Implementation of BCH encoder and decoder

Akshit Kumar, *EE14B127* & Sachin Agrawal, *EE14B104*

***Abstract*—In this project we design a narrow sense BCH code of length $n = 127$ with design distance $\delta = 15$. We implement a systematic encoder for the code. We also implement a decoder using the Berlekamp-Massey decoding algorithm. In case the decoder fails, it gives a reason for the same. The decoder can correct errors as well as erasures within certain limits.**

***Keywords*—Binary codes, narrow sense, BCH code, Berlekamp Massey, Chien's Search**

I. Introduction

For any positive integers $m(m \geq 3)$ and $t(t < 2^{m-1})$, there exists a BCH code having the following parameters:

Block length : $n = 2^m - 1$
Number of parity-check digits: $n - k \leq mt$
Minimum Distance : $d_{min} \geq 2t + 1$

In this project we are given $n = 127$ and $d_{min} = 15$, on solving the above parameters,we get $k = 78$, $t = 7$ and $m = 7$.

II. Systematic Encoding

A. BCH Encoding

BCH codes can be encoded in the same fashion as any other cyclic code. Given a message vector $\mathbf{m} = (m_0, m_1, m_2, ..., m_{k-1})$, we can write the message polynomial as $m(x) = m_0 + m_1 x + m_2 x^2 + ... + m_{k-1} x^{k-1}$ where $m_i \in GF(2)$. The generator polynomial $g(x)$ of the $t-$error-correcting BCH code is the *lowest-degree-polynomial* over $GF(2)$ which has $\alpha, \alpha^2, ..., \alpha^{2t}$ as its roots.Therefore $g(x)$ is given as

$$g(x) = LCM\{\phi_1(x), \phi_2(x), ..., \phi_{2t}(x)\}$$

The systematic encoding gives the codeword polynomial as

$$c(x) = x^{n-k} m(x) + x^{n-k} m(x) \% g(x)$$

B. Implementation Details

The messages to be encoded are read from a file named "msgs.txt" in the binary form (string of 0's and 1's). The read messages are encoded in the systematic form and written out on to a file named "codeword.txt".
Multiplication of polynomials in $GF(2)$ is done by convolving the coefficients of the two polynomials. The generator polynomial $g(x)$ is obtained by the repeated multiplication of $\phi_i(x)$ for $i \in \{0, 1, ..., 2t\}$. Division of polynomials in $GF(2)$ is done by deconvolving the coefficients of the two polynomials. MATLAB returns the quotient and remainder. We use this to calculate the remainder of division operation between $x^{n-k} m(x)$ and $g(x)$. The remainder bits act as the parity bits and are appended to the message bits to obtain the encoded codeword.

III. Decoding

Decoding follows the following steps:

- Syndrome computation from the received vector
- Using the generated syndrome to compute the error locator polynomial using the simplified Berlekamp Massey algorithm as given in Shu-Lin and Costello.
- Finding the roots of the error locator polynomial using Chien's search or brute force.
- Find the multiplicative inverse of the roots to find the error location.
- Add the error received to the received vector.

The decoding algorithm fails if the number of errors is more than t. That is if the degree of the error locator polynomial is more than t then the algorithm does not decode. Another place where the decoder fails is when the error locator polynomial has repeated roots. In all other cases the decoder gives an estimate of the transmitted codeword.

The decoder also decodes correctly for up to $2t$ erasures. The decoding for erasures follows the following steps:

- Replace all the erasures with 1's and compute the error estimate say $\hat{e_1}$.
- Replace all erasures with 0's and compute the error estimate say $\hat{e_0}$
- If $nnz(\hat{e_1}) < nnz(\hat{e_0})$ then decode using the received vector with all erasures replaced by 1's. Otherwise decode using the received vector with all erasures replaced by 0's

A. Syndrome computation

For decoding a t-error-correcting binary BCH Code, the syndrome is a 2t-tuple given as $\mathbf{S} = (S_1, S_2, S_3, ..., S_{2t})$. For $1 \leq i \leq 2t$, $S_i = \mathbf{r}(\alpha^i)$. We can also compute S_i as $S_i = \mathbf{r}(\alpha^i) = \mathbf{b}_i(\alpha^i)$ where $b_i(x)$ is the remainder with degree less than that of $\phi_i(x)$.

B. Simplified Berlekamp Massey Algorithm

The Berlekamp-Massey algorithm is an iterative algorithm for finding the best fit for a set of equations involving an Linear Finite Shift Register relationship with a quadratic computational complexity. The algorithm is presented as follows :

C. Chien's Search

Chien's Search performs an exhaustive search for roots in the field $GF(128)$ by evaluating the error-locator polynomial for all values of the field and taking the values which evaluate to 0 as the roots of the error locator polynomial.

Algorithm 1 Berlekamp Massey Algorithm

function BERLEKAMP($syndrome$)
  $\mu = \{\frac{-1}{2}, 0, 1, 2, ..., t\}$
  Initialize $\sigma(x)$, d_μ, l_μ
  for $\mu = 1$ to t **do**
    if $d_\mu = 0$ **then**
      $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x)$
    else
      Find another row preceeding the μ^{th} row, say the ρ^{th} row, such that the number $2\rho - l_\rho$ is maximum and $d_\rho \neq 0$
      $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x)$
    end if
    $d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} +$ ..... $+ \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}}$
  end for
end function

D. Finding the Error Location Polynomial

After finding the roots of the error locator polynomial, we find the inverse of the roots to find the error location polynomial. The inverse of the roots is given as

$$(\alpha^i)^{-1} = \alpha^{p^m - 1 - i}$$

.The inverse of the roots gives the error-location numbers. The error-location polynomial is given as

$$e(x) = \Sigma_{i \in I} x^i$$

where I is the set of degrees of inverse of roots of the error-locator polynomial.

