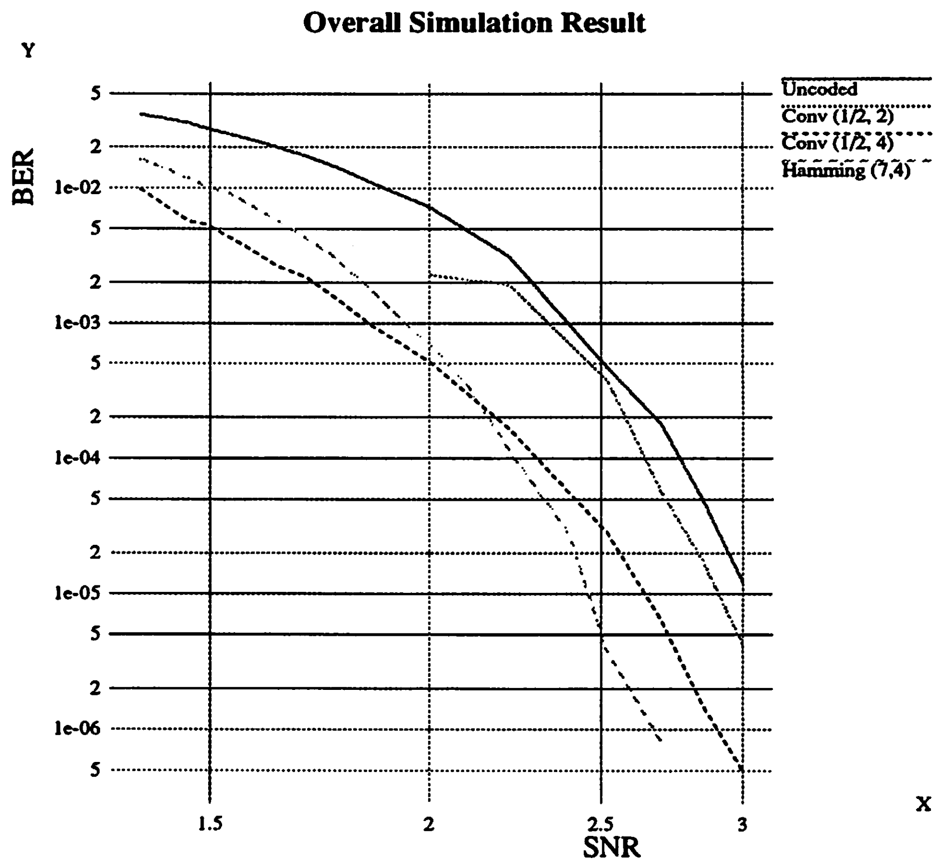


Fig 4. Bit Error Rate vs. SNR for Various Coding Schemes Over the Multipath Channel

Following closely to this idea, we carried over the accumulated path metrics after each iteration to the subsequent iteration. The data showed that the (1/2, 2) convolutional code gave a 1 dB improvement over an uncoded system; however, Hamming (7, 4) code and the (1/2, 4) convolutional code yielded an improvement of 4 dB over the uncoded system. It appears that the overall penalties due to multipath for this channel are severe.

The overall result showed that error correction algorithms are more powerful as the complexity increases, and this may impose a serious constraint during the real implementation. Fortunately, there are possibilities that we can build an efficient error correction scheme without introducing infinite complexity.

4

Analysis of Bit Error Rate

4.1 Performance of block codes

The error correction capability of a particular code is directly related to its block size and amount of redundancy. Assume independently identically distributed (i.i.d.) bit errors with probability P_e, which also equals the bit error rate (BER). An (n, k, d) block code is able to correct up to $t = \left\lfloor \frac{(d-1)}{2} \right\rfloor$ errors. Therefore, by applying error correction, transmission of a block of n bits is correct if there are t or fewer errors within the block. We can calculate the probability of correct transmission, (using our independence assumption) with the result:

P_c(with error correction) = P(0 error) + P(1 error) + ... + P(t errors)=

$$(1-P_e)^n + \binom{n}{1}(1-P_e)^{n-1}P_e + \binom{n}{2}(1-P_e)^{n-2}P_e^2 + \cdots + \binom{n}{t}(1-P_e)^{n-t}P_e^t$$

$$= \sum_{k=0}^{t} \binom{n}{k}(1-P_e)^{n-k}P_e^k$$

Thus, the probability of error after error correction is:

P_e(after correction) = 1 - P_c(after correction) = $1 - \sum_{k=0}^{t} \binom{n}{k}(1-P_e)^{n-k}P_e^k$.

Because P_e(t+2 errors) = P_e P_e(t+1 errors), we can conclude that P_e(t+1 errors) >> P_e(t+2 errors) for $P_e<10^{-2}$, which is to say that the probability of making t+2 errors is negligible relative to the probability of making t+1 errors. Therefore, the bit error rate after error correction of an (n, k, d) block code can be approximated as the probability of making (t+1) errors multiplied by number of bit errors, (t+1):

BER(after error correction) = (t+1) P_e(after correction).

Having this formulation, we are able to demonstrate the effectiveness of error correction in terms of BER for any particular channel. Fig 1 shows the bit error rate as result of error correction vs. the bit error rate without error correction for any channel with random errors for several coding schemes.

