PROJECT BASED LEARNING REPORT

on

“CYCLIC REDUNDANCY CHECK”

Submitted in the partial fulfillment of the requirements

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

in

Electronics & Communication Engineering

By

PRN NAME

2214110407 PRITUL AGRAWAL

2214110427 RAJAN KUMAR

2214110446 AYUSH MISHRA

Under the guidance of Course In-charge

Dr. TANUJA S. 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, “CYCLIC REDUNDANCY CHECK ON MATLAB” is work done by

2214110407 PRITUL AGRAWAL

2214110427 RAJAN KUMAR

2214110446 AYUSH MISHRA

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

Date:

Dr. Tanuja .S Dhope Dr. Arundhati A. Shinde

Course In-charge Professor & Head

**INDEX**

|  |  |  |
| --- | --- | --- |
| SR.NO. | CONTENT | PAGE NO. |
| 1 | Chapter 1   * Why do we need cyclic redundancy check * Solution of cyclic redundancy check | 1-3 |
| 2 | Chapter-2   * Introduction | 4-6 |
| 3 | Chapter-3   * Software Used | 8-9 |
| 4 | Chapter-4   * Code | 10-12 |
| 5 | Chapter-5   * Output | 13-14 |
| 6 | Chapter-6   * Conclusion * Outcome * Reference * Github link | 15-17 |

**CHAPTER-1**

**Why Do We Need “Cyclic Redundancy Check”**

We need Cyclic Redundancy Check (CRC) to detect errors in data transmission and storage. When data travels over a network or is stored on a device, it can become corrupted due to interference, noise, or hardware malfunctions. CRC helps ensure that the data received is exactly the same as the data sent.

**Data integrity:** It helps detect accidental changes in transmitted data. It ensures the correctness of the data by comparing the CRC value calculated on both the sending and receiving ends. If the values don’t match, it means the data is corrupted.

**Error detection:** CRC can identify corrupted data, so actions like retransmission can be initiated. CRC helps detect if any bits in the data have been altered during transmission. This is important in ensuring that the message arrives intact

**Efficiency**: It’s a lightweight way to detect errors without using much processing power.CRC provides a fast and lightweight method to check data integrity compared to other error-checking methods. It can detect common types of errors, such as single-bit errors and burst errors, with minimal computational effort.

**Reliability in Communication:** In networks, data corruption can occur frequently due to noise, signal degradation, or collisions. CRC provides a simple, efficient way to ensure data reliability, enabling error detection before the data is processed.

In summary, we need CRC to maintain data accuracy, reliability, and integrity in communication systems and storage devices, making sure corrupted data can be detected and addressed.

**Solution Of “Cyclic Redundancy Check”**

Cyclic Redundancy Check (CRC) is a method used to detect errors in data transmission or storage. Here's how it works in simple terms:

**Original Data:** When data (like files or messages) is sent or stored, CRC creates a special code (a "checksum") based on the content of the data. This code is generated by running the data through a mathematical formula.

**Checksum:** The checksum is a small sequence of numbers added to the end of the data. It acts as a summary of the data's content.

**Data Transmission:** When the data is received or retrieved, the system recalculates the checksum using the same formula as before.

**Comparison:** The system compares the original checksum with the new checksum. If both match, the data is likely correct. If they don’t match, it means some error has occurred during transmission or storage.

**Retransmission:** The corrupted data is requested again, usually in communication protocols like TCP.

**Error correction**: More advanced systems use algorithms that can correct certain types of errors without needing retransmission (like ECC in memory systems).

**CRC Calculation:** The sender generates a CRC value (checksum) by applying a mathematical algorithm to the data. This value is sent along with the data.

**Error Detection:** When the data reaches the receiver, the same CRC algorithm is applied to the received data. The result is compared with the original CRC value sent.

**Error Response:**

**If CRC values match:** The data is considered intact and error-free.

**If CRC values don’t match:** This indicates data corruption. The typical solutions include:

**Retransmission:** The receiver requests the sender to resend the corrupted data. This is commonly used in communication protocols like TCP.

In short, CRC helps maintain data accuracy by quickly identifying errors, allowing corrective measures to be taken.

