



BCH code

Presented for:

Dr.Haidar Al Mokdad

Presented by:

Mohammad Haidar 3967

Maher ayoub

Summary:

|  |  |
| --- | --- |
| 1.Introduction |  |
| 2.Coding theory |  |
| 3.BCH code |  |
| 4.MATLAB implementation |  |
| 5.Conclusion |  |

1.Introduction

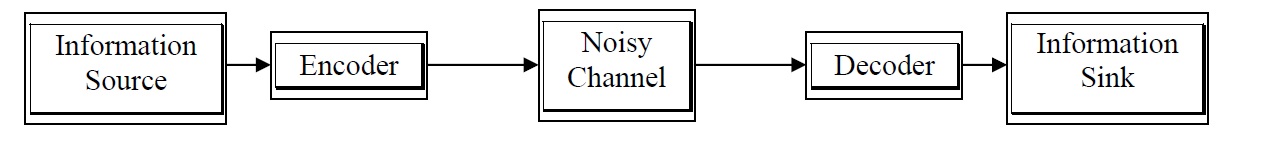
In coding theory, the BCH codes form a class of cyclic error-correcting codes that are constructed using finite fields. BCH codes were invented in 1959 by French mathematician Alexis Hocquenghem, and independently in 1960 by Raj Bose and D. K. Ray-Chaudhuri. The acronym BCH comprises the initials of these inventors' names.

One of the key features of BCH codes is that during code design, there is a precise control over the number of symbol errors correctable by the code. In particular, it is possible to design binary BCH codes that can correct multiple bit errors. Another advantage of BCH codes is the ease with which they can be decoded, namely, via an algebraic method known as syndrome decoding. This simplifies the design of the decoder for these codes, using small low-power electronic hardware.

BCH codes are used in applications such as satellite communications, compact disc players, DVDs, disk drives, solid-state drives and two-dimensional bar codes.

The Basic Structure of a Communications System:

The basic structure of a communications system is presented in this diagram.



The information can of course be anything representable in symbols. For our purposes, this information should be representable as a string of binary digits. The purpose of the communications system is to convey the information from one point to another with no degradation.

The noisy channel adds noise to the information without our consent, corrupting the information to a degree related to the character and strength of the noise. The detailed behavior of communications systems in the presence of noise is a lengthy study in itself, and will not be pursued further in this short paper, except to state the limits of our coding methods.

We are concerned mostly with the encoder and decoder blocks. These blocks implement the “coding system” spoken of by Shannon, adding some extra bits in the encoder, and removing them again in the decoder. The decoder detects errors, corrects errors, or a combination of both. If we have a good design, the information on the input and output will be identical.

Not surprisingly, there are numerous coding systems. Each has a suitable area of service. This investigation targets only the most basic of coding systems, so our assumptions must match. We are assuming that we are transmitting symbols of only binary bits; that the channel adds a purely randomly determined amount of noise to each bit transmitted; and that each bit is independent of the others (the channel is memoryless). This is called a binary symmetric channel. Lin and Costello have an excellent introduction to this concept .

We will develop a random error correcting code here called the BCH code. This code handles randomly located errors in a data stream according to its inherent limitations. Other coding systems are optimized to correct bursts of errors, such as the Reed-Solomon code used in compact discs. Errors induced in a compact disc are more likely to damage a bunch of bits together, as a scratch or finger smudge would do.

On top of these coding systems, other techniques are used to ensure data accuracy. These are only mentioned here, as they are full studies alone. Typically, if data errors are detected on a message in a network, the receiving station requests a retransmission from the sender. This is called automatic repeat request, or ARQ. Even if error correction codes are used, usually another detection-only code is used on data messages to check the results of the correction. If that test fails, a retransmission is requested. On top of that, systems may use a combination of burst and random

error correcting codes, gaining benefits from each method at the cost of added complexity. From this brief overview, the reader can see that we are only touching the surface.

2.Coding theory

Coding theory is the study of the properties of codes and their fitness for a specific application. Codes are used for data compression, cryptography, error-correction and more recently also for network coding. Codes are studied by various scientific disciplines—such as information theory, electrical engineering, mathematics, and computer science—for the purpose of designing efficient and reliable data transmission methods. This typically involves the removal of redundancy and the correction (or detection) of errors in the transmitted data.