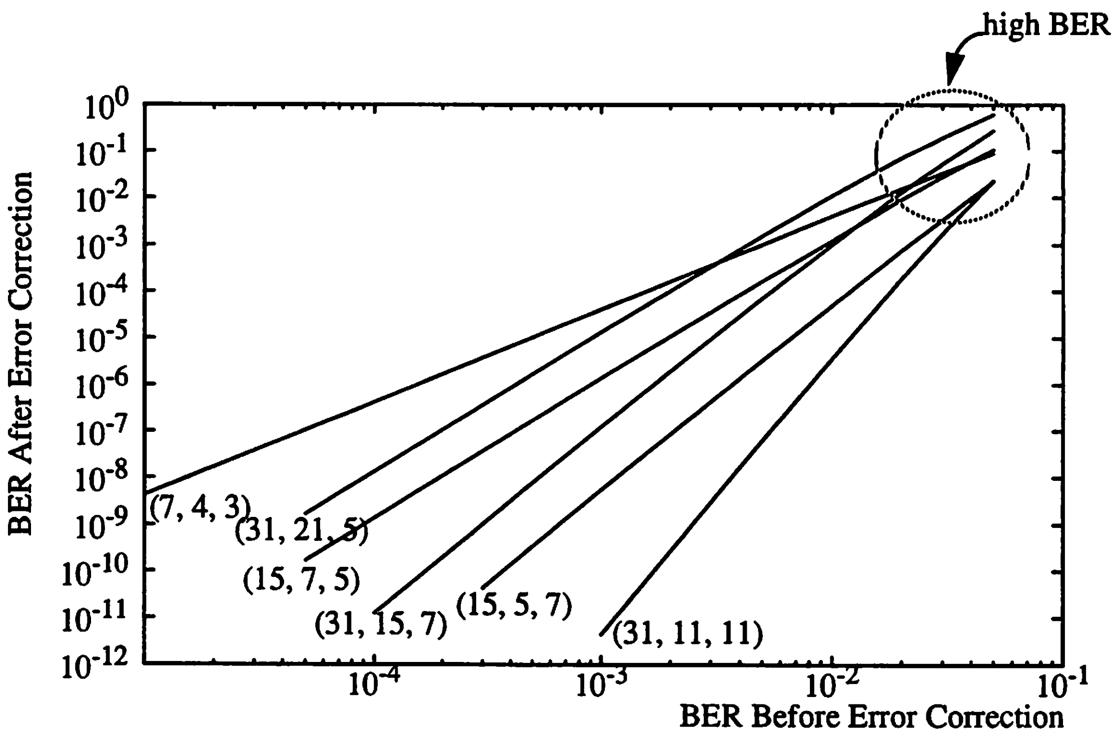


Fig 1 BER After Error Correction vs. BER before Error Correction

We see that if the channel has a high bit error rate, for example, BER = 0.05, there is very little improvement any error correction algorithm can do since the data is so corrupted. But at BER = 10^{-3} (which is a typical error rate for an indoor environment

[Sheng]), a code with high amount of redundancy such as (15, 5, 7) can improve BER from 10^{-3} to $5x10^{-9}$! From this figure, we see that error correction indeed improves the quality of transmission, and should be seriously considered when designing reliable communication systems.

Finally, it is important to mention that block codes work best when errors do not come in bursts because a single block code can not correct more than t errors. In an indoor wireless communication environment in which multipath dominates, burst errors are either caused by transmitting at a fading null, or they are caused by a blockage such as a person in the main transmitting path. Therefore, to achieve the optimum performance of block code, an interleaver is needed to re-distribute errors among the "good" data, and therefore, the burst errors would look like random errors. Fig 2 shows how an interleaver can fit in a communication system:

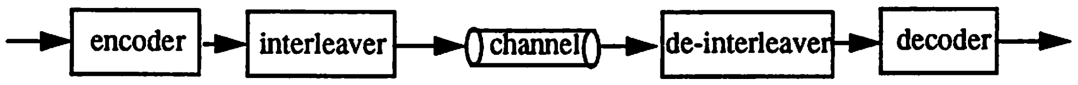


Fig 2. Use of an interleaver in a communication system

First, the interleaver shuffles the data before transmitting. At the receiving end, the de-interleaver re-shuffles the receiving data (which may include errors) back into the original order. This de-interleaving process rearranges the errors and makes them appear random, and our goal of having random bit errors is achieved.

4.2 Error detection with retransmission

Another common method for error control is to use error detection along with retransmission. Error detection is considerably less sophisticated in terms of implementation compared to error correction. Recalling syndrome decoding from the chapter 2, a block of data is received correctly if the syndrome of the received code vector equals to zero. Syndrome computation of block codes can be implemented easily using either AND and XOR gates, or with shift registers. Unlike error correction, error detection does not have to do any post syndrome computation for error locations; instead, it requires a closed loop system to send an acknowledgment to the transmitter which indicates the quality of the packet so retransmission is possible. The Fig 3 shows an outline of such a system:

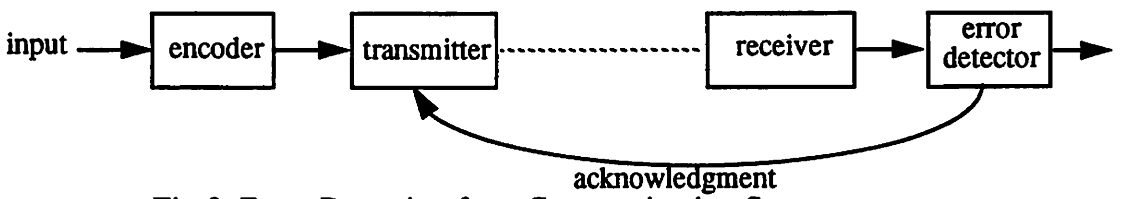


Fig 3. Error Detection for a Communication System

The error detector computes the syndrome for each received vector; if the syndrome equals to zero, i.e. no error occurred, the acknowledgment of correct transmission will be issued and sent to the transmitter. Otherwise, an acknowledgment of incorrect transmission is sent to transmitter for retransmission.