The CRC method is fast, efficient, and widely used in network communications, file transfers, and digital storage to ensure data integrity. It can detect common types of errors but is not foolproof for detecting all possible errors.

**CHAPTER-2**

**INTRODUCTION**

Cyclic Redundancy Check (CRC) is a method used to detect errors in digital data, especially during data transmission. It works by generating a small, fixed-length code (the "checksum") from the data using a mathematical formula. When data is sent, the CRC value is calculated and sent along with it. The receiver then recalculates the CRC value from the received data and compares it with the transmitted CRC. If they match, the data is assumed to be correct; if not, an error has occurred, and the data may need to be retransmitted. CRC is widely used because it can quickly identify errors in large amounts of data without adding significant complexity.

Here’s a detailed explanation of how CRC works, its importance, and its applications.

**1. Purpose of CRC**

When data is transmitted over a network or stored on a disk, it is possible for errors to occur due to noise, interference, or physical damage. CRC is used to ensure the integrity of the data by detecting these errors. It’s a way of ensuring that the data received is the same as the data sent.

**2. How CRC Works**

CRC is based on binary division, where the data is treated as a large binary number. Here's a step-by-step breakdown of the process:

**Step 1: Data Representation**:The data to be transmitted is represented as a binary sequence (a string of bits).

**Step 2: Choose a Polynomial (Divisor):**A predetermined generator polynomial is chosen. The generator is represented as a binary number and is used as a divisor in the CRC calculation. This polynomial is agreed upon by both the sender and receiver.

**Step 3: Append Zeros to Data:**Before performing the division, the data is padded with a number of zeros equal to the length of the generator polynomial minus one. These zeros act as placeholders for the checksum that will be calculated.

**Step 4: Perform Binary Division:**The padded data is then divided by the generator polynomial using binary division. This division process is a series of XOR (exclusive OR) operations instead of traditional subtraction.

**Step 5: Calculate the Remainder (Checksum):**The result of the division is a remainder, which becomes the CRC checksum. This remainder is typically shorter than the original data and is attached to the end of the data to form the transmitted message.

**Step 6: Transmission and Validation:**The combined message (data + CRC checksum) is sent to the receiver. The receiver repeats the division process using the same generator polynomial. If the remainder from this division is zero, the data is assumed to be error-free.

**3. Example of CRC Calculation**

Let’s say we want to send the binary data 101110 using a generator polynomial 1101 (which represents a degree-3 polynomial).

**Append zeros:** The original data 101110 is extended by adding three zeros (since the generator is 3 bits long minus 1), becoming 101110000.

**Perform division**: We divide 101110000 by 1101 using binary division (XOR at each step).

**Find remainder:** The remainder of this division is the CRC, let’s say it turns out to be 011. So, the transmitted message will now be the original data 101110 followed by the CRC 011, resulting in 101110011.

**Receiver check:** Upon receiving 101110011, the receiver performs the same division. If the remainder is zero, the data is correct. If not, it indicates an error occurred during transmission.

**4. Advantages of CRC**

**Error detection:** CRC can detect many types of errors, including single-bit errors, double-bit errors, burst errors, and more.

**Efficiency:** It requires minimal computational power, making it ideal for hardware implementation.

**Low overhead:** The CRC value is small relative to the data size, so it doesn't add significant overhead to transmissions.

**5.Limitations of CRC.**

**Error correction**: CRC can only detect errors; it cannot correct them. Once an error is detected, the usual response is to request a retransmission.

**Vulnerability to some errors**: While CRC is effective for detecting most errors, it may not detect all possible error patterns, especially in very specific cases where the corrupted data still satisfies the CRC formula.

**6. Applications of CRC**

CRC is used in a wide variety of fields:

**Networking:** Ethernet, Wi-Fi, and many other network protocols use CRC to ensure data packets are not corrupted during transmission.

**File storage**: ZIP files, RAR files, and many other archive formats use CRC to verify data integrity.Checksums for storage: CRC is used to protect data on disks, SSDs, and other storage devices.

**CHAPTER-3**

** SOFTWARE USED**

**MATLAB**

MATLAB is a powerful tool, especially for tasks related to engineering, scientific research, and mathematical problem-solving. Here’s a more detailed look at its components and uses:

**1. Programming Language:**

MATLAB is primarily designed around matrix and array mathematics, which is why it excels at handling numerical data. The syntax is straightforward and user-friendly, making it easier for people to learn and use, even if they aren’t professional programmers.

