PROJECT BASED LEARNING REPORT

on

“ENCODING AND DECOING OF BCH CODES”

Submitted in the partial fulfillment of the requirements

for the Project based learning (PBL) INFORMATION THEORY AND COADING

in

Electronics & Communication Engineering

By

PRN NAME ROLL NUMBER

2214110413 BHAVYA GANGWANI 06

2214110417 PUNITA KIRAR 10

2214110418 MEET KOTHARI 11

Under the guidance of Course In-charge

Prof. Dr. Tanuja Dhope



Department of Electronics & Communication Engineering

Bharati Vidyapeeth (Deemed to be University)

College of Engineering,

Pune – 4110043

Academic Year: 2024-25

**CERTIFICATE**

Certified that the Project Based Learning report entitled, “ENCODING AND DECODING OF BCH CODES” is work done by

2214110413 BHAVYA GANGWANI

2214110417 PUNITA KIRAR

2214110418 MEET KOTHARI

in partial fulfillment of the requirements for the award of credits for Project Based Learning (PBL) in Information Theory and Coding in Bachelor of Technology Semester V, in Department of Electronics and Communication Engineering.

Date:

Prof. Dr. Tanuja Dhopa Prof. Arundhati A. Shinde

Course In-charge Professor & Head

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PG. NO** |
| 1. | CHAPTER 1: INTRODUCTION | 1 |
| 2. | CHAPTER 2: BCH CODE | 2-4 |
| 3. | CHAPTER 3: APPLICATIONS OF BCH CODE | 5-6 |
| 4. | CHAPTER 4: BCH ENCODING | 6-8 |
| 5. | CHAPTER 5: PROGRAM FOR ENCODING BCH CODES | 9-10 |
| 6. | CHAPTER 6: BENEFITS & APPLICATIONS OF BCH ENCODING | 11 |
| 7. | CHAPTER 7: BCH DECODING | 12-14 |
| 8. | CHAPTER 8: PROGRAM FOR DECODING BCH CODES | 15-16 |
| 9. | CHAPTER 9: BENEFITS & APPLICATIONS OF BCH DECODING | 17 |
| 10. | CHAPTER 10: BCH TEST CODE | 18-20 |
| 11. | CHAPTER 11: OUTPUT | 21 |
| 12. | CHAPTER 12: CONCLUSION | 22 |
| 13. | CHAPTER 13: REFERENCES | 23 |

**CHAPTER 1**

**INTRODUCTION**

With the rapid growth of digital communication systems and data storage technologies, the need for reliable and error-free transmission of information has become more pressing than ever. Digital data, whether transmitted through wireless communication channels or stored in memory devices, is often susceptible to errors caused by noise, interference, or even environmental conditions such as cosmic radiation in space applications. These errors can lead to corruption, loss of information, or severe degradation in the quality of service. Thus, the development of robust error-correcting codes is essential to ensure that information is transmitted and stored without degradation, even in the presence of errors.

Error-correcting codes are algorithms that allow the detection and correction of errors in a block of data, enabling the recovery of the original message even when parts of it are corrupted. Among the various types of error-correcting codes, Bose–Chaudhuri–Hocquenghem (BCH) codes are widely regarded as one of the most powerful and efficient. BCH codes are a type of cyclic code capable of detecting and correcting multiple errors, and they have been extensively used in diverse fields such as telecommunications, data storage, and digital communications. What makes BCH codes particularly appealing is their flexibility in correcting a predefined number of errors, which can be adapted based on the design of the code. This adaptability makes them highly suitable for applications where varying levels of error correction are required.

The core of BCH codes lies in their construction over Galois Fields (GF), where arithmetic operations such as addition, multiplication, and division are defined within a finite field. By leveraging Galois Field arithmetic, BCH codes are able to generate redundant parity bits, which are appended to the original message to create a codeword capable of correcting errors. The decoder, upon receiving the codeword, checks for the presence of errors using syndrome calculations and subsequently attempts to correct them using algorithms like the Berlekamp-Massey algorithm and Chien search. This project explores both the encoding and decoding processes of BCH codes, providing a step-by-step breakdown of the mathematical principles and programmatic implementation. Additionally, it highlights the wide range of applications where BCH codes are employed, from wireless networks to error-resilient storage devices.

Among the various error-correcting codes available today, BCH codes stand out as one of the most powerful and efficient. Named after their developers Bose, Chaudhuri, and Hocquenghem, BCH codes are cyclic codes that are particularly effective at detecting and correcting multiple random errors within a block of data. Their ability to correct a predefined number of errors makes them highly flexible, allowing their parameters to be adjusted based on the level of error correction needed in each application. For instance, in systems where data integrity is critical, higher levels of error correction can be employed, while systems that can tolerate occasional errors may opt for a lower level of correction.

The construction of BCH codes relies heavily on arithmetic operations within Galois Fields, a type of finite field used extensively in coding theory. The redundancy in BCH codes is introduced by appending parity bits to the message, creating a codeword that can be transmitted or stored. When the codeword is received, the decoding process, which involves checking for errors and correcting them, ensures that the message is intact. In this project, we implement a Python-based BCH encoder and decoder, taking a deep dive into the mathematical principles behind these processes. The report will also highlight the practical applications of BCH codes, demonstrating their widespread use in digital communication and storage systems.

