BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA, BENGALURU - 560064

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Report on Cyclic Redundancy Check (CRC)

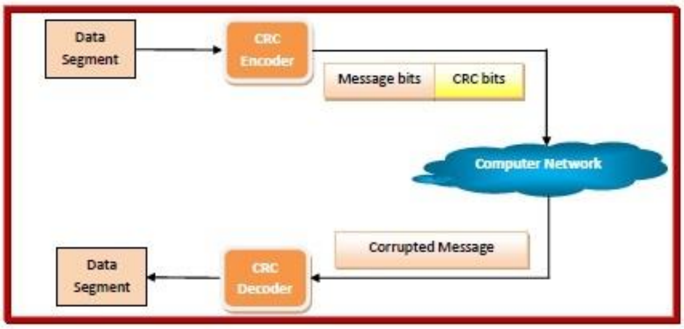
Name	Raghavendra K M
USN	1BY18IS093
Course Code	18CSL57
Course Name	Computer Network Laboratory
Faculty	Prof. Gireesh babu C N
Title	Cyclic Redundancy Check
Date	14-09-2020

Signature of a Student **Signature of a Faculty**

CYCLIC REDUNDANCY CHECK

A **Cyclic Redundancy Check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction .

CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyse mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.



The process of CRC

Computation of CRC

When messages are encoded using CRC (polynomial code), a fixed polynomial called generator polynomial, G(x) is used. The value of is mutually agreed upon by the sending and the receiving parties. A k – bit word is represented by a polynomial which ranges from X^0 to x^{k-1}. The order of this polynomial is the power of the highest coefficient, i.e.(K-1) The length of G(x) should be less than the length of the message it encodes. Also, both its MSB (most significant bit) and LSB (least significant bit) should be 1. In the process of encoding, CRC bits are appended to the message so that the resultant frame is divisible by G(x).

- **Algorithm for Encoding using CRC**
  - The communicating parties agrees upon the size of message, M(x) and the generator polynomial, G(x).

- If r is the order of G(x), r, bits are appended to the low order end of M(x). This makes the block size bits, the value of which is $x^r M(x)$.
- The block $x^r M(x)$ is divided by G(x) using modulo 2 division.
- The remainder after division is added to $x^r M(x)$ using modulo 2 addition. The result is the frame to be transmitted, T(x). The encoding procedure makes exactly divisible by G(x).

- **Algorithm for Decoding using CRC**
  - The receiver divides the incoming data frame T(x) unit by G(x) using modulo 2 division. Mathematically, if E(x) is the error, then modulo 2 division of [M(x) + E(x)] by G(x) is done.
  - If there is no remainder, then it implies that E(x). The data frame is accepted.
  - A remainder indicates a non-zero value of E(x), or in other words presence of an error. So, the data frame is rejected. The receiver may then send an erroneous acknowledgment back to the sender for retransmission.

Figure : CRC encoder and decoder

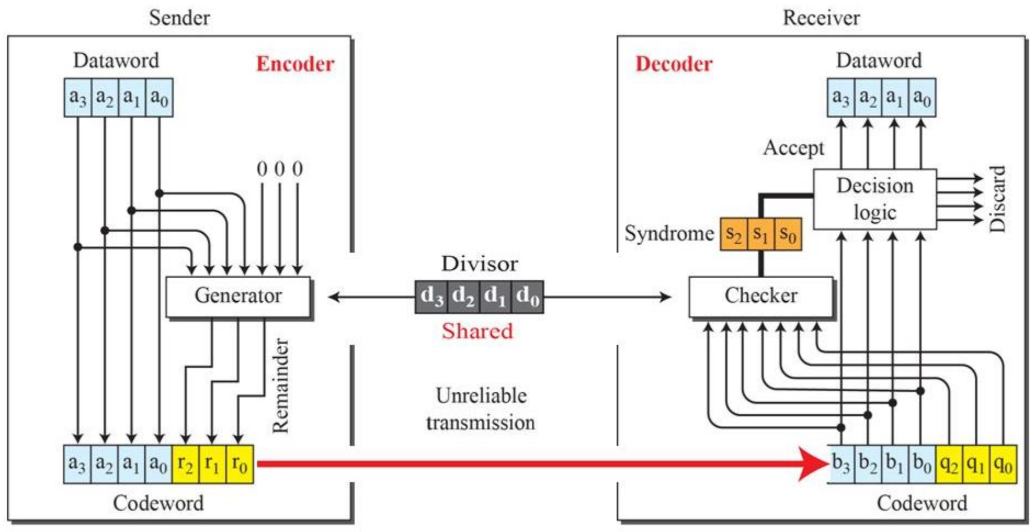
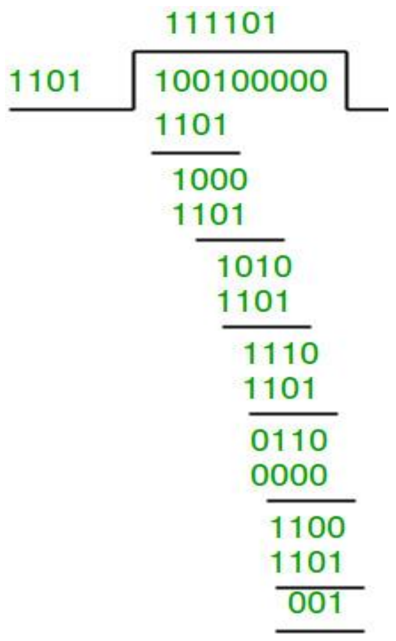


Illustration:

Example1: No error in transmission.