**2. Key Capabilities:**

**Numerical Computing:** MATLAB can handle large sets of numbers, perform complex calculations, and manipulate matrices easily. This is especially useful in fields like linear algebra, signal processing, and control systems.

**Symbolic Math:** Besides numerical computing, MATLAB also supports symbolic math, allowing you to work with mathematical formulas symbolically rather than as numbers (with an additional package).

**Algorithm Development:** MATLAB is ideal for developing new algorithms because of its clear and concise coding structure.

**Data Analysis:** You can import data from various sources, process it, and extract meaningful patterns or trends, making it very popular in data science and research communities.

**3. Visualization:**

MATLAB is widely recognized for its advanced visualization tools. You can create high-quality 2D and 3D plots, histograms, contour plots, and even animations. This ability makes it great for interpreting results, analyzing trends, and presenting data in an understandable way.

**4. Toolboxes:**

MATLAB is extremely versatile because of its toolboxes, which are collections of specialized functions that extend its capabilities for specific fields. Some popular toolboxes include:

**Signal Processing Toolbox:** For analyzing and manipulating signals.

**Image Processing Toolbox**: For working with images and videos.

**Control Systems Toolbox:** For designing, simulating, and analyzing control systems.

**Machine Learning and Deep Learning Toolbox:** For building predictive models and neural networks.

These toolboxes make MATLAB an industry standard in many fields like aerospace, automotive, communications, and finance.

**5. Simulink:**

Simulink, an add-on product to MATLAB, is a graphical tool used to model, simulate, and analyze dynamic systems. This is particularly useful for industries that need to model systems over time, such as in robotics, electrical engineering, and automotive systems.

**6. Interfacing with Other Languages:**

MATLAB allows integration with other programming languages like C, C++, Java, and Python, which enables it to be part of larger software systems. It can call functions from other languages or export results to them.

**7. Applications:**

**Engineering:** MATLAB is widely used in electrical, mechanical, and civil engineering for simulations, system designs, and modeling.

**Finance:** Financial professionals use MATLAB to model financial markets, simulate portfolios, and develop risk management tools.

**Research:** MATLAB is a favorite in academia for its ability to handle data analysis, algorithm development, and modeling, particularly in scientific research.

**Education**: Many universities use MATLAB for teaching mathematics, engineering, and science because it simplifies complex tasks and provides immediate feedback on code.

In summary, MATLAB is a high-level computing environment that combines numerical calculations, algorithm development, data visualization, and simulation. It’s used across a wide range of industries for both academic and professional purposes because of its versatility and the depth of its tools.

**CHAPTER- 4**

**CODE**

function [encoded\_message, remainder] = crc\_encode(message, generator)

% Convert message and generator polynomial to binary

message\_bin = logical(message - '0');

generator\_bin = logical(generator - '0');

% Append zeros to the message (length of generator - 1)

padded\_message = [message\_bin, zeros(1, length(generator\_bin) - 1)];

% Initialize remainder with the padded message

if remainder(i) == 1

remainder = padded\_message;

% Perform the division using XOR

for i = 1:length(message\_bin)

remainder(i:i + length(generator\_bin) - 1) = xor(remainder(i:i + length(generator\_bin) - 1), generator\_bin);

end

end

% The CRC is the remainder after the division

crc = remainder(end - length(generator\_bin) + 2:end);

% Append the CRC to the original message

encoded\_message = [message\_bin, crc];

end

function is\_valid = crc\_check(received\_message, generator)

[~, remainder] = crc\_encode(received\_message, generator);

is\_valid = all(remainder == 0);

end

% Example usage:

message = '110100111000';

generator = '10111';

% Encode the message

[encoded\_message, ~] = crc\_encode(message, generator);

disp('Encoded Message:');

disp(num2str(encoded\_message));