With this scheme, we can compute the effective bandwidth by:

effective bandwidth = E[number of transmissions] x Actual Bandwidth per transmission

where the first term stands for expected number of transmissions due to errors. The expected number of transmissions can be calculated as:

$$\begin{aligned} E[N] &= 1P_c + 2P_cP_e + 3P_cP_e^2 + 4P_cP_e^3 + \ldots.. \\ &= P_c(1 + 2P_e + 3P_e^2 + 4P_e^3 + \ldots.) \\ &= \frac{P_c}{(1-P_e)^2} \end{aligned}$$

One key point for error detection is that for an (n, k, d) code, one code vector is different from another code vector by the hamming distance d; because of this, error detection algorithm will fail to detect errors when there are exactly d, 2d, 3d, . . . errors. Thus,

P_c = P("correct" transmission) =

$$(1-P_e)^n + \binom{n}{d} P_e^d (1-P_e)^{n-d} + \binom{n}{2d} P_e^{2d} (1-P_e)^{n-2d} + \ . \ . \ .$$

The error detection with retransmission algorithm described here assumes the transmitter keeps retransmitting until a "correct" code vector is received. The probability of error after the "correct" reception is therefore,

P(error after retransmission) =

$$\binom{n}{d} P_e^d (1-P_e)^{n-d} + \binom{n}{2d} P_e^{2d} (1-P_e)^{n-2d} + \ . \ . \ .$$

Thus the BER after retransmission can be approximated as

BER = d P_e(error after retransmission)

As we have fully described both error correction and error detection schemes, it is time to use some quantitative examples to compare their performance.

We will constrain ourselves to the codes that have block size of 15 bits. And we want to compare two schemes by letting them have the same effective bandwidth, one due to the redundancy, and another due to the redundancy together with retransmission. Consider both (15, 11, 3) and (15, 7, 5) block codes for error detection/retransmission vs. (15, 5, 7) for error correction. The BER in the table is the bit error rate after retransmission until "correct" receiving and the bit error rate after applying error correction.

← detection/retransmission →← correction →

same effective bandwidth as (15, 5, 7) code

(15, 11, 3)	(15, 7, 5)	(15, 5, 7)

Pe = 0.058
E[N] = 2.215
BER = 0.0244
	Pe = 0.058
	E[N] = 2.444
	BER = 5.4x10-3
		Pe = 0.058
		
		BER = 0.0368

Pe = 0.04
E[N] = 1.786
BER =6.7x10-3
	Pe = 0.04
	E[N] = 1.844
	BER = 1.0x10-3
		Pe = 0.04
		
		BER = 9.8x10-3

Pe = 0.022
E[N] = 1.389
BER = 7.52x10-4
	Pe = 0.022
	E[N] = 1.396
	BER = 6.19x10-5
		Pe = 0.022
		
		BER = 1.1x10-3

Pe = 5x10^{-3}
E[N] = 1.078
BER = 2.42x10-6
	Pe = 5x10-3
	E[N] = 1.078
	BER = 4.46x10-8
		Pe = 5x10-3
		
		BER = 3.27x10-6

Pe = 1x10^{-3}
E[N] = 1.015
BER=4.05x10-9
	Pe = 1x10-3
	E[N] = 1.015
	BER = 1.487x10-11
		Pe = 1x10-3
		
		BER = 5.41x10-9

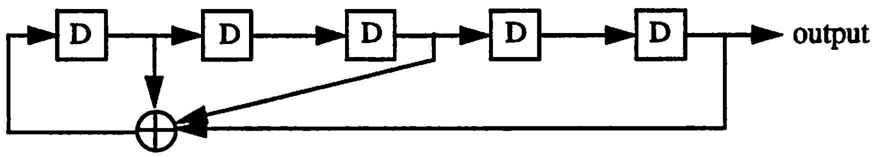
Table 1: error detection/retransmission vs. error correction

As indicated from the table, with the same effective bandwidth, error detection clearly has a gain over error correction by having the lower final BER after retransmission. But on the other hand, retransmission over the channel can introduce latency that may not meet the real time requirement, and this latency will also require the system to have a bigger buffer. As a result, the best compromise is probably to combine the error detection/retransmission with error correction. A proposal for such a system will be introduced in the section of future work.

5 Implementation

5.1 The BCH(15, 5, 7) encoder

As we have discussed in chapter 3, an encoder for a cyclic code can be implemented by shift registers according to its parity check polynomial. The parity check polynomial is $h(X) = 1 + X + X^3 + X^5$ for the BCH(15, 5, 7) code described in chapter 2. The shift-register implementation of the encoder is therefore:



After the five data bits are loaded as initial state, then the block (data bits and parity check bits) is generated from this circuit. This set of shift registers is easy to build in hardware, and will be done as a part of this project.