There are essentially two aspects to coding theory:

Data compression (or, source coding)

Error correction (or channel coding)

These two aspects may be studied in combination. Source encoding attempts to compress the data from a source in order to transmit it more efficiently. This practice is found every day on the Internet where the common Zip data compression is used to reduce the network load and make files smaller. The second, channel encoding, adds extra data bits to make the transmission of data more robust to disturbances present on the transmission channel. The ordinary user may not be aware of many applications using channel coding. A typical music CD uses the Reed-Solomon code to correct for scratches and dust. In this application the transmission channel is the CD itself. Cell phones also use coding techniques to correct for the fading and noise of high frequency radio transmission. Data modems, telephone transmissions, and NASA all employ channel coding techniques to get the bits through, for example the turbo code and LDPC codes.

Types of Codes

We are looking at the BCH error detection and correction code, going first through some preliminary background on the mathematical basis of the code.

Now that we have a field in which to do computations, what we need next is a philosophy for encoding and decoding data in order to detect and possibly correct errors. Here we are taking advantage of results discovered by the hard work of perhaps a hundred individuals just in the last 50 years.

There are several types of codes. The first major classification is linear vs. nonlinear. Linear codes in which we are interested may be encoded using the methods of linear algebra and polynomial arithmetic. Then we have block codes vs. convolutional codes. Convolutional codes operate on streams of data bits continuously, inserting redundant bits used to detect and correct errors.

Our area of investigation here will be linear block codes. Block codes differ from convolutional codes in that the data is encoded in discrete blocks, not continuously. The basic idea is to break our information into chunks, appending redundant check bits to each block, these bits being used to detect and correct errors. Each data + check bits block is called a codeword. A code is linear when each codeword is a linear combination of one or more other codewords. This is a concept from linear algebra and often the codewords are referred to as vectors for that reason.

Another characteristic of some block codes is a cyclic nature. That means any cyclic shift of a codeword is also a codeword. So linear, cyclic, block code codewords can be added to each other and shifted circularly in any way, and the result is still a codeword. You might expect that it takes some finesse to design a set of binary words to have these properties.

Since the sets of codewords may be considered a vector space, and also may be generated through polynomial division (the shifting algorithm, above), there are two methods of performing computations: linear algebra and polynomial arithmetic.

Linear Block Codes

The first code developed was the Hamming code, in 1950. It actually consists of a whole class of codes with the following characteristics :

Block Length: n = 2m - 1 Information Bits: k = 2m - m - 1 Parity Check Bits: n - k = m Correctable Errors: t = 1

These conditions are true for m > 2. For example, with m = 4, there are n = 15 total bits per block or codeword, k = 11 information bits, n - k = 4 parity check bits, and the code can correct t = 1 error. A representative codeword would be

10010100101 0010

where the four bits on the right (0010) are the parity checkbits. By choosing the value of m, we can create a single error correcting code that fits our block length and correction requirements. This one is customarily denoted a (15, 4) code, telling us the total number of bits in a codeword (15) and the number of information bits (4).

We omit the details of encoding and decoding the Hamming code here because such will be covered in detail for the BCH code, later.

The Golay code is another code, more powerful than the Hamming code, and geometrically interesting. This (23, 12) code was discovered by Marcel J. E. Golay in 1949 4. It may also be extended using an overall parity bit to make a (24, 12) code. The minimum distance is seven, so it can detect up to six errors, or correct up to t = (7 - 1)/2 = 3 errors.

The aspect of the Golay and Hamming codes that makes them interesting is the fact that they are perfect. With any code, the codewords can be considered to reside within spheres packed into a region of space. The entire space is GF(2m). Each sphere contains a valid codeword at its center and also all the invalid codewords that correct to the valid codeword, those being a distance of three or fewer bits from the center in the case of the Golay code (t = 3). If there are orphan binary words outside the spheres, then the code is termed imperfect.

3.BCH code

The BCH abbreviation stands for the discoverers, Bose and Chaudhuri (1960), and independently Hocquenghem (1959). These codes are multiple error correcting codes and a generalization of the Hamming codes. These are the possible BCH codes for m>=3 and t < 2^m-1 :

Block Length: n = 2^m - 1

Parity Check Bits: n – k<=mt

Minimum distance: d=>2t + 1

The codewords are formed by taking the remainder after dividing a polynomial representing our information bits by a generator polynomial. The generator polynomial is selected to give the code its characteristics. All codewords are multiples of the generator polynomial.

Let us turn to the construction of a generator polynomial. It is not simply a minimal, primitive polynomial as in our example where we built GF(16). It is actually a combination of several polynomials corresponding to several powers of a primitive element in GF(2m).

