Department of
Computer Science and Engineering

University of Dhaka

Title: Implementation of CRC (Cyclic Redundancy Check) for Error Detection

CSE 2213: Data and Telecommunication Lab

Batch: 29/2nd Year 2nd Semester 2024

Course Instructors

Dr. Md. Mustafizur Rahman (MMR)
Mr. Palash Roy (PR)

1 Objective(s)

- To understand the theory and operation of CRC (Cyclic Redundancy Check).
- To implement a CRC encoding and decoding mechanism using polynomial division.
- To simulate data transmission and detect errors using CRC, and evaluate CRC's effectiveness in ensuring data integrity.

2 Background Theory

Cyclic Redundancy Check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. CRC uses a **Generator Polynomial**, which is available on both the sender and receiver sides. It treats the data as a binary polynomial and divides it by a generator polynomial. The remainder of this division is called the **CRC code**, which is appended to the data before transmission. CRC can detect single-bit, burst, and multi-bit errors. It is more reliable than parity and checksum methods. Therefore, it is used in Ethernet, storage devices (like HDDs), and communication protocols (e.g., HDLC, CAN bus).

An example generator polynomial is $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^8 + x^2 + x + 1$ that represents key 100000111. There are two primary variables in CRC:
n: Number of bits in data to be sent from the sender side
k: Number of bits in the key obtained from the generator polynomial.

2.1 How CRC Works:

2.1.1 Encoded Data Generation from Generator Polynomial (Sender Side)

- Appends $(k-1)$ zeros to the data (where k is the length of the generator polynomial).
- Performs binary division using XOR with the generator.
- Appends the remainder to the original data — this becomes the transmitted frame.

2.1.2 Checking Error in Transmission (Receiver Side)

- Receives the full frame (original data + CRC).
- Performs the same division.
- If the remainder is zero, the data is assumed to be correct.
- If non-zero, an error is detected.

2.2 Example Scenario

Two example scenarios are depicted as follows

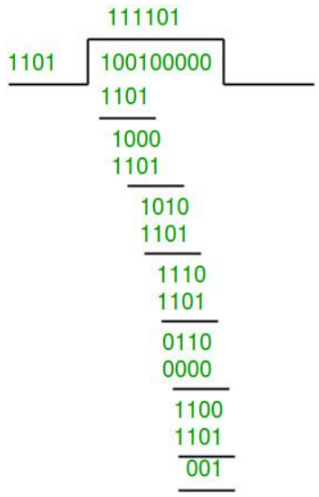
2.3 Case 1: No Error in Transmission

Fig. 1 illustrates scenario of sending Data = 100100, with Generator Polynomial (Key) = $x^3 + x^2 + 1(1101)$. In Fig. 1a, we can see that the generator polynomial (a binary string, e.g., "1101") works as the divisor and data + $(k-1)$ zeros, i.e., "100100000" works as the dividend. The remainder is 001. Thus, the data sent to the receiver is data + reminder, i.e., 100100001.

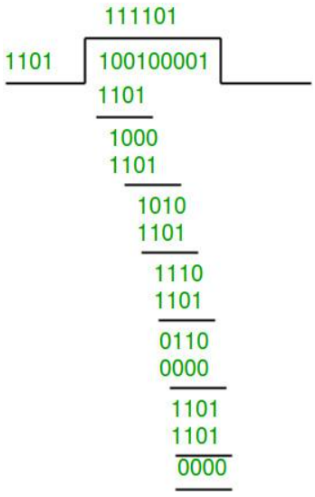
Now, from Fig. 1b, we can see that the Code word received at the receiver side is 100100001. As the remainder is 0, the data obtained has no errors.

2.4 Case 2: Error in Transmission

Fig. 2 illustrates scenario of sending same Data = 100100, with same Generator Polynomial (Key) = $x^3 + x^2 + 1(1101)$. However, there is an error, and the code word received on the receiver side is 100000001 instead of 100100001. As the remainder is not 0 (as illustrated in Fig. 2b), there is some error detected on the receiver side.