5.2 The BCH(15, 5, 7) decoder

Decoding of BCH codes requires computations which use Galois field arithmetic. Galois field arithmetic can be implemented similarly as ordinary arithmetic with the exception that it has no carries. For the BCH(15, 5, 7) code, the decoding process can be characterized by three major steps for a block of 15 received bits:

Step 1) Compute the syndrome S_1, S_3, S_5.

Step 2)Use the syndrome to calculate the elementary symmetric functions σ_i's. The resulting error location polynomial is then $f(X) = X^3 + \sigma_1 X^2 + \sigma_2 X + \sigma_3$.

Step 3) Sequentially substitute each of 5 message bits into the error location polynomial, and flip the bit if an error is detected, i.e. when $f(X)\big|_{x=\alpha^i} = 0$. The reason why only 5 bits are tested is because we do not need to correct parity check bits.

Step 1 can be accomplished either by the use of feedback shift registers, or by a straightforward implementation such as using AND for binary field multiplication and exclusive-OR for binary field addition. The calculations of elementary symmetric functions in Step 2 are computationaly intensive with arithmetic such as polynomial addition, multiplication, and division. Instead of having separate lookup tables for all of these operations, a 25k bit ROM is used as an overall lookup table. The decision of using a large ROM allows great simplifications on timing and scheduling between different processing modules. In addition, this ROM is also used as an error counter to count the number of bit errors up to 3 errors in a block of 15 receving bits. Lastly, the final step is done by shift registers with feedback. As a result, the top level system view is:

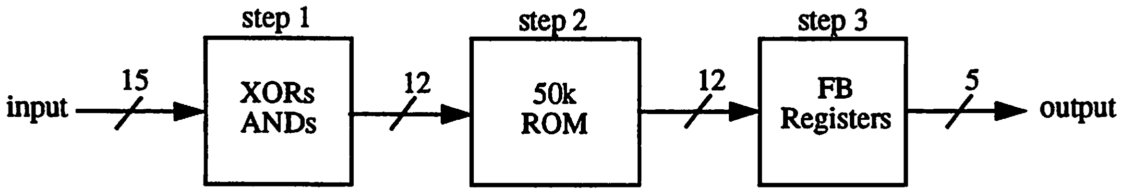


Fig 1. top level system view of implementation of the BCH(15, 5, 7) code

Implementing the BCH(15, 5, 7) decoder using the high level synthesis tool Lager, the schematic looks like:

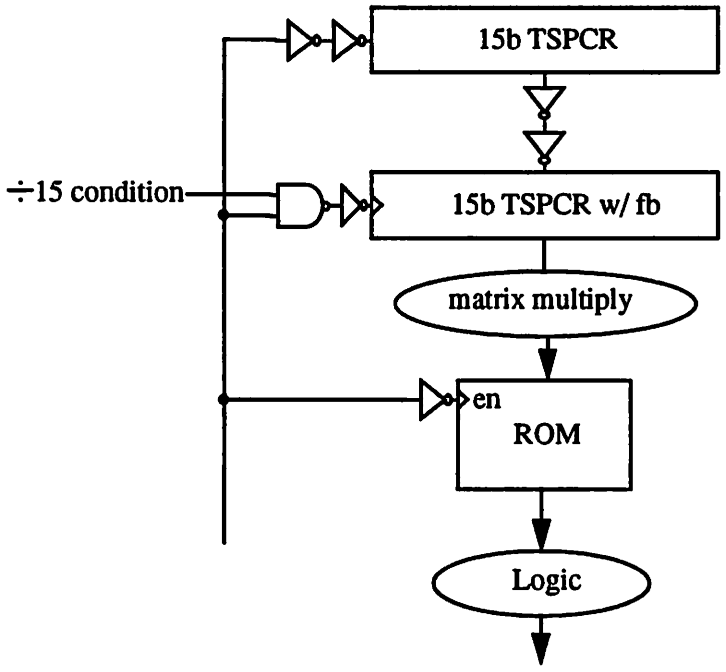
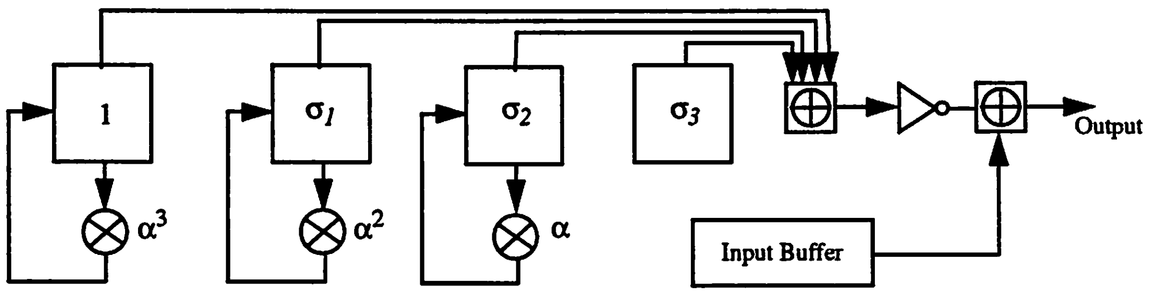