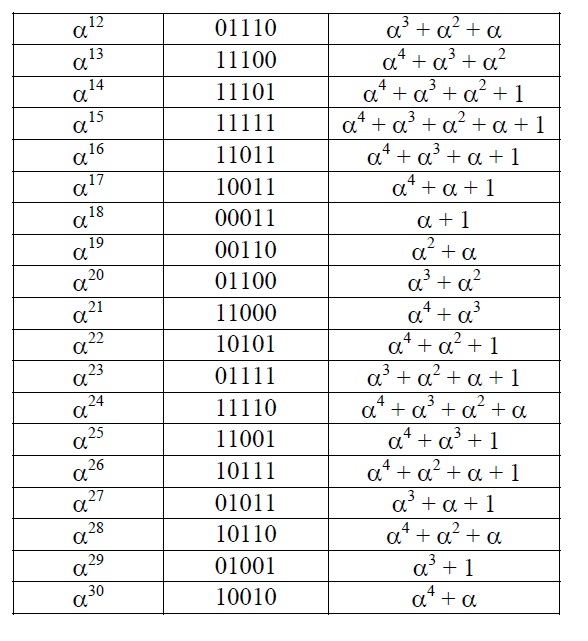
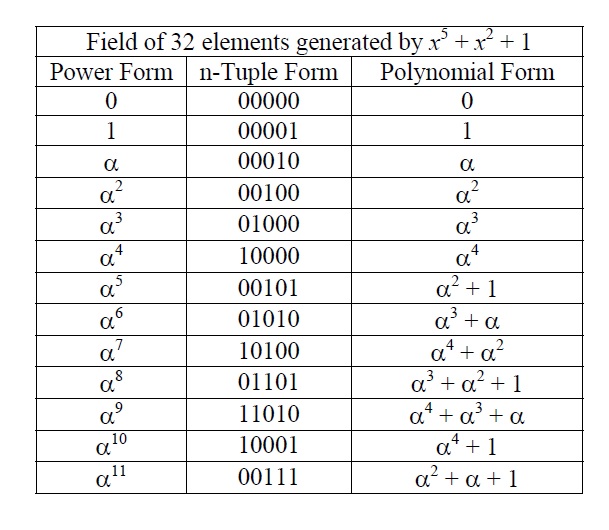
The discoverers of the BCH codes determined that if is a primitive element of GF(2m), the generator polynomial is the polynomial of lowest degree over GF(2) with alpha,alpha^2 ,alpha^3 ,...,alpha^2t as roots. The length of a codeword is 2m - 1 and t is the number of correctable errors. Lin concludes8 that the generator is the least common multiple of the minimal polynomials of each i term. A simplification is possible because every even power of a primitive element has the same minimal polynomial as some odd power of the element, halving the number of factors in the polynomial. Then g(x) = lcm(m1(x), m3(x), , m2t-1(x)).

These BCH codes are called primitive because they are built using a primitive element of GF(2m). BCH codes can be built using nonprimitive elements, too, but the block length is typically less than 2m - 1.

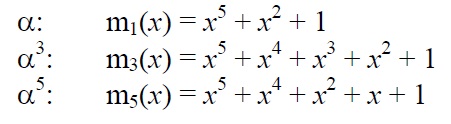
As an example, let us construct a generator polynomial for BCH(31,16). Such a codeword structure would be useful in simple remote control applications where the information transmitted consists of a device identification number and a few control bits, such as “open door” or “start ignition.”

This code has 31 codeword bits, 15 check bits, corrects three errors (t = 3), and has a minimum distance between codewords of 7 bits or more. Therefore, at first glance we need 2t - 1 = 5 minimal polynomials of the first five powers of a primitive element in GF(32). But the even powers’ minimal polynomials are duplicates of odd powers’ minimal polynomials, so we only use the first three minimal polynomials corresponding to odd powers of the primitive element.

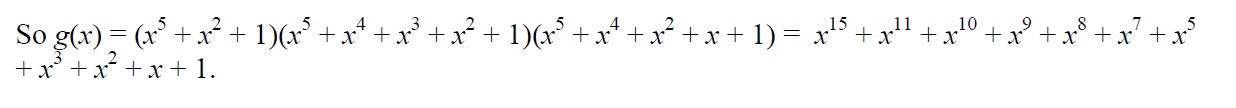
The field we are working in is GF(32), shown below. This was generated using primitive polynomial x5 + x2 + 1 over GF(32).