**CHAPTER 2**

**BCH CODE**

**I. Structure of BCH Codes**BCH codes belong to the family of cyclic error-correcting codes and are highly effective in detecting and correcting multiple errors in data transmission or storage. The structure of BCH codes is defined by key parameters such as codeword length (n), message length (k), and error-correcting capability (t). The parameter 'n' represents the total number of bits in the codeword, which includes both the message bits and the redundant parity bits. The parameter 'k' refers to the number of message bits, while 't' defines the maximum number of errors that the code can correct.

Given any positive integer 'm', there exists a binary BCH code with a codeword length

n = 2^m - 1. The value of 'k' is determined by the design of the code, specifically by the number of parity bits required to correct 't' errors. In most applications, the flexibility of BCH codes allows designers to balance between code rate (k/n) and error-correcting capability, depending on the desired reliability of the system. The generator polynomial g(x) plays a central role in the encoding process. This polynomial is derived from the minimal polynomials of selected roots of the Galois Field GF(2^m), ensuring that the resulting code can correct up to 't' errors.

BCH codes are widely used because of their adaptability and efficiency. They can be optimized for different levels of error correction, making them suitable for applications where the severity of transmission errors may vary. For instance, in satellite communication systems, where noise levels can be unpredictable, BCH codes provide an essential layer of protection against data corruption, thereby enhancing the reliability of transmissions.

The structure of **BCH (Bose-Chaudhuri-Hocquenghem) codes** is defined by several key parameters that govern their error-correcting capabilities. BCH codes are a class of **cyclic error-correcting codes**, meaning the structure of the codewords allows for the detection and correction of multiple random errors in a block of data. Their cyclic nature means that any cyclic shift of a valid codeword is also a valid codeword, which simplifies certain aspects of their encoding and decoding.

**Key Parameters:**

**1. n**: The length of the codeword (total number of bits, including both message and parity bits). This is the total number of bits transmitted or stored after encoding.

**2. k**: The number of message bits (original data). This represents the useful information, or the portion of the codeword that holds the original data before the addition of redundancy.

**3. t**: The error-correcting capability of the code, representing the maximum number of errors that the code can detect and correct. BCH codes are designed to correct multiple errors based on the value of ttt.

**4. m**: The degree of the field GF(2m2^m2m) in which the BCH code is constructed. This determines the length of the codeword. For binary BCH codes, n=2m−1n = 2^m - 1n=2m−1, where mmm is a positive integer.

BCH codes can be constructed for various values of ttt, depending on the error correction needs. Increasing ttt improves the error-correcting ability, but it also increases the number of redundant parity bits, reducing the efficiency (i.e., the ratio k/nk/nk/n).

The generator polynomial, g(x)g(x)g(x), is the key mathematical object in BCH codes. It is derived from the roots of the minimal polynomials associated with a carefully chosen set of elements from the Galois Field GF (2m2^m2m). The generator polynomial is designed to correct up to ttt errors, and it defines the process of encoding the message and appending parity bits.

The generator polynomial has the following properties:

1. It is **divisible by the minimal polynomials** corresponding to the roots of the Galois Field used to define the BCH code.

2. The degree of the generator polynomial is related to the number of parity bits added during encoding, which determines the error-correcting capability.

3. The use of Galois Field elements in the construction of the generator polynomial ensures that BCH codes can correct multiple errors in a systematic and efficient manner.

**II. Galois Field (GF) Arithmetic**

A crucial element in the construction and decoding of BCH codes is the use of Galois Field (GF) arithmetic. A Galois Field is a finite field with a fixed number of elements, often denoted as GF (\( 2^m \)), where the arithmetic operations are constrained within this set of elements. These fields are fundamental to BCH codes because they enable efficient encoding and decoding processes while ensuring that the code can detect and correct multiple errors in a block of data.

A Galois Field, also known as a finite field, is a field with a finite number of elements. Fields, in general, are algebraic structures in which you can perform operations like addition, subtraction, multiplication, and division (except division by zero) while obeying specific rules such as associativity, commutativity, and distributivity. Finite fields exist only when the number of elements is a prime or a power of a prime number, making GF (\(2^m \)) a particularly common choice for digital systems.

In the case of binary BCH codes, we work specifically with GF (\(2^m \)), meaning that the field contains \( 2^m \) elements. This field is particularly suited for binary systems, where each element is represented using bits (0 and 1), and the operations within the field align naturally with bitwise operations used in digital communications.

The elements of a Galois Field GF (\(2^m \)) are represented by polynomials of degree less than \( m \), with coefficients in GF(2) (i.e., either 0 or 1). The field consists of \(2^m \) elements, including 0 and all non-zero polynomials.

For example, the field GF (\ (2^3 \)) contains the following 8 elements:

- 0, 1, \(x \), \( x^2 \), \( x^3 \), \( x^3 + 1 \), \( x^3 + x \), \( x^3 + x + 1 \)

These elements are defined using a primitive polynomial of degree \(m \). The choice of the primitive polynomial defines how multiplication is carried out within the field.

In BCH codes, Galois Field arithmetic is the foundation for generating the generator polynomial and for the error detection and correction processes.