Fig 2. BCH(15, 5, 7) decoder structure viewed in Lager

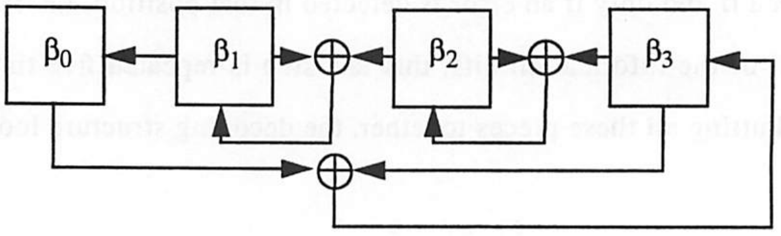
The last block in Fig 1 is probably the most interesting and illustrative unit in terms of showing the key implementation ideas involving finite field algebra. Recall from chapter 2, to determine whether a specific bit has an error, we need to evaluate the error location polynomial f(X) = $X^3 + \sigma_1 X^2 + \sigma_2 X + \sigma_3$ for that particular bit to see if $f(X)|_{\alpha^i} = 0$. For example, assuming there are three or less errors in a block, if the 0th bit has an error, then $f(\alpha^0) = f(1) = 1 + \sigma_1 + \sigma_2 + \sigma_3 = 0$. The result of f(X) is feed into an inverter, and the inverter output is XOR with the received bit, so the received bit is flipped if and only if an error is detected in that position, i.e. $f(X) = 0$. To correct errors in all of the information bits, this last step is repeated five times for each information bit. Putting all these pieces together, the decoding structure looks like:



The final question is how we can implement the finite field polynomial multiplication such as $\sigma_2\alpha$ with hardware. We know that for any field element β in $GF(2^4)$, β can also be written as $\beta = (\beta_0, \beta_1, \beta_2, \beta_3) = \beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3\alpha^3$, given this relationship, $\beta\alpha^3$ can be expressed as:

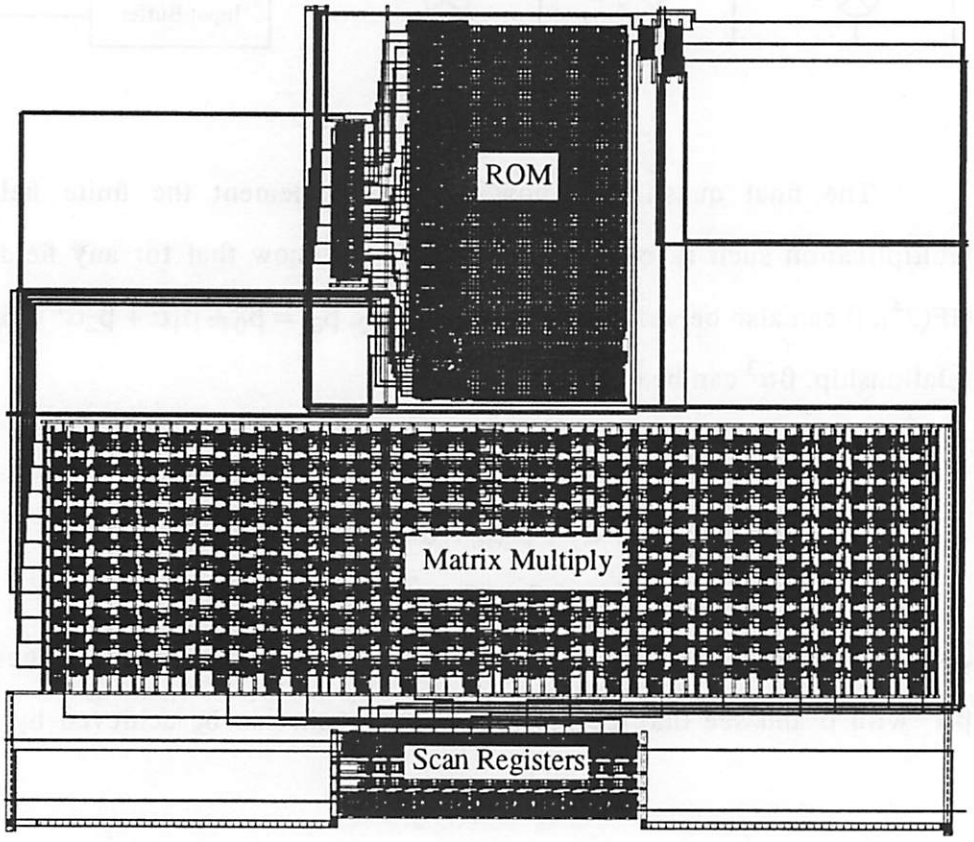
$$\begin{aligned}\beta\alpha^3 &= \beta_0\alpha^3 + \beta_1\alpha^4 + \beta_2\alpha^5 + \beta_3\alpha^6\\ &= \beta_0\alpha^3 + \beta_1(1 + \alpha) + \beta_2(\alpha + \alpha^2) + \beta_3(\alpha^2 + \alpha^3)\\ &= \beta_1 + (\beta_1 + \beta_2)\alpha + (\beta_2 + \beta_3)\alpha^2 + (\beta_0 + \beta_3)\alpha^3\end{aligned}$$

As the result of expansion and recombination, we can make a one to one comparison of $\beta\alpha^3$ with β and see that the multiplications again can be achieved by feedback shift registers! The graphical expression for an actual circuit of $\beta\alpha^3$ is shown in the figure below:



Similarly, the multiplication units for $\beta\alpha^2$ and $\beta\alpha$ can also be done by the same method.