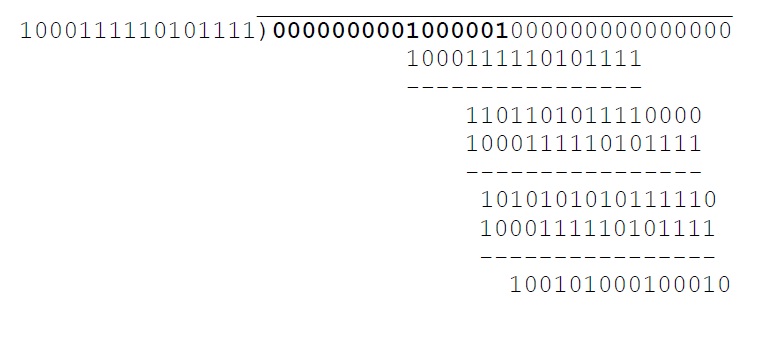
We need first a primitive element. Well, is a primitive element in GF(32). Next we need the minimal polynomials of the first three odd powers of alpha. Tables of minimal polynomials appear in most texts on error control coding. Lin and Costello, Pless, and Rorabaugh exhibit algorithms for finding them using cyclotomic cosets. From Lin and Costello, the first three odd power of alpha minimal polynomials are:



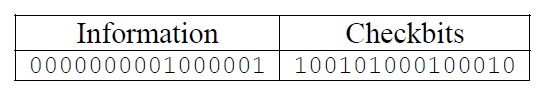
Therefore, g(x) = lcm(m1(x), m3(x), m5(x)) = m1(x) m3(x) m5(x) (since these are irreducible).



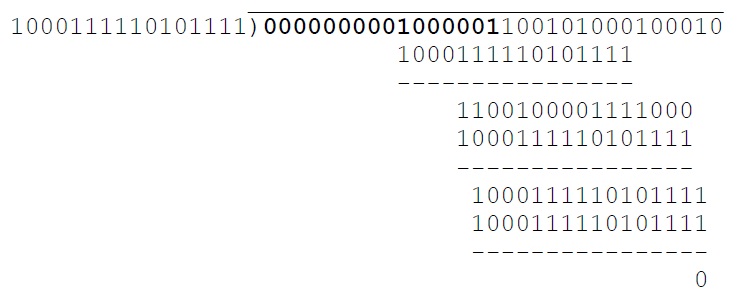
To encode a block of bits, let us first select as our information the binary word 1000001 for the letter “A” and call it f (x), placing it in the 16-bit information field . Next, we append a number of zeros equal to the degree of the generator polynomial (fifteen in this case). This is the same as multiplying f (x) by x15. Then we divide by the generator polynomial using binary arithmetic (information bits are bold):



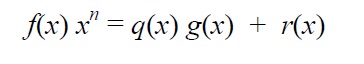
The quotient is not used and so we do not even write it down. The remainder is 100101000100010, or x^14 + x^11 + x^9 + x^5+ x in polynomial form, and of course it has degree less than our generator polynomial, g(x). Thus the completed codeword is



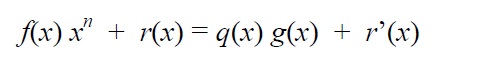
This method is called systematic encoding because the information and check bits are arranged together so that they can be recognized in the resulting codeword. Nonsystematic encoding scrambles the positions of the information and check bits. The effectiveness of each type of code is the same; the relative positions of the bits are of no matter as long as the encoder and decoder agree on those positions11 . To test the codeword for errors, we divide it by the generator polynomial:



The remainder is zero if there are no errors. This makes sense because we computed the checkbits (r(x)) from the information bits (f (x)) in the following way:



The operation f (x)x^n merely shifts f (x) left n places. Concatenating the information bits f (x) with the checkbits r(x) and dividing by g(x) again results in a remainder, r’(x), of zero as expected because



If there are errors in the received codeword, the remainder, r’(x), is nonzero, assuming that the errors have not transformed the received codeword into another valid codeword. The remainder is called the syndrome and is used in further algorithms to actually locate the errant bits and correct them, but that is not a trivial matter.

The BCH codes are also cyclic, and that means that any cyclic shift of our example codeword is also a valid codeword. For example, we could interchange the information and checkbits fields in the last division above (a cyclic shift of 15 bits) and the remainder would still be zero.

Decoding the BCH(31,16) Code:

Determining where the errors are in a received codeword is a rather complicated process. (The concepts here are from the explanation of Lin and Costello.) Decoding involves three steps:

1.Compute the syndrome from the received codeword.

2.Find the error location polynomial from a set of equations derived from the syndrome.