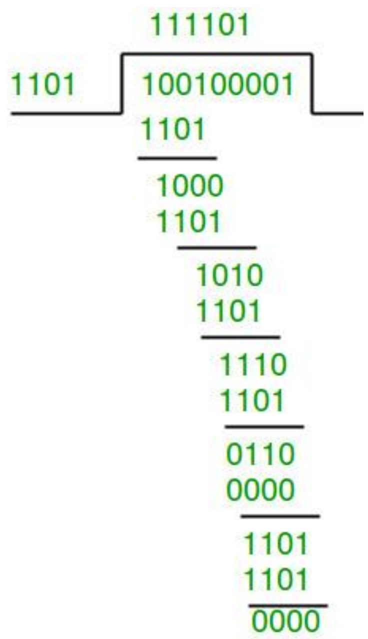
Data word to be sent: 100100
Key: 1101 [Or generator polynomial $x^3 + x^2 + 1$]

Sender side:



Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver Side:



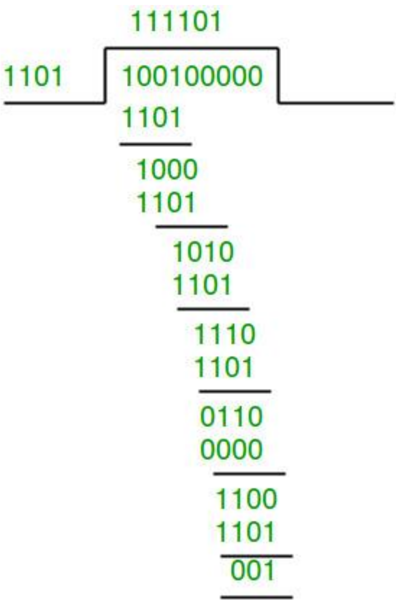
Code word received at the receiver side 100100001

Therefore, the remainder is all zeros. Hence, the data received has no error.

Example 2: (Error in transmission)

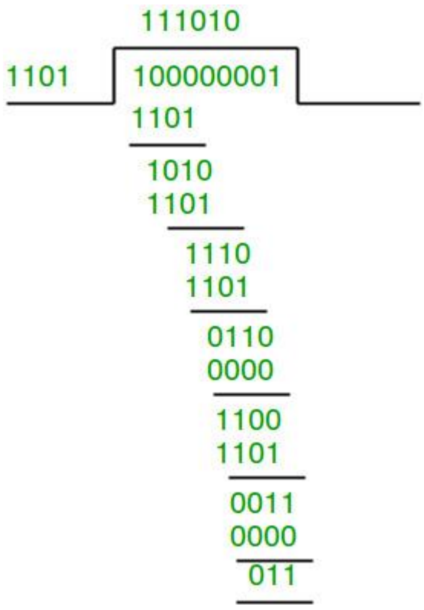
Data word to be sent – 100100
Key – 1101

Sender Side:



Therefore, the remainder is 001 and hence the code word sent is 100100001.

Receiver Side:



Let there be an error in transmission media Code word received at the receiver side – 100000001

Since the remainder is not all zeroes, the error is detected at the receiver side.

Write a program for error detecting code using CRC-CCITT (16- bits).