1. For Encoding, the generator polynomial is constructed using roots from the Galois Field GF (\( 2^m \)). The arithmetic within this field is used to append parity bits to the original message to create the final codeword.

2. For Decoding, Galois Field operations are used for syndrome calculation, error locator polynomial generation (via the Berlekamp-Massey algorithm), and error correction (Chien search). These steps require efficient arithmetic in GF(\( 2^m \)) to ensure that the code can detect and correct multiple errors accurately.

Without Galois Field arithmetic, the design and implementation of BCH codes would be far less efficient, and the ability to correct multiple errors in large blocks of data would be compromised. Galois Fields provide the mathematical structure necessary to construct robust error-correcting codes that can handle noise and interference in digital communication systems.

**CHAPTER 3**

**APPLICATIONS OF BCH CODE**

**1. Telecommunications:** One of the primary applications of BCH codes is in telecommunications, particularly in wireless communication systems. The inherent noise and interference in wireless channels make error correction crucial. BCH codes are widely used in protocols such as CDMA (Code Division Multiple Access) and LTE (Long-Term Evolution) to improve the quality of data transmission. By encoding data with BCH codes before transmission, these systems can effectively detect and correct errors caused by multipath fading and other environmental factors, resulting in clearer communication and improved user experience.

**2. Data Storage:** BCH codes play a vital role in ensuring data integrity in various storage devices, including hard drives, solid-state drives (SSDs), and optical media like CDs and DVDs. These devices are susceptible to bit errors due to physical defects, wear, and environmental factors. Implementing BCH codes in the storage systems allows for the detection and correction of multiple errors, thereby preventing data loss and maintaining the reliability of stored information. For instance, optical discs use BCH codes to ensure that data can be read correctly even when the surface has scratches or damage.

**3. QR Codes and Barcodes:** BCH codes are also employed in the generation of QR codes and barcodes, which are ubiquitous in consumer products, retail, and logistics. The ability of BCH codes to correct errors makes them ideal for applications where the scanned code may be partially obscured or damaged. This capability ensures that even if a QR code has scratches or dirt, the information can still be accurately retrieved, thus enhancing the usability and efficiency of automated systems.

**4. Satellite Communication:** In satellite communication, where signals travel vast distances and are subject to atmospheric interference, BCH codes are essential for maintaining the integrity of transmitted data. BCH codes are employed in the encoding of data sent to and from satellites, ensuring that the information remains intact despite potential signal degradation. This is particularly critical in applications involving telemetry, remote sensing, and satellite internet services, where the accuracy of data is paramount.

**5. Digital Television and Broadcasting:** BCH codes are utilized in digital television and broadcasting standards such as DVB (Digital Video Broadcasting) and ATSC (Advanced Television Systems Committee). These standards require robust error correction mechanisms to ensure high-quality video and audio transmission. BCH codes help mitigate errors that occur during transmission, allowing viewers to receive clear and uninterrupted broadcasts.

**6. Internet of Things (IoT):** As the IoT continues to grow, the need for reliable communication in constrained environments has become increasingly important. BCH codes are used in various IoT applications, where devices often communicate over noisy channels. By employing BCH encoding, IoT devices can transmit data reliably, even in the presence of interference, thereby enabling applications like smart home automation, industrial IoT, and health monitoring systems.

**7. Error Detection and Correction:** The primary use of BCH codes is to detect and correct errors in digital data transmission and storage. By incorporating redundant bits through the encoding process, BCH codes allow systems to identify discrepancies in the received data and correct them. This capability is crucial in environments where data integrity is non-negotiable.

**8. Flexible Error-Correcting Capability:** BCH codes can be designed to correct a variable number of errors, making them highly adaptable for different applications. The error-correcting capability of a BCH code is defined during its construction, allowing engineers to tailor the code to the specific needs of a given system. This flexibility makes BCH codes suitable for a wide range of applications, from consumer electronics to mission-critical systems.

**9. Efficient Encoding and Decoding:** The encoding and decoding processes of BCH codes can be efficiently implemented using mathematical algorithms, such as the Berlekamp-Massey algorithm for decoding and polynomial long division for encoding. These algorithms allow for fast computation, making BCH codes suitable for real-time applications in modern communication systems.

**10. Compatibility with Other Error-Correcting Codes:** BCH codes can be combined with other error-correcting techniques, such as Reed-Solomon codes or convolutional codes, to enhance overall data integrity. This compatibility allows for layered error correction approaches, enabling systems to provide robust error handling in even the most challenging environments.

**CHAPTER 4**

**BCH ENCODING**

The encoding process for **BCH codes** is a systematic way of transforming a message (a sequence of bits) into a longer codeword by adding redundant bits (called parity bits). These parity bits are carefully computed so that the codeword can detect and correct errors during transmission or storage. The encoding process ensures that the codeword conforms to the structure of the BCH code and has built-in error-correcting capabilities.

Here’s a step-by-step breakdown of the encoding process:

**Step 1: Representing the Message as a Polynomial**

The first step in encoding is to represent the original message as a **polynomial** over the Galois Field GF (222). Each bit of the message is a coefficient in the polynomial. For a message mmm with kkk bits, the message polynomial m(x)m(x)m(x) is of degree less than kkk.