% Check if the received message is valid

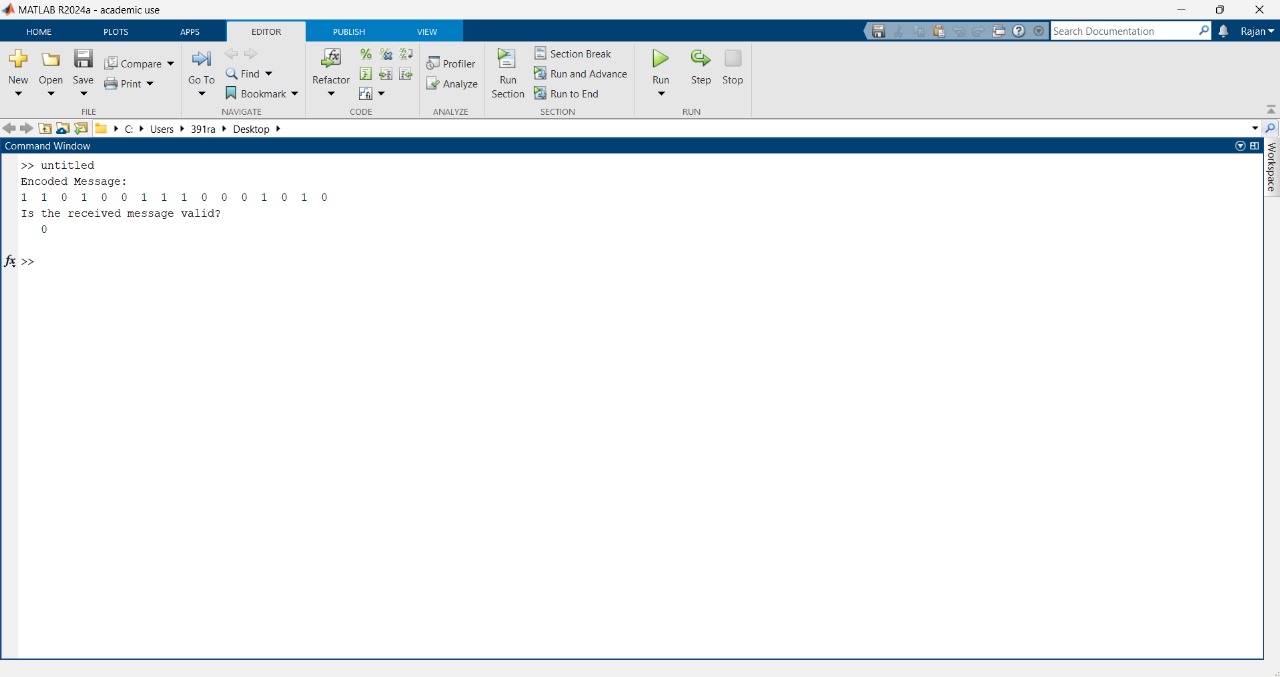
is\_valid = crc\_check(encoded\_message, generator);

disp('Is the received message valid?');

disp(is\_valid);

**CHAPTER-5**

**OUTPUT**

****

**CHAPTER-6**

**CONCLUSION**

The conclusion of Cyclic Redundancy Check (CRC) is that it is a highly effective and efficient error-detection method used to ensure data integrity in digital communications and storage systems. CRC works by generating a small checksum that can detect errors caused by accidental alterations during transmission or storage. While it cannot correct errors, its ability to quickly identify most errors with minimal computational overhead makes it essential in networking, file storage, and many other digital applications. Its simplicity and effectiveness contribute to its widespread use across various technologies.

**OUTCOME**

The outcome of using a Cyclic Redundancy Check (CRC) is the detection of errors in data transmission or storage. After applying CRC, the system can determine if the data has been corrupted during transfer by comparing the received data's checksum with the original checksum.

**If no error is detected**, the outcome is that the data is considered intact and reliable.

**If an error is detected,** the outcome is typically an error notification, prompting the system to either discard the data, request a retransmission, or take other corrective actions.

Thus, the main outcome of CRC is ensuring data integrity by quickly identifying errors.

**COURSE OUTCOME(CO)**

CO4 Is used in this.

To make student aware of various error control coding algorithm

**REFERENCE**

BY Pundir, Meena; Sandhu, Jasminder Kaur

BY GeeksForGeeks – CRC

BY P.S.V Rao and R.Chithra

BY John G. Proakis and Masoud Salehi

**GITHUB LINK**