E. Decoding the Code

The decoding of the code is completed by adding the (modulo-2) error-location polynomial $e(x)$ and the received vector polynomial $r(x)$.

IV. Decoding Failure

There are 2 cases for the decoder to give a decoding failure:

- The decoder will fail if there are some erasures along with the errors such that $2w + e > t$ where e is the number of erasures and w is the number of errors. In which case the reason is ***too many errors and erasures for the decoder to handle***. In this case the decoded bit values are flagged by -1.
- While performing Chien's search, we assume that the number of roots of the error locator polynomial is same as the degree of the error locator polynomial. So in case the error locator polynomial has a degree k while Chien's Search returns less than k roots, then the decoder is unable to find the location of some or all it's roots. In which case, the reason is ***inability to locate the position of all the errors***.In this case the decoded bit values are flagged by -2.

V. Decoding error

A decoding error occurs if the received vector ends up in the Hamming ball of radius 7 of another codeword(different from the transmitted one). This can occur only if the number of errors is more than t. In this case there is no way to flag erroneous transmission.

VI. Sample outputs

In this section the stepwise output for the Berlekamp-Massey algorithm is specified.The Berlekamp Massey codes writes such an output to 'tableofsteps.txt' and 'tableofsteps.csv'. The table mentioned here is for the first codeword mentioned on the last page of the report. The received codeword has 3 errors. The syndrome for this code vector is [31 62 7 124 82 14 3 121 44 37 34 28 91 6]

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$\frac{-1}{2}$	1	0	0	-1
0	1	103	0	0
1	1+$\alpha^{103}X$	125	1	1
2	1+$\alpha^{103}X + \alpha^{22}X^2$	25	2	2
3	1+$\alpha^{103}X + \alpha^{104}X^2 + \alpha^3X^3$	-1	3	3
4	1+$\alpha^{103}X + \alpha^{104}X^2 + \alpha^3X^3$	-1	3	5
5	1+$\alpha^{103}X + \alpha^{104}X^2 + \alpha^3X^3$	-1	3	7
6	1+$\alpha^{103}X + \alpha^{104}X^2 + \alpha^3X^3$	-1	3	9
7	1+$\alpha^{103}X + \alpha^{104}X^2 + \alpha^3X^3$	-1	3	11

- The first case where the number of errors is less than (or equal to) t in the absence of erasures, the algorithm decodes correctly to the sent message vector.
- The second case where the number of error is more than t (number of errors taken is 8), the decoder can give a failure or decode to an incorrect codeword depending on the number of errors. In this case we get a decoding failure.
- The third case deals only with erasures and no errors. The algorithm can correct for up to $2t$ erasures. In this case 6 erasures are taken and it is demonstrated that the algorithm decodes correctly.
- In the fourth case we take more than $2t$ erasures, to demonstrate that we may get a decoder failure. 15 erasures are added to demonstrate this
- In the fifth case a code with w errors and t erasures is taken such that $2w + e < 2t$, to demonstrate that the decoder works correctly
- In the sixth case a code with w errors and t erasures is taken such that $2w + e > 2t$, to demonstrate that the decoder may fail in such a case. 3 errors and 9 erasures are added.
- In the seventh and last case, we show the encoding output of the polynomial $x + x^{77}$.

VII. Outputs

A. *Case 1 - Less than t errors and no erasures*

Received Codeword :0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded Codeword : 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded message: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 1

B. *Case 2 - More than t errors and no erasures*

Received Codeword : 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

The first 8 bits have been flipped. The decoder fails in this case. To flag this, the received codeword is a vector of -1s of size 127

C. *Case 3 - Less than $2t$ erasures and no errors*

Received Codeword : 2 2 2 2 2 2 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded Codeword : 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded message : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 1

D. *Case 4 - More than than $2t$ erasures and no errors*

Received Codeword : 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 2 2 2 2 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

The decoder fails in this case and to flag the same the codeword is written as a sequence of -1s.

E. *Case 5 - w errors and e erasures where $2w + e \leq t$*

Received Codeword :0 1 1 1 2 2 2 2 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded codeword: 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded message :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 1

F. *Case 6 - $2w + e > t$*

Received Codeword : 0 1 1 0 2 2 2 2 2 2 2 0 2 0 0 0 2 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
Decoded Codeword : The decoder fails in this case. and is flagged similarly.

G. *Case 7 - Encoding the polynomial $x + x^7 7$*

Message Codeword : 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1
Encoded Codeword : 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

VIII. Conclusion

The project implements a systematic BCH encoder and decoder. The decoding algorithm used is Berlekamp Massey algorithm optimised for binary codewords. Implementation in MATLAB uses various Galois Field functions like gfadd and gfmul. The code was tested on solved examples in Shu Lin and Huffman and Pless for smaller codewords. For larger codewords $(n = 127)$, various cases were taken for a particular message to demonstrate that the decoder can correct up to t errors and $2t$ erasures.

Acknowledgment

The authors would like to thank Professor Pradeep Sarvepalli for giving us the opportunity to work on this project. We would also like to thank Arpan Paul and Harikumar for the enlightening discussions we had while solving this assignment.

References

[1] Shu Lin, Costello, *Error Control Coding - Fundamentals and Applications 2ed.*

[2] Todd K Moon, *Error-Correction Coding*

[3] Huffman and Pless, *Fundamentals of error correcting codes*