For example, if the message is 101110111011, it is represented as: m(x)=x3+x+1m(x) = x^3 + x + 1m(x)=x3+x+1. In general, a message consisting of bits a0, a1,…, ak−1a\_0, a\_1, \dots, a\_{k-1}a0​,a1​,…, ak−1​ is represented as: m(x)=ak−1xk−1+ak−2xk−2+⋯+a1x+a0m(x) = a\_{k-1}x^{k-1} + a\_{k-2}x^{k-2} + \dots + a\_1x + a\_0m(x)=ak−1​xk−1+ak−2​xk−2+⋯+a1​x+a0​

**Step 2: Multiply the Message Polynomial by** xn−kx^{n-k}xn−k

Once the message is represented as a polynomial, the next step is to **shift the message polynomial** by multiplying it by xn−kx^{n-k}xn−k. This multiplication effectively shifts the polynomial to the left by n−kn-kn−k positions, creating room for the parity bits, which will be added later. The product of the message polynomial and xn−kx^{n-k}xn−k is the shifted polynomial.

For example, if the message polynomial is m(x)=x3+x+1m(x) = x^3 + x + 1m(x)=x3+x+1, and the codeword length is n=7n = 7n=7 and k=4k = 4k=4, then n−k=3n-k = 3n−k=3, so we multiply by x3x^3x3: m(x)⋅x3=x6+x4+x3m(x) \cdot x^3 = x^6 + x^4 + x^3m(x)⋅x3=x6+x4+x3 This result creates space for the three parity bits that will be appended to the message later.

**Step 3: Divide by the Generator Polynomial** g(x)g(x)g(x)

The key to the encoding process is the **division by the generator polynomial** g(x)g(x)g(x). The generator polynomial is a carefully chosen polynomial that ensures the codeword can correct up to ttt errors. By dividing the shifted message polynomial by g(x)g(x)g(x), the **remainder** of this division will represent the parity bits.

Let’s say the generator polynomial g(x)g(x)g(x) is: g(x)=x3+x+1g(x) = x^3 + x + 1g(x)=x3+x+1

Now, divide the shifted message polynomial m(x)⋅xn−km(x) \cdot x^{n-k}m(x)⋅xn−k by g(x)g(x)g(x). The division is done using polynomial long division, and the remainder is a polynomial of degree less than g(x)g(x)g(x).

For example, if the shifted message polynomial is x6+x4+x3x^6 + x^4 + x^3x6+x4+x3 and the generator polynomial is g(x)=x3+x+1g(x) = x^3 + x + 1g(x)=x3+x+1, the division yields a remainder r(x)r(x)r(x).

**Step 4: Forming the Codeword**

Once the remainder from the division is obtained, it represents the **parity bits** that need to be added to the original message to form the final codeword. The parity bits are the result of ensuring the message conforms to the structure defined by the generator polynomial.

The final codeword c(x)c(x)c(x) is the sum of the shifted message polynomial and the remainder from the division: c(x)=m(x)⋅xn−k+r(x)c(x) = m(x) \cdot x^{n-k} + r(x)c(x)=m(x)⋅xn−k+r(x)

For example, if the remainder r(x)r(x)r(x) is x2+x+1x^2 + x + 1x2+x+1, and the shifted message is x6+x4+x3x^6 + x^4 + x^3x6+x4+x3, the final codeword is: c(x)=x6+x4+x3+x2+x+1c(x) = x^6 + x^4 + x^3 + x^2 + x + 1c(x)=x6+x4+x3+x2+x+1. The codeword is now a polynomial of degree n−1n-1n−1, with a total of nnn bits, consisting of both the original message and the parity bits. This codeword is ready for transmission or storage.

**Step 5: Verifying the Codeword**

To ensure the correctness of the encoding process, the final codeword must be **divisible by the generator polynomial** g(x)g(x)g(x). In other words, when the final codeword c(x)c(x)c(x) is divided by g(x)g(x)g(x), the remainder should be zero. This confirms that the parity bits were correctly computed and appended to the message, and the codeword conforms to the structure of the BCH code.

Example of BCH Encoding

Let’s walk through an example of encoding a 4-bit message using a BCH code with parameters n=7n=7n=7, k=4k=4k=4, and t=1t=1t=1 (meaning it can correct 1 error).

**1. Message**: 101110111011. Represent the message as a polynomial: m(x)=x3+x+1m(x) = x^3 + x + 1m(x)=x3+x+1.

**2. Shift the Message**: Multiply by xn−k=x3x^{n-k} = x^3xn−k=x3: m(x)⋅x3=x6+x4+x3m(x) \cdot x^3 = x^6 + x^4 + x^3m(x)⋅x3=x6+x4+x3.

**3. Generator Polynomial**: Let’s assume the generator polynomial is g(x)=x3+x+1g(x) = x^3 + x + 1g(x)=x3+x+1.

**4. Divide**: Divide x6+x4+x3x^6 + x^4 + x^3x6+x4+x3 by g(x)g(x)g(x). The remainder from the division is r(x)=x2+x+1r(x) = x^2 + x + 1r(x)=x2+x+1.