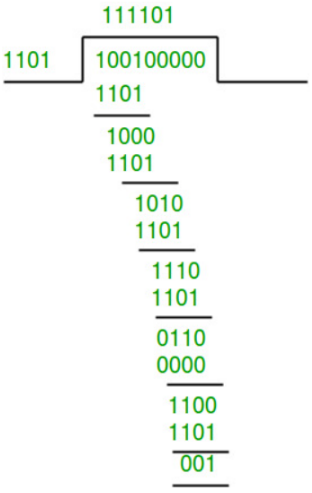


(a) Sender Side

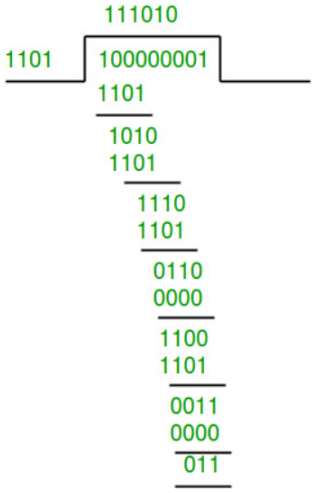


(b) Receiver Side

Figure 1: Scenario of Sending Data = 100100 Considering No Error During Transmission



(a) Sender Side



(b) Receiver Side

Figure 2: Scenario of Sending Data = 100100 Considering Random Error During Transmission

3 Experiment Overview

In this lab, we have to simulate a communication system where a client (sender) encodes binary data using CRC and transmits it to a server (receiver). The server checks for transmission errors by recalculating the CRC and comparing it with the received CRC. The main requirements of this lab are as follows:

- Implement CRC encoding at the sender side using a user-defined generator polynomial.
- Simulate transmission of data and CRC from sender to receiver over a socket connection.
- Simulate bit-level transmission errors by introducing random bit flips.
- At the receiver side, re-calculate CRC on the received data and compare the calculated and received CRC values.
- Display intermediate and final results (data, polynomial, codeword, and validation status).

3.1 Sender Side (Client) Algorithm for Encoding

- Create a Socket object that takes the IP address and port number as input.

- Create an object of the DataOutputStream class, which is used to send data to the server side.
- Open and read content from a file (e.g., input.txt). The text file (input.txt) contains with ASCII character data (e.g., "HELLO").
- Convert the character string to a binary representation (ASCII to 8-bit binary).
- Input the generator polynomial (a binary string, e.g., "10011").
- Append $(k-1)$ zeros to the original data.
- Perform modulo-2 division on the appended data using the generator polynomial (XOR-based bitwise division).
- Obtain the remainder, which is the CRC code.
- Append the CRC code to the original data to form the codeword.
- Send the codeword (data + CRC) to the receiver through the socket.
- Close the socket and terminate the program.

3.2 Server Side Algorithm for Decoding

- Create a ServerSocket object, namely handshaking socket, which takes a port number as input.
- Create a plain Socket object that accepts client requests
- Create an object of the DataInputStream class, which is used to read data
- Read the received codeword (data + CRC) from the input stream.
- Input or define the generator polynomial (same as the sender).
- Perform modulo-2 division on the received codeword using the generator polynomial.
- Check the remainder:
  - If the remainder is all 0s, assume no error.
  - If the remainder contains 1s, declare error detected.
- Optionally —Introduce a function to flip random bits in the received codeword before verification (simulating transmission errors).
- Close connection and terminate server after processing is complete.

4 Input/Output

Sample Input and Output of the above experiment can be as follows

Client Side

```
Client connected to the server on Handshaking port 5000
Client's Communication Port: 56190
Client is Connected
File Content: HELLO
Converted Binary Data: 01001000010001010100110001001100010011111
After Appending zeros Data to Divide: 010010000100010101001100010011000100111110000
CRC Remainder: 1011
Transmitted Codeword to Server: 010010000100010101001100010011000100111111011
```

Server Output without Error Simulation

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
Client request is accepted at port no: 56190
Server's Communication Port: 5000
Received Codeword: 0100100001000101010011000100110001001111011
Calculated Remainder: 0000
No error detected in transmission.
```

Server Output with Error Simulation (Bit flipped)

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
Client request is accepted at port no: 56190
Server's Communication Port: 5000
Received Codeword: 0100100001000101010011000100110001001011011
Calculated Remainder: 1101
Error detected in transmission!
```

5 Discussion & Conclusion

Based on the focused objective(s), this task will help us to demonstrate how single-level time division multiplexing can combine multiple data streams and reconstruct them accurately, even with uneven data lengths.

6 Policy

Copying from the Internet, classmates, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.