3.Use the error location polynomial to identify errant bits and correct them.

We have seen that computing the syndrome is not difficult. However, with the BCH codes, to implement error correction we must compute several components which together comprise a syndrome vector. For a t error correcting code, there

are 2t components in the vector, or six for our triple error correcting code. These are each formed easily using polynomial division, as above, however the divisor is the minimal polynomial of each successive power of the generating element,

.

Let v(x) be our received codeword. Then Si = v(x) mod mi(x), where mi(x) is the minimal polynomial of i. In our example,

S 1(x) = v(x) mod m1(x)

S2(x) = v(x) mod m2(x)

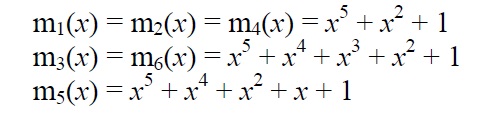
S3(x) = v(x) mod m3(x)

S4(x) = v(x) mod m4(x)

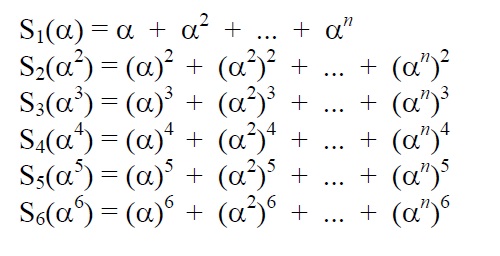
S5(x) = v(x) mod m5(x)

S6(x) = v(x) mod m6(x)

Now in selecting the minimal polynomials, we take advantage of that property of field elements whereby several powers of the generating element have the same minimal polynomial. If f (x) is a polynomial over GF(2) and is an element of GF(2m), then if b = 2^i, alpha^b is also a root of f (x) for i>0^13. These are called conjugate elements. From this we see that all powers of such as alpha^2 , alpha^4 , alpha^8,alpha^16,… are roots of the minimal polynomial of . In GF(32) which applies to our example, we must find the minimal polynomials for through 6. The six minimal polynomials are:



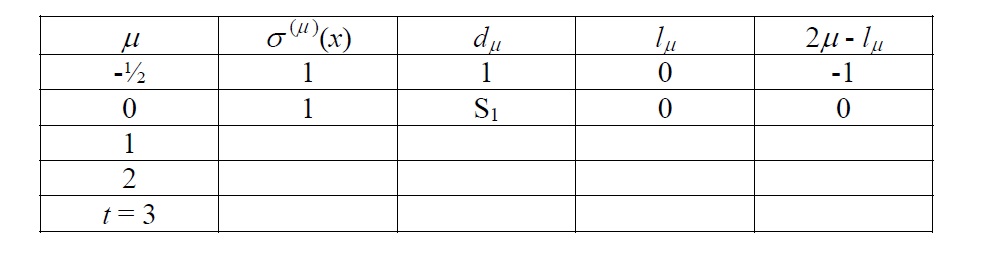
Next, we form a system of equations in alpha:

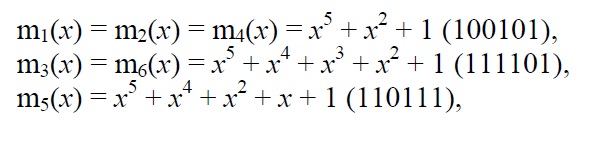
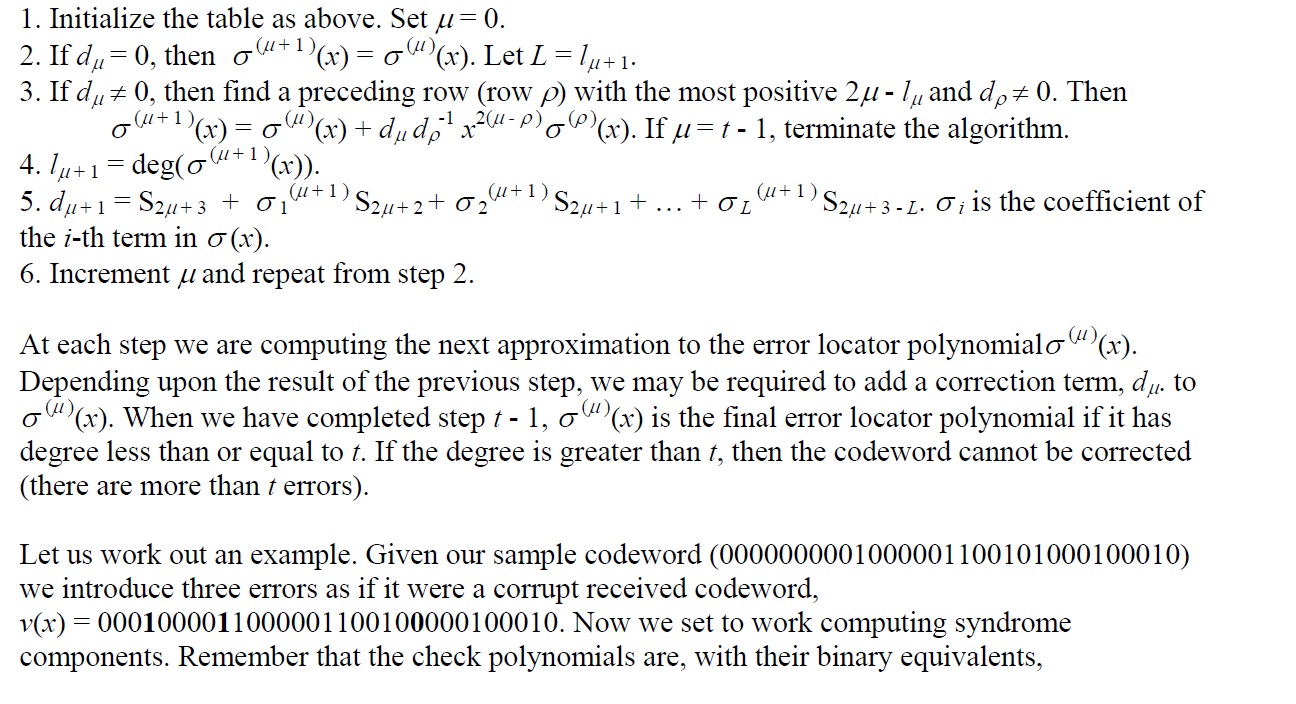


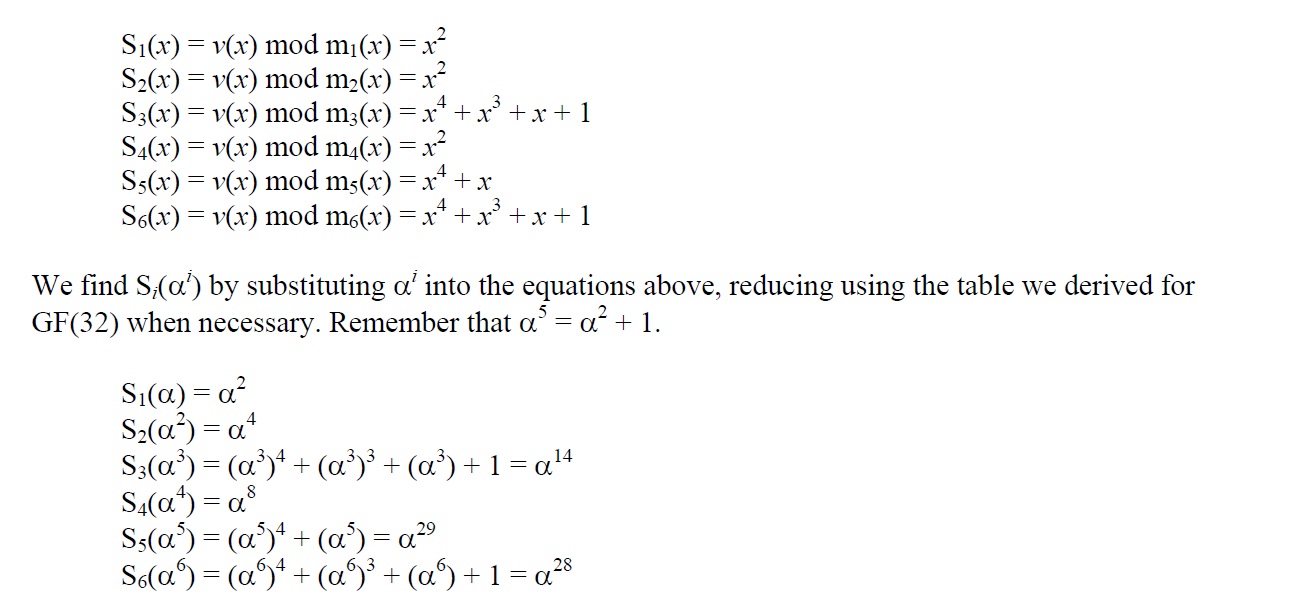
It turns out that each syndrome equation is a function only of the errors in the received codeword. The alpha^i are the unknowns, and a solution to these equations yields information we use to construct an error locator polynomial. One can see that this system is underconstrained, there being multiple solutions. The one we are looking for is the one that indicates the minimum number of errors in the received codeword (we are being optimistic).