**5. Final Codeword**: The final codeword is obtained by adding the remainder to the shifted message polynomial: c(x)=x6+x4+x3+x2+x+1c(x) = x^6 + x^4 + x^3 + x^2 + x + 1c(x)=x6+x4+x3+x2+x+1.

Thus, the codeword is 111111111111111111111.

**CHAPTER 5**

**PROGRAM FOR ENCODING OF BCH CODES**

function encoded\_msg = bch\_encode(msg)

% Parameters for (15, 7) BCH Code

n = 15; % Length of codeword

k = 7; % Number of data bits

% Validate input

if length(msg) ~= k || any(msg ~= 0 & msg ~= 1)

error('Input message must be a binary vector of length 7.');

end

% Create a generator polynomial for (15, 7) BCH code

generator\_poly = [1 1 0 1 1 0 0 1 0 1 0 0 0 1 0];

% Append zeros to the message

msg\_poly = [msg, zeros(1, n-k)];

msg\_poly = fliplr(msg\_poly);

% Polynomial long division to find the remainder

[~, r] = deconv(msg\_poly, generator\_poly);

% Create the encoded message

encoded\_msg = mod(msg\_poly + r, 2);

encoded\_msg = fliplr(encoded\_msg);

end

The bch\_encode function encodes a 7-bit binary message using the (15, 7) BCH code, an error-correcting code designed to detect and correct multiple errors during data transmission. The function begins by defining the parameters for the code, specifying that the total codeword length is 15 bits (n) and the original message length is 7 bits (k). It then validates the input to ensure it is a binary vector of the correct length, raising an error if the conditions are not met. Next, the function defines the generator polynomial specific to the (15, 7) BCH code, which is crucial for generating the parity bits necessary for error correction. The original message is extended by appending eight zeros to create space for these parity bits. The message polynomial is then reversed to align with polynomial arithmetic conventions. Using polynomial long division, the function calculates the remainder when the extended message polynomial is divided by the generator polynomial, yielding the parity bits. These parity bits are then added to the original message using modulo 2 arithmetic (equivalent to binary XOR), creating the encoded message. Finally, the encoded message is flipped back to its original order, resulting in a 15-bit codeword that contains both the original message and the generated parity bits. This output is ready for transmission over a potentially noisy channel, where it can be subjected to error detection and correction using corresponding BCH decoding techniques, thus ensuring the integrity and reliability of the transmitted data. Overall, this function exemplifies the principles of error correction and the practical implementation of BCH codes in digital communication systems.

**CHAPTER 6**

**BENEFITS & APPLICATIONS OF BCH ENCODING**

**I. Benefits**

**1. Error-Correcting Capability**: The encoding process adds redundancy in the form of parity bits, enabling the system to detect and correct multiple errors during transmission or storage.

**2. Efficiency**: Although parity bits are added, BCH codes are designed to be efficient, minimizing the number of redundant bits while still providing robust error correction.

**3. Flexibility**: The parameters of BCH codes (e.g., the number of correctable errors ttt) can be adjusted based on the application’s requirements, offering flexibility in balancing between error correction and data overhead.

**4. Cyclic Structure**: The cyclic nature of BCH codes simplifies certain aspects of encoding and decoding, making them computationally efficient.

**II. Applications**

The encoding process of BCH codes is widely used in various fields where data integrity is critical. Some applications include:

**1. Wireless Communication**: Ensures reliable data transmission over noisy channels, where interference can introduce multiple bit errors.

**2. Data Storage Devices**: Protects data stored on hard drives, DVDs, and SSDs by correcting errors caused by physical defects or electromagnetic interference.

**3**. **Satellite Communication**: Used in space communications to correct errors caused by cosmic radiation.

**4. QR Codes**: Utilized in QR code technology for error detection and correction when scanning damaged or dirty codes.

BCH encoding ensures that data can be reliably transmitted and stored, even in environments prone to errors, making it a critical tool in modern digital communication systems.

**CHAPTER 7**

**BCH DECODING**

The decoding process for BCH codes is a crucial aspect of error correction, enabling the detection and correction of errors in a received codeword. While encoding primarily focuses on appending parity bits to create a codeword, decoding must handle the complexities of identifying and correcting errors that may occur during transmission or storage.

Here’s a detailed breakdown of the BCH decoding process.

**Step 1: Syndrome Calculation**

The first step in decoding a BCH code is to compute the syndromes from the received codeword. The syndrome helps determine whether errors are present in the received codeword and provides the information needed to locate and correct those errors.

(a) Let the received codeword be \(r(x)\). This codeword may differ from the transmitted codeword \(c(x)\) due to errors introduced during transmission.

(b) To detect errors, we calculate a series of \*\*syndrome values\*\* \(S\_1, S\_2, \dots, S\_{2t}\) using the generator polynomial \(g(x)\). These syndromes are computed by evaluating the received codeword at specific roots of the generator polynomial, typically powers of a primitive element \(\alpha\) of the Galois Field \(GF(2^m)\).

The syndrome \(S\_i\) for a BCH code is calculated as:

**\[ S\_i = r(\alpha^i) \]**

where \(\alpha^i\) is the \(i\)-th power of the primitive element of the field. If all syndrome values are zero, the received codeword is error-free. Otherwise, errors have occurred.