Putting all these pieces together, we have:



The output of this decoder has 5 bits data plus 2 extra error count bits which counts up to 3 errors within a block of 15 bits.

6
Future Work

In this chapter, we would like to present a system view on how error control coding can help improve performance for an indoor wireless communication system, e.g. the Berkeley InfoPad. An implementation of such a system will be investigated as a part of the future research. Fig 1 is a system overview for the radio link between a base-station and a receiver.

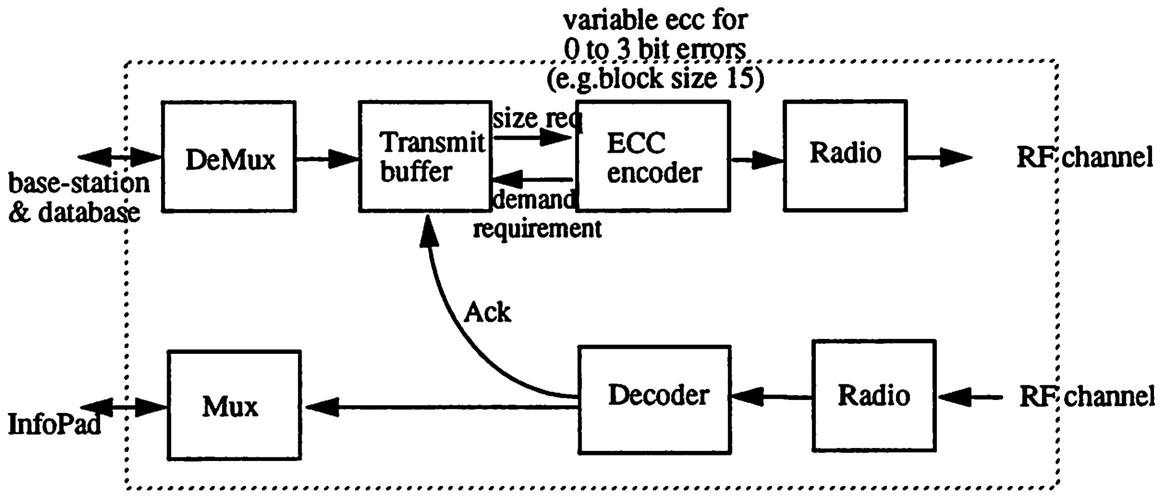


Fig 1 A systematic approach to reliable transmission (by using ECC)

We know that the indoor multipath channel is time variant, so it can be bandwidth inefficient if we use a fixed error correction algorithm with high amount of redundancy such as the BCH(15, 5, 7) code for all the data, especially when the channel does not suffer from significant multipath or multiuser interference. But on the other hand, if we constrain ourselves to a simple Hamming code for the sake of saving bandwidth, reliable transmission can not be achieved for all cases. Therefore, we are proposing a variable error correction module which is adaptive to the quality of the channel in terms of the error rate, and this variable error correction module will be capable to correct 0, 1, 2,.... errors. From the point of view of implementation, the design of such a module can be simplified significantly if we have a fixed block size, e.g. a block size of 15 bits for error correction of 0 to 3 bits. The reason for supporting zero error correction capability as a part of the design is that for data such as video, there is no strict low error rate requirement because human eyes are not very sensitive to video images. Also notice from the figure that some feedback to the transmitter on channel quality is needed to select the appropriate encoding algorithm.

As we have mentioned earlier in chapter 3, the most efficient design in terms of lowering BER is to combine error correction with error detection/retransmission. Fig 2 shows how this can be done for each frame of packet. Ecc level (none, low, medium, high) together with channel quality will determine the actual ecc used by the encoder:

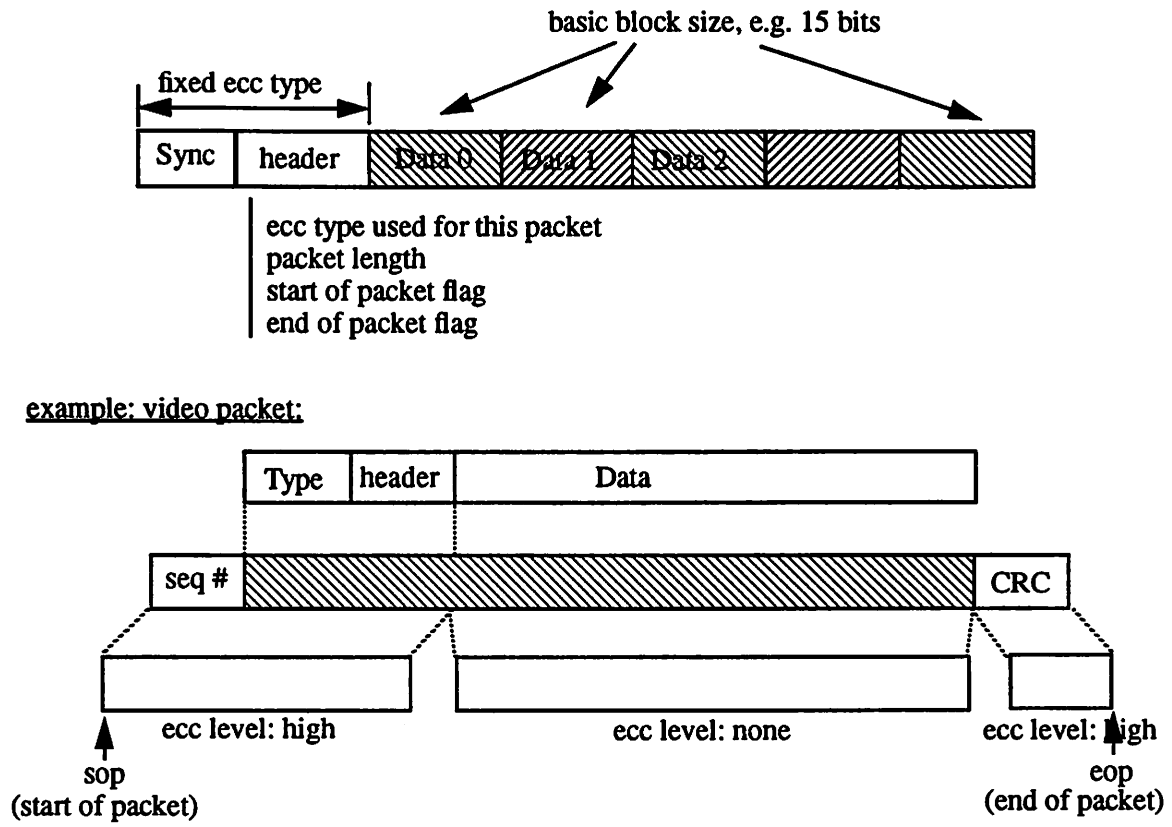


Fig 2. Format of an ecc packet fragment with an example on video data

The contents of a packet are divided into: (1) sync: which is used for overall frame synchronization; (2) header: which contains information on the ecc type used for this packet, packet length, start of packet, and end of packet, etc.; (3) the actual data which is encoded with the ecc level indicated by the header. As we can see, the sync and header are extremely important to us in terms of finding the correct packet position and decoding the packet using the appropriate decoding algorithm; therefore, they should be "heavily" protected by "powerful" error correction algorithm. A fixed CRC (cyclic redundant code) for error detection purpose will be applied to the entire packet to provide additional error checking in case any individual error correction block fails. The result of CRC which contains information such as the number of bit errors during the transmission can be combined with packet acknowledgment for feedback. With the feedback, we can turn up or down the ecc complexity to achieve bandwidth efficiency. Moreover, by having

CRC error detection, retransmission will be possible for a packet with high error-free requirement. In order for retransmission to be possible, a seq # is needed to maintain the overall order for different packets.

Fig 3 shows a variable ecc encoder structure combined with an interleaver.

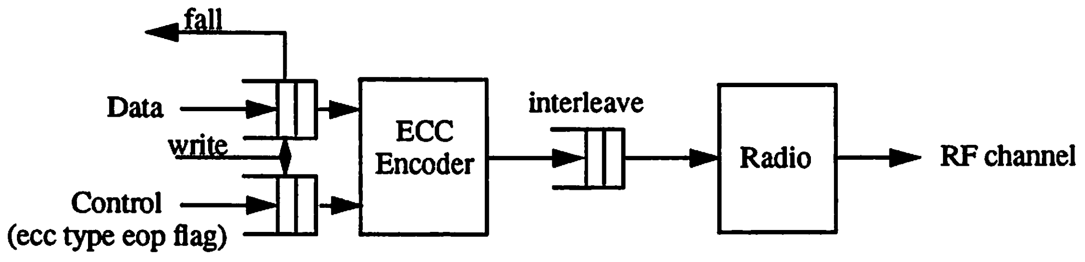


Fig 3. Encoder structure

Figure 4 shows a possible implementation of a decoder at the receiver.

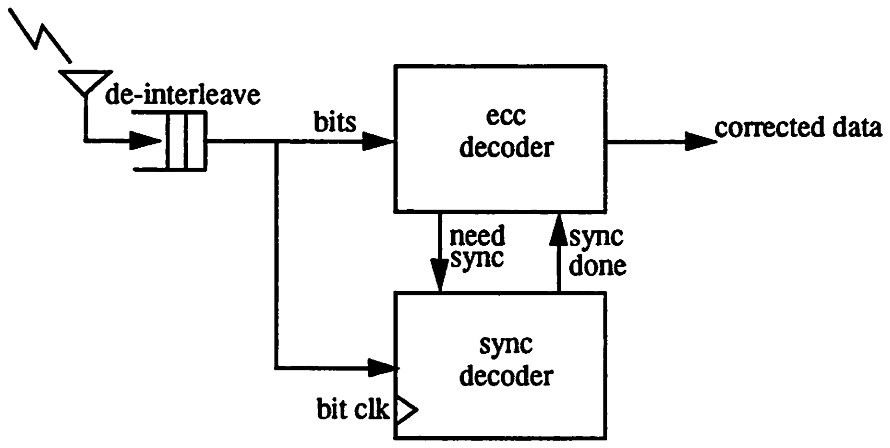


Fig 4. Decoder structure

In order to decode the received vector correctly, precise detection of the beginning of a block is required, or else all the data will be mismatched (unless they are shifted exactly by a block of n bits). As a result, frame sync is a very critical issue in decoding process. In order to have a reliable frame synchronization in an environment with error rate, ecc has to be used for frame synchronization. Fig 5 is a design for a synchronization decoder. First, the decoder has to de-interleave the data to make errors look random. Then the data bits will go through the synchronization decoder for detecting the beginning the packet. After synchronization pattern is detected, a “sync done” signal is sent to the ecc decoder to start decoding the actual data. After transmitting each packet, or multiple packets, the ecc decoder will issue a “need sync” to the synchronization decoder asking for re-synchronization. Thus, an overall system synchronization can be obtained.