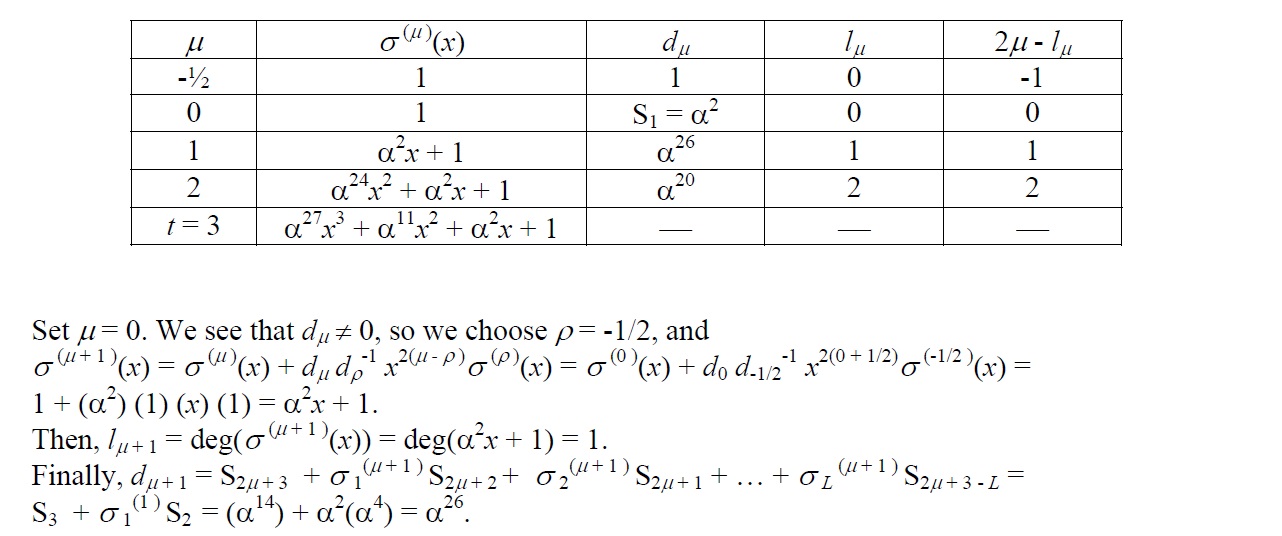
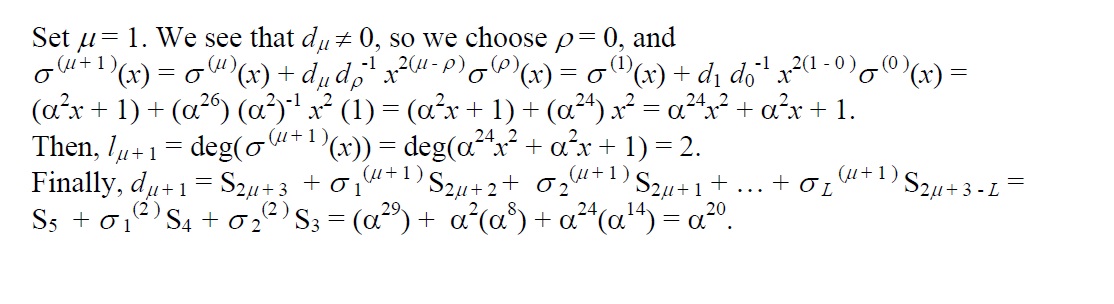
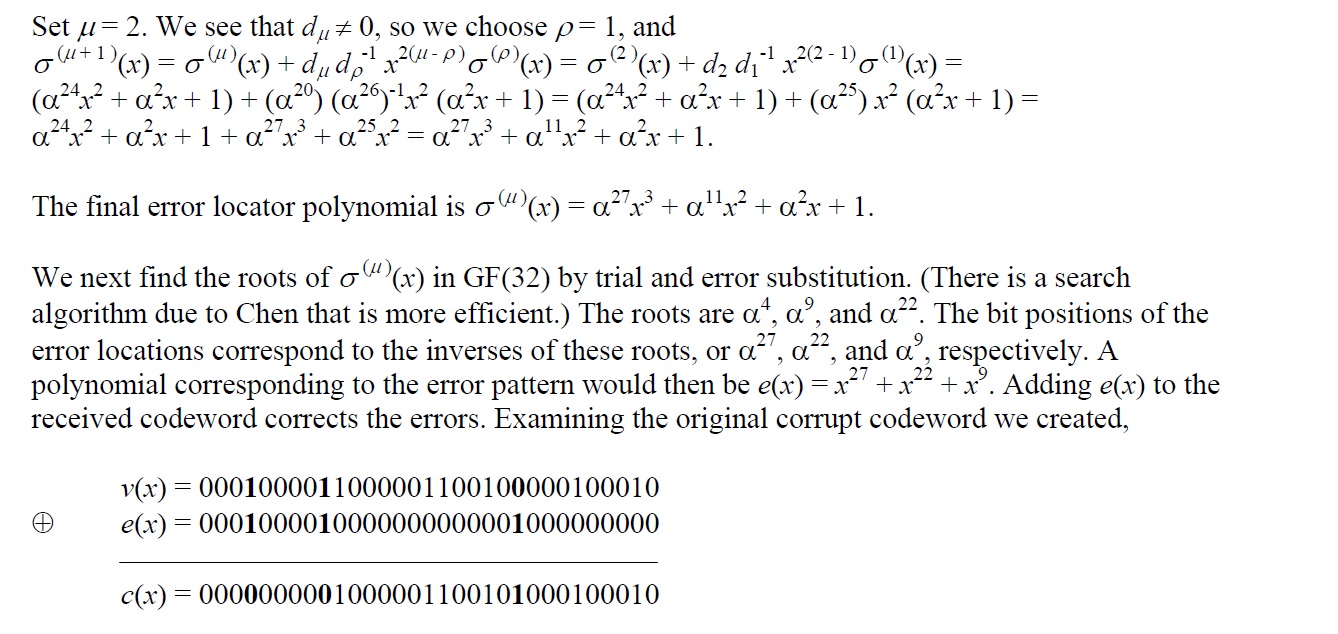
The method of solution of this system involves elementary symmetric functions and Newton's identities and is beyond the scope of this paper. However, this method has been reduced to an algorithm by Berlekamp that builds the error locator polynomial iteratively. Using the notation of Lin and Costello, a t + 2 line table may be use to handle the bookkeeping details of the error correction procedure for binary BCH decoding. It is described next.