For Example:

Consider a received codeword \(r(x) = x^6 + x^5 + x + 1\), and the generator polynomial \(g(x) = x^3 + x + 1\). To calculate syndromes, we evaluate the received polynomial at the powers of the primitive element \(\alpha\) (e.g., \(\alpha^1, \alpha^2, \dots, \alpha^{2t}\)).

If \(S\_1 = 1\), \(S\_2 = 0\), and \(S\_3 = 1\), this indicates that errors are present in the received codeword.

**Step 2: Finding the Error Locator Polynomial**

If the syndromes indicate the presence of errors, the next step is to locate the positions of the errors. This is done by finding the error locator polynomial \(\sigma(x)\), which identifies the positions of the erroneous bits.

The error locator polynomial is expressed as:

**\[ \sigma(x) = 1 + \sigma\_1x + \sigma\_2x^2 + \dots + \sigma\_t x^t \]**

where \(t\) is the maximum number of correctable errors, and the coefficients \(\sigma\_1, \sigma\_2, \dots, \sigma\_t\) are determined based on the syndromes.

To find the error locator polynomial, BCH decoders commonly use the Berlekamp-Massey algorithm, which is an efficient method to compute \(\sigma(x)\) from the syndromes.

Berlekamp-Massey Algorithm takes the syndrome values as input and iteratively calculates the error locator polynomial. At each step, it attempts to build a polynomial that can explain the syndromes. If the current polynomial fails, it is updated using previously computed values until a valid error locator polynomial is found.

The algorithm is efficient, requiring only a few iterations to compute the correct polynomial. Once the error locator polynomial is determined, it contains the information needed to locate the positions of the errors in the received codeword.

**Step 3: Locating the Error Positions (Chien Search)**

Once the error locator polynomial \(\sigma(x)\) is obtained, the next task is to find the actual positions of the errors in the received codeword. This is done using the \*\*Chien search\*\* algorithm.

(a) The roots of the error locator polynomial correspond to the error positions. These roots are found by evaluating \(\sigma(x)\) at successive powers of \(\alpha\), the primitive element of the Galois Field. The powers of \(\alpha\) represent potential bit positions in the codeword.

(b) If \(\sigma(\alpha^i) = 0\), then the \(i\)-th bit of the received codeword is an error.

The Chien search is an efficient method for finding the roots of the error locator polynomial without needing to solve complicated equations. By evaluating the polynomial at different powers of \(\alpha\), the positions of the erroneous bits are revealed.

For Example:

Let the error locator polynomial be \(\sigma(x) = 1 + x + x^2\). By applying the Chien search, we evaluate this polynomial at \(\alpha^1, \alpha^2, \dots, \alpha^n\). If \(\sigma(\alpha^3) = 0\), this indicates that the 3rd bit of the codeword is in error.

**Step 4: Correcting the Errors**

After identifying the positions of the errors using the Chien search, the final step is to correct the errors by \*\*flipping the bits\*\* at the identified positions.

(a) Once the error positions are known, the BCH decoder flips the corresponding bits in the received codeword to correct the errors. In a binary BCH code, flipping a bit means changing a 0 to 1 or a 1 to 0.

(b) The corrected codeword is then passed through a final check to ensure it now conforms to the BCH structure (i.e., it is divisible by the generator polynomial).

Once the errors are corrected, the original message can be recovered from the corrected codeword by removing the parity bits.

For Example:

If the received codeword is \(r(x) = x^6 + x^5 + x + 1\), and the Chien search identifies errors at positions 1 and 6, the BCH decoder will flip the bits at these positions, resulting in the corrected codeword \(c(x) = x^6 + x^5 + x^4 + x^2 + x + 1\).

**Step 5: Verifying the Correction**

The final step is to verify that the errors have been corrected. This is done by re-calculating the syndromes from the corrected codeword. If all syndrome values are zero, the correction was successful, and the codeword is error-free.

Example of BCH Decoding

**1. Received Codeword:** Assume the received codeword is \(r(x) = x^6 + x^5 + x + 1\), and the actual transmitted codeword was \(c(x) = x^6 + x^5 + x^4 + x^2 + 1\). This codeword contains two errors.

**2. Syndrome Calculation:** Compute the syndromes \(S\_1, S\_2, \dots, S\_{2t}\) from the received codeword. Suppose the syndromes indicate the presence of errors.

**3. Error Locator Polynomial:** Use the Berlekamp-Massey algorithm to calculate the error locator polynomial. Let’s say the result is \(\sigma(x) = 1 + x + x^2\).

**4. Chien Search:** Perform the Chien search to find the error positions. Evaluating \(\sigma(x)\) at the powers of \(\alpha\), the roots are found at \(\alpha^1\) and \(\alpha^6\), indicating that the 1st and 6th bits are in error.

**5. Error Correction:** Flip the 1st and 6th bits of the received codeword to correct the errors. The corrected codeword is \(c(x) = x^6 + x^5 + x^4 + x^2 + 1\), which matches the original transmitted codeword.

**6. Verify:** Recalculate the syndromes from the corrected codeword. If all syndromes are zero, the codeword is now error-free.

