# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune -37

*(An autonomous Institute of Savitribai Phule Pune University)*



**Department of Artificial intelligence and Data Science**

|  |  |
| --- | --- |
| **Division** | A |
| **Batch** | 1 |
| **Roll no** | 07 |
| **PRN no** | 12211542 |
| **Name** | Anish Naphade |
| **Subject** | Computer Network |

**Assignment 3**

**Problem Statement:**

Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer-to-peer mode.

**Introduction**

Cyclic Redundancy Check (CRC) is a widely used error detection and correction technique in data communication and storage systems. It is particularly effective in detecting errors caused by noise during data transmission. This assignment aims to explore the implementation of a program for error detection and correction using CRC for 7/8 bits ASCII codes.

**Cyclic Redundancy Check (CRC)**

CRC is a type of hash function that produces a fixed-size output (checksum) based on the input data. The key characteristic of CRC is its cyclic nature, meaning it treats the input data as a sequence of coefficients of a polynomial, and the division operation is performed in a cyclic manner. The resulting checksum is appended to the data for error detection and correction purposes.

**CRC Terms and Attributes**

As discussed in the previous section, CRC is performed both at the sender and the receiver side. CRC applies the CRC Generator and CRC Checker at the sender and receiver sides, respectively.

The CRC is a complex algorithm derived from the CHECKSUM error detection algorithm, using the MODULO algorithm as the basis of operation. It is based on the value of polynomial coefficients in binary format for performing the calculations.

**For Example:**

x2+x+1 (polynomial equation)

Converting to binary format-

Going through the equation, we have value at the 0th position (x), value at the 1’st position (x), and the 2nd position (x2).

So, the binary value will be - [111]

Similarly for equation, [x2+1], the binary value will be, [101].

There is no value at the “x” position, so the value is [0].

**Applications of CRC**

**--Data Transmission:** CRC is commonly used in network protocols, such as Ethernet, to ensure the integrity of data during transmission. It helps identify and correct errors that may occur due to noise in the communication channel.

**--Storage Systems:** CRC is employed in storage systems, such as hard drives and flash memory, to verify the integrity of stored data. It aids in preventing data corruption and ensuring the reliability of stored information.

**--File Transfer:** Many file transfer protocols, including the widely used File Transfer Protocol (FTP), use CRC to verify the integrity of files during the transfer process.

**Pros of Using CRC for Error Detection and Correction**

**--Efficiency:** CRC is computationally efficient, making it suitable for real-time applications and systems with limited processing resources.

**--Versatility:** It can be adapted to various data sizes and applications, providing flexibility in implementation.

**--High Error Detection Capability:** CRC is effective in detecting a wide range of errors, including single-bit errors, burst errors, and certain types of multiple-bit errors.

**--Simple Implementation:** Implementing CRC is relatively straightforward, making it a practical choice for a wide range of applications.

**Cons of Using CRC for Error Detection and Correction**

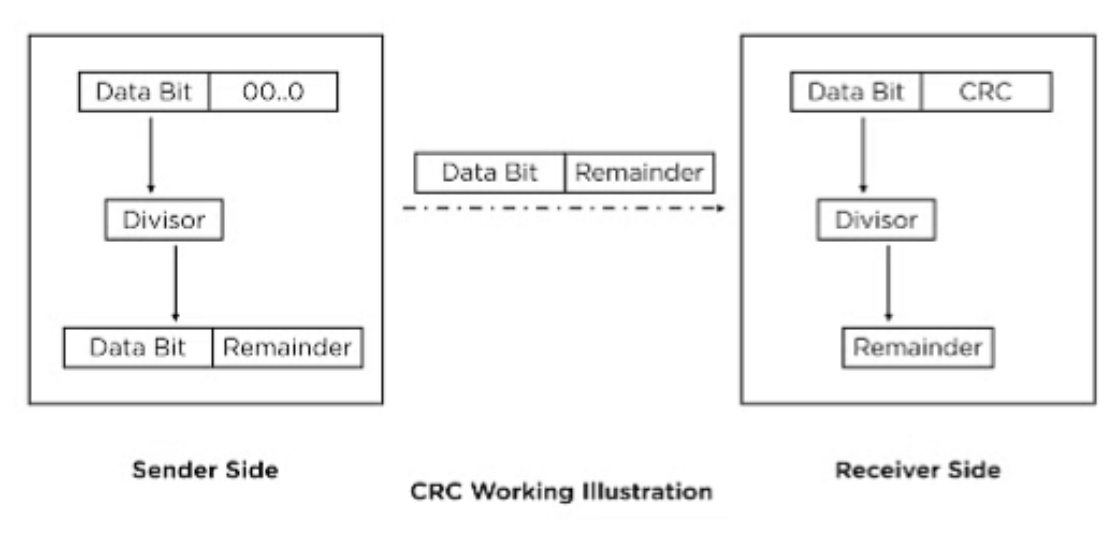
**--Limited Error Correction:** While CRC is excellent for error detection, it has limited error correction capabilities. It can identify errors but cannot always correct them, depending on the severity of the corruption.