First, make a table (using BCH(31,16) as our example):





so we have three divisions to do to find six syndrome components. These are

Using the algorithm, we fill in the table.

and it is clear that the calculated error pattern matches the actual error pattern and c(x) matches our original codeword.

If there are no errors, then the syndromes all work out to zero. One to three errors produce the corresponding number of bits in e(x). More than three errors typically results in an error locator polynomial of degree greater than t = 3. However, it is again possible that seven bit errors could occur, resulting in a zero syndrome and a false conclusion that the message is correct. That is why most error correction systems take other steps to ensure data integrity, such as using an overall check code on the entire sequence of codewords comprising a message.

In practice, error correction is done in either software or hardware. The Berlekamp algorithm is complex and not too attractive when considered for high-speed communications systems, or operation on power limited microprocessors. One version of the BCH code is used in pagers and another in cell phones, so optimization of the algorithm for the application is important. A cursory scan of the literature shows efforts are being made to discover alternative methods of decoding BCH codes.

4.MATLAB implementation

We implemented the following cone for BCH decoder using matlab

function code = BCHencode (n,k,msg)

% the degree m of GF(2^m)

m = ceil(log2(n+1));

% nb of parity bits

p = n-k;

% nb of detectable errors

t = ceil(p/m);

% minimun distance

d = 2\*t+1;

% initializing g(x)

pol=1;

% findind minimal 2t-1 polynomials and their lcm

for (i=1:t)

phi= fliplr(gfminpol(2\*i-1,m));

pol=gfconv(pol,phi);

end

% encoding the msg

inf = [zeros([1 k-length(msg)]) msg];

block = [inf zeros([1 length(pol)-1])];

[~,r] = gfdeconv(fliplr(block),fliplr(pol));

code = [inf fliplr(r)];

poly2sym(pol)

This function gets as input the message and the parameters for BCH system (block length and number of correction bits) and give the coded message as output .

5.Conclusion

* BCH is a multilevel cyclic variable-length error detection coding.
* It is an efficient code to detect and correct errors.
* It is easy to decode using the syndrome decoding method.
* It can be implemented using a simple electronic hardware.