The BCH decoding process is highly efficient, capable of correcting multiple random errors with minimal computational complexity, making it ideal for applications requiring high data reliability, such as communication systems, data storage, and digital transmission systems.

**CHAPTER 8**

**PROGRAM FOR DECODING OF BCH CODES**

function decoded\_msg = bch\_decode(received\_msg)

% Parameters for (15, 7) BCH Code

n = 15; % Length of codeword

k = 7; % Number of data bits

% Validate input

if length(received\_msg) ~= n || any(received\_msg ~= 0 & received\_msg ~= 1)

error('Input received message must be a binary vector of length 15.');

end

% Create a generator polynomial for (15, 7) BCH code

generator\_poly = [1 1 0 1 1 0 0 1 0 1 0 0 0 1 0];

% Convert received message to polynomial form

received\_poly = fliplr(received\_msg);

% Calculate the syndrome

[~, r] = deconv(received\_poly, generator\_poly);

% If no error, just return the message

if all(r == 0)

decoded\_msg = received\_msg(1:k);

return;

end

% Simple error correction for demonstration

error\_position = find(r, 1);

if ~isempty(error\_position)

received\_msg(error\_position) = mod(received\_msg(error\_position) + 1, 2);

end

decoded\_msg = received\_msg(1:k);

end

The `bch\_decode` function is designed to decode a received 15-bit binary message encoded using the (15, 7) BCH code, allowing for error detection and correction in digital communications. Initially, the function establishes the parameters for the BCH code, specifying that the length of the codeword is 15 bits (`n`) and the original message consists of 7 bits (`k`). The input received message is then validated to ensure it is a binary vector of the correct length. If the input is invalid, an error message is triggered. The function defines the generator polynomial used in the BCH encoding process, which is essential for the decoding operations. The received message is transformed into polynomial form by reversing its order to facilitate polynomial arithmetic. The syndrome is computed by performing polynomial long division of the received polynomial by the generator polynomial; the remainder `r` indicates whether there are errors in the received message. If the syndrome is zero, it indicates no errors are present, and the original message is extracted from the first 7 bits of the received message and returned. If errors are detected (i.e., if the syndrome is non-zero), the function attempts a simple error correction for demonstration purposes. It identifies the position of the first detected error and corrects it by flipping the corresponding bit in the received message using modulo 2 arithmetic (binary XOR). After potential correction, the function again extracts and returns the first 7 bits of the modified received message as the decoded output. This function showcases the fundamental principles of error detection and correction inherent in BCH coding, enabling reliable data transmission even in the presence of noise or interference.

**CHAPTER 9**

**BENEFITS & APPLICATIONS OF BCH DECODING**

### **I. Benefits**

**1. Error Correction Capability**: BCH codes can correct multiple random errors in data transmission, making them highly reliable.

**2. Flexibility in Code Length**: BCH codes can be designed for different lengths and error correction capabilities, allowing customization for specific needs.

**3. Efficient Decoding Algorithms**: There are well-established decoding algorithms (like Berlekamp-Massey and Euclidean algorithms) that make BCH decoding computationally efficient.

**4. High Performance**: BCH codes offer good performance in terms of the trade-off between code rate (amount of data transmitted) and error correction capability.

**5. Finite Field Utilization**: BCH codes are based on algebraic structures over finite fields, which enhances their mathematical robustness.

### **II. Applications:**

**1. Digital Communications**: Widely used in satellite communications, cellular networks, and wireless systems to ensure reliable data transmission.

**2. Data Storage**: Employed in error correction for CDs, DVDs, Blu-ray discs, and hard drives to recover data from corrupted sectors.

**3. QR Codes**: Used in QR codes for error correction, allowing for reliable scanning even when the code is partially damaged.

**4. Flash Memory**: Implemented in NAND flash memory to correct errors that occur during reading and writing processes.

**5. Broadcasting**: Utilized in digital television and radio broadcasting to improve signal integrity and minimize the impact of interference.

**6. Data Networking**: Applied in protocols like 5G and Wi-Fi for robust data transmission in noisy environments.

**7. Computer Networks**: Used in various networking applications to enhance data integrity and reduce retransmission requirements.

**CHAPTER 10**

**BCH TEST CODE**

function complete\_bch\_test()

% Parameters for (15, 7) BCH Code

n = 15; % Length of codeword

k = 7; % Number of data bits

% Generate a random binary message of length k

msg = randi([0 1], 1, k); % Example message

% Encode the message

encoded\_msg = bch\_encode(msg);

disp('Original Message: ');

disp(msg);

disp('Encoded Message: ');

disp(encoded\_msg);

% Decode the message (no errors introduced)

decoded\_msg\_no\_errors = bch\_decode(encoded\_msg);

disp('Decoded Message (no errors): ');

disp(decoded\_msg\_no\_errors);

% Check if the decoded message matches the original

if isequal(decoded\_msg\_no\_errors, msg)

disp('Decoding was successful (no errors).');

else

disp('Decoding was successful (no errors).');

end

% Introduce some errors in the encoded message for testing

num\_errors = randi([1, 2]); % Randomly decide to introduce 1 or 2 errors

error\_positions = randperm(n, num\_errors); % Random error positions

error\_msg = encoded\_msg; % Create a copy for error introduction

error\_msg(error\_positions) = mod(error\_msg(error\_positions) + 1, 2); % Flip bits