**--Sensitivity to Polynomial Choice:** The effectiveness of CRC depends on the choice of the polynomial used in the algorithm. Selecting an inappropriate polynomial may result in reduced error detection capability.

**--Not Suitable for All Types of Errors:** CRC may struggle to detect certain patterns of errors, especially if they do not align with the characteristics of the polynomial chosen.

**--Overhead:** The addition of the CRC checksum increases the size of the transmitted or stored data, leading to a slight overhead.

**Working of CRC**



* **CODE:**

package com.main.crc.crc;

import java.util.Scanner;

public class CRC {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input dividend and divisor

System.out.print("Enter the 7/8 bits ASCII code (dividend): ");

String dividend = scanner.nextLine();

System.out.print("Enter the divisor: ");

String divisor = scanner.nextLine();

// Perform CRC encoding

String encodedData = encodeData(dividend, divisor);

System.out.println("Encoded Data (with CRC): " + encodedData);

// Simulate transmission by introducing errors

System.out.print("Enter the received data: ");

String receivedData = scanner.nextLine();

// Perform CRC decoding

boolean isErrorDetected = decodeData(receivedData, divisor);

if (isErrorDetected) {

System.out.println("Error detected in received data.");

} else {

System.out.println("No errors detected in received data.");

}

scanner.close();

}

// CRC Encoding

private static String encodeData(String data, String divisor) {

int dataLength = data.length();

int divisorLength = divisor.length();

StringBuilder encodedData = new StringBuilder(data);

// Append zeros to the dividend (data) for padding

for (int i = 0; i < divisorLength - 1; i++) {

encodedData.append('0');

}

for (int i = 0; i < dataLength; i++) {

if (encodedData.charAt(i) == '1') {

// XOR operation

for (int j = 0; j < divisorLength; j++) {

encodedData.setCharAt(i + j, xor(encodedData.charAt(i + j), divisor.charAt(j)));

}

}

}

return encodedData.toString();

}

// CRC Decoding

private static boolean decodeData(String receivedData, String divisor) {

int dataLength = receivedData.length();

int divisorLength = divisor.length();

StringBuilder receivedDataBuilder = new StringBuilder(receivedData);

for (int i = 0; i <= dataLength - divisorLength; i++) {

if (receivedDataBuilder.charAt(i) == '1') {

// XOR operation

for (int j = 0; j < divisorLength; j++) {

receivedDataBuilder.setCharAt(i + j, xor(receivedDataBuilder.charAt(i + j), divisor.charAt(j)));

}

}

}

// Check for remaining errors

for (int i = dataLength - divisorLength + 1; i < dataLength; i++) {

if (receivedDataBuilder.charAt(i) == '1') {

return true; // Error detected

}

}

return false;

// No errors detected

}

// XOR operation

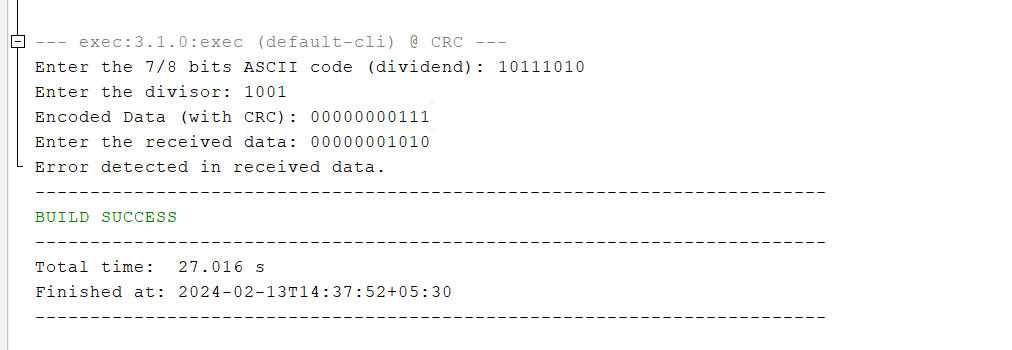
private static char xor(char a, char b) {

return (a == b) ? '0' : '1';

}

}

* **OUTPUT:**

****

* **Conclusion**

In conclusion, implementing a program for error detection and correction using CRC for 7/8 bits ASCII codes is a practical approach to enhance the reliability of data communication and storage systems. Understanding the strengths and limitations of CRC is crucial for making informed decisions when choosing error detection and correction techniques in different applications.