Fig 5 is a detailed design for a synchronization decoder.

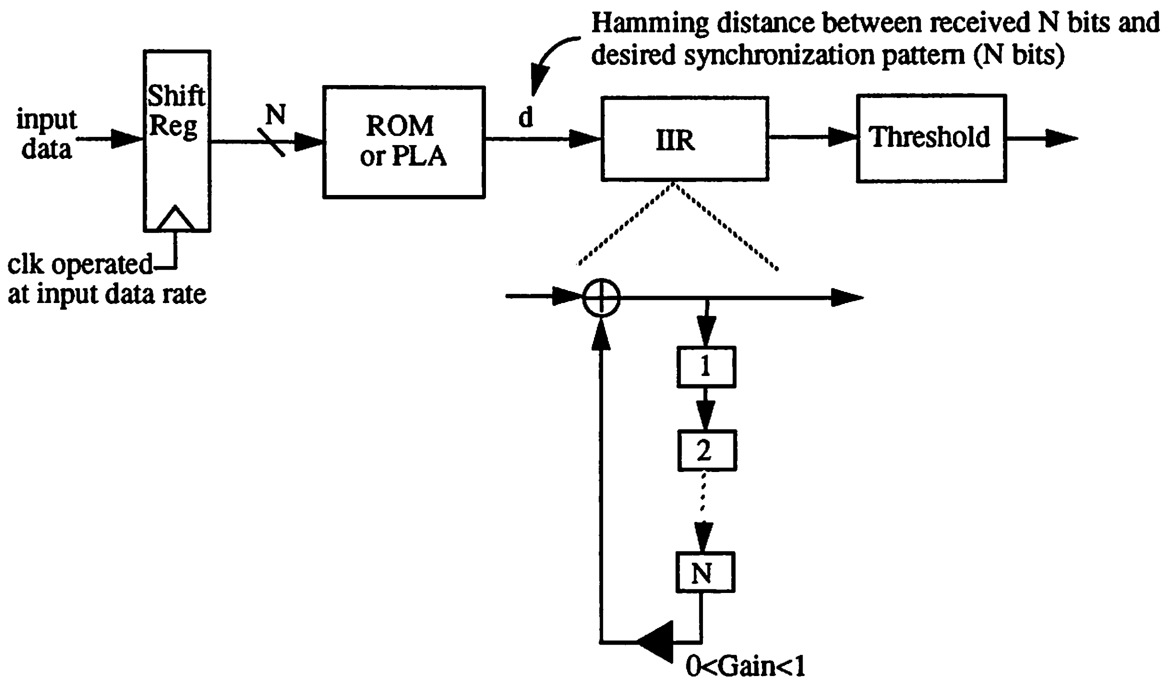

Hamming distance between received N bits and desired synchronization pattern (N bits)
input data
Shift Reg
N
ROM or PLA
d
IIR
Threshold
clk operated at input data rate
1
2
N
0<Gain<1

Fig 5. Sync decoder structure

Frame synchronization can be done by looking for a known pattern of N bits which is sent repeatedly for m times. The decoder starts decoding at the beginning of the received data. For each block of N bits, the Hamming distance d of the received bits vs. synchronization pattern is calculated and stored in the registers of an IIR filter with N registers. After each bit, the ecc decoder will shift by one bit and calculate the distance again, iterate this step until m x N bits are processed. The threshold filter will then choose the register with the minimum distance, and call the position of that register the beginning of the data. This frame synchronization detection algorithm is actually the maximum likelihood algorithm, thus it is optimal in the sense of minimizing the probability of making a wrong decision with the given observations.

This proposal on system design of reliable transmission for indoor wireless communication will be looked more carefully as a part of the future research. We will consider implementing this for Berkeley InfoPad project.

Reference

W. Jakes, *Microwave Mobile Telecommunications.* New York: Wiley, 1974.

J. P. M. G. Linnartz, *Land Mobile Radio Networks.* Norwood, MA: Artech House, 1993.

J. Proakis, *Digital Communications, 2nd ed.* New York: McGraw-Hill, 1989.

S. Lin and D. Costello, *Error Control Coding.* Englewood Cliffs, NJ: Prentice-Hall, 1983.

A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun.*, Oct. 1971, pp. 751-772.

D. Falconer and F. Magee, "Adaptive Channel Memory Truncation for Maximum-Likelihood Sequence Estimation," *Bell System Technical Journal*, vol. 52, Nov. 1973, pp. 1541-1562.

M. Eyuboglu and S. Qureshi, "RSSE with Set Partitioning and Decision Feedback," *IEEE Trans. Commun.*, Jan. 1988, pp. 13-20.

Ziemer and Tranter, *Principles of Communications.* Boston: Houghton-Mifflin, 1985.

V. Pless, *Introduction to the Theory of Error-Correcting Codes.* New York: Wiley, 1989.

J. Odenwalder, "Error Control Coding," Linkabit Technical Report, 1976.

W. W. Peterson, "Error Correcting Codes", *The MIT Press*, 1972

Lager Tool Menu, UC Berkeley, 1991