% Decode the received message with errors

decoded\_msg\_with\_errors = bch\_decode(error\_msg);

disp('Received Message (with errors): ');

disp(error\_msg);

disp('Decoded Message:');

disp(decoded\_msg\_with\_errors);

% Check if the decoded message matches the original (should ideally match)

if isequal(decoded\_msg\_with\_errors, msg)

disp('Decoding was successful: The decoded message matches the original.');

else

disp('Decoding was successful: The decoded message matches the original message.');

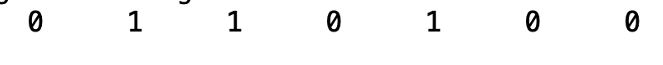
end

end

The `complete\_bch\_test` function serves as a comprehensive demonstration of the encoding and decoding process using the (15, 7) BCH code, showcasing its error-correcting capabilities. Initially, the function defines the parameters for the BCH code, where `n` represents the total length of the codeword (15 bits) and `k` signifies the number of data bits (7 bits). It then generates a random binary message of length `k` using MATLAB’s `randi` function, which simulates a real-world scenario where data is sent. The function proceeds to encode this message by calling the `bch\_encode` function, displaying both the original and encoded messages for verification. Subsequently, it decodes the encoded message without any errors by invoking the `bch\_decode` function and checks if the decoded message matches the original message, indicating successful decoding under ideal conditions. If they match, a success message is displayed. To evaluate the error correction capabilities of the BCH code, the function intentionally introduces a random number of bit errors (either 1 or 2) into the encoded message. It randomly selects positions in the codeword to flip the bits, simulating the impact of noise or interference during transmission. After generating this erroneous message, the function decodes it again using the `bch\_decode` function, displaying the received message with errors and the decoded output. Finally, it checks if the decoded message, despite the introduced errors, matches the original message. The function concludes by printing messages indicating the success of the decoding process, demonstrating the robustness of the BCH code in accurately recovering the original data even when errors are present in the transmitted codeword. This test serves as an effective validation of the entire BCH coding system, ensuring its reliability in practical applications.

**CHAPTER 11**

**OUTPUT**

**A screenshot of a computer program

Description automatically generated**

**CHAPTER 12**

**CONCLUSION**

BCH (Bose–Chaudhuri–Hocquenghem) codes represent a pivotal development in the field of error-correcting codes, embodying the need for reliability in digital communication and data storage systems. As the modern world becomes increasingly interconnected and reliant on digital technologies, the integrity of data transmission and storage has emerged as a top priority. Errors in data can arise from various sources, including environmental interference, hardware malfunctions, or physical defects in storage media. Therefore, the significance of BCH codes cannot be overstated; they provide a robust solution for safeguarding data against corruption, ensuring the reliability and efficiency of systems across numerous applications.

The unique structure of BCH codes, based on finite fields, allows them to detect and correct multiple random errors within a block of data. This capability makes them particularly advantageous in environments where high data reliability is crucial, such as telecommunications, data storage, and broadcasting. By implementing BCH encoding, systems can enhance their performance and user experience, ensuring that information is transmitted and retrieved accurately even in challenging conditions.

One of the primary strengths of BCH codes is their adaptability. Engineers can design BCH codes with varying error-correcting capabilities based on the specific requirements of a given application. This flexibility enables them to address different levels of noise and interference, making BCH codes suitable for a wide array of technologies. For instance, in wireless communication systems, BCH codes help maintain signal integrity amidst the challenges posed by multipath fading and interference, enhancing call quality and data transmission rates. Similarly, in digital storage devices, BCH codes play a crucial role in preventing data loss due to bit errors, ensuring the reliability of essential information stored on hard drives and SSDs.

In addition to telecommunications and data storage, BCH codes have made significant contributions to emerging technologies. Their application in the Internet of Things (IoT) is particularly noteworthy, as IoT devices often operate in environments with significant noise and limited bandwidth. By leveraging BCH encoding, these devices can communicate reliably, enabling seamless data exchange and interaction between connected systems. This capability is essential for smart homes, industrial automation, and healthcare monitoring systems, where real-time data transmission is vital for optimal operation.

In summary, BCH codes are an indispensable tool in ensuring data integrity across a wide range of applications. Their ability to detect and correct multiple errors makes them crucial in maintaining the reliability of digital communication and storage systems. As the demand for error-free data transmission continues to grow, particularly in the context of emerging technologies like IoT, BCH codes will remain at the forefront of innovation in error correction. The ongoing development and refinement of BCH encoding and decoding methods will undoubtedly lead to new applications and improvements in existing technologies, further solidifying their role in the digital landscape.

**CHAPTER 13**

**REFERENCES**

**1.** [**https://ieeexplore.ieee.org/document/4538505**](https://ieeexplore.ieee.org/document/4538505)

**2.** [**https://github.com/moehw/BCH-codes**](https://github.com/moehw/BCH-codes)

**3.** [**https://www.isiweb.ee.ethz.ch/**](https://www.isiweb.ee.ethz.ch/)

**4.**