Source code:

```
import java.util.*;

public class CRC {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m, g[], n, d[], z[], r[], msb, i, j, k;
        //m->size of the generator
        //g[]-> Generator or divisor array
        //n-> size of the dataword
        //d[]-> Array for dataword
        //z[]-> Zero Array
        //r[]-> Array for remainder
        //msb-> most significant bit
        //i, j, k->Array variables
        System.out.println("Enter no. of bits in Dataword(d): ");
        n = scanner.nextInt();

        System.out.println("Enter no. of generator bits: ");
        m = scanner.nextInt();

        d = new int[n + m];
        g = new int[m];

        System.out.println("Enter Dataword bits(one bit per line): ");
        for (i = 0; i < n; i++)
            d[i] = scanner.nextInt();

        System.out.println("Enter generator bits(one bit per line): ");
        for (j = 0; j < m; j++)
            g[j] = scanner.nextInt();
        //d[n+i]->Augmented dataword Array
        for (i = 0; i < m - 1; i++)
            d[n + i] = 0; //Appending (m-1) 0s
```

```
r = new int[m + n];
for (i = 0; i < m; i++)
    r[i] = d[i];//initial data to be divided(First part of dividend)

z = new int[m];
for (i = 0; i < m; i++)
    z[i] = 0; // XORed when msb=0

for (i = 0; i < n; i++) {
    k = 0; // Number of iterations or division steps
    msb = r[i];
    for (j = i; j < m + i; j++) {
        if (msb == 0)
            r[j] = xor(r[j], z[k]);
        else
            r[j] = xor(r[j], g[k]);
        k++;
    }
    r[m + i] = d[m + i]; // Final remainder
}

System.out.println("The redundant(r) bits added are: ");
for (i = n; i < n + m - 1; i++)   // i started with n value
{
    d[i] = r[i];
    System.out.println(d[i]);
}

System.out.println();

System.out.println("The codeword (c=d+r) is :");
for (i = 0; i < n + m - 1; i++)  // i started with 0 value
{
    System.out.println(d[i]);
}

int c[] = new int[n + m];
System.out.println();
```

```java
        System.out.println("Enter the data bits received (one bit per line):");

        for (i = 0; i < n + (m - 1); i++)
            c[i] = scanner.nextInt();
        int count = 0;    //to print error at particular position
        for (i = 0; i < n + m - 1; i++) {
            if (c[i] == d[i])
                count++;
            else
                break;
        }

        if (count == n + m - 1)
            System.out.println("No error");
        else {
            System.out.println("Error present");
            System.out.println("Error occurs at " + (count + 1));
        }
    }

    public static int xor(int x, int y) // xor function
    {
        if (x == y)
            return (0);
        else
            return (1);
    }
}
```

Output:

Case 1: Without error

```
C:\Windows\System32\cmd.exe - java  CRC

I:\Documents (C)\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\CRC report\CNS Lab programs>java CRC
Enter no. of bits in Dataword(d) :
5
Enter no. of generator bits :
17
Enter Dataword bits(one bit per line):
1
1
0
0
1
Enter generator bits(one bit per line):
1
0
0
1
1
0
0
0
0
1
1
1
0
0
1
1
0

The redundant(r) bits added are :
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0

The codeword(c=d+r) is :
1
1
0
0
1
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0

1
Enter the data bits recieived(one bit per line)
1
1
0
0
1
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0
No error

I:\Documents (C)\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\CRC report\CNS Lab programs>
```

Sender side:

No. of dataword = 5

No. of bits of generator word = 17

Data word = 11001

Generator word = 10011000011100110

Total data word = 11001000000000000000000

```
                    11011
10011000011100110 | 110010000 0000 0000 0000 0000
                    10011000011100110
                    01010000011100 1100
                     10011000011100110
                      00111 0001001 0 101000
                        10011000011100110
                          11110100010011100
                          10011000011100110
                           11011000011111010
```

Codeword = dataword + remainder , message to be transmitted.

```
  110010000 0000 0000 0000
          11011000011111010
  ------------------------
  1100111011000011111010
```

Receiver side:

Data word = 1100111011000011111010

Generator word = 10011000011100110

```
                    11011
10011000011100110 | 1100111011000011111010
                    10011000011100110
                     1010110101100011
                     10011000011100110
                        110101000100101 01
                        10011000011100110
                         0100110000111001 10
                          10011000011100110
                          0000000000000000
```

Since, the remainder is zero, data received has no error.

Case 2: With error

```
C:\Windows\System32\cmd.exe

I:\Documents (C)\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\CRC report\CNS Lab programs>java CRC
Enter no. of bits in Dataword(d) :
5
Enter no. of generator bits :
17
Enter Dataword bits(one bit per line):
1
1
0
0
1
Enter generator bits(one bit per line):
1
0
0
1
1
0
0
0
0
1
1
1
0
0
1
1
0
The redundant(r) bits added are :
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0
The codeword(c=d+r) is :
1
1
0
0
1
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0

Enter the data bits recieived(one bit per line)
1
0
0
1
1
1
1
0
1
1
0
0
0
0
1
1
1
1
0
1
0
Error present
Error occurs at 2

I:\Documents (C)\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\CRC report\CNS Lab programs>
```

Sender Side:

No. of data word = 5
No. of bits of generator word = 17
Data word = 11001
Generator word = 10011000011100110
Total data word = 110010000000000000000

```
                       11011
                    ______________________
10011000011100110 | 110010000000000000000
                    10011000011100110
                    ---------------------
                     10100000111001100
                     10011000011100110
                    ---------------------
                       1110001001010100 0
                       10011000011100110
                    ---------------------
                       01110100010011100
                        10011000011100110
                    ---------------------
                         1101100001111010
```

Code word = dataword + remainder , message to be transmitted

```
  110010000000000000000
      1101100001111010
  ---------------------
  110011101100001111010
```

Receiver side:

Received bit = 100111011000011111010

Gen word = 10011000011100110

```
                      10000
                    _______________________
10011000011100110 | 100111011000011111010
                    10011000011100110
                    ---------------------
                    000001101011000011010
                     0000000000000000000
                    ---------------------
                       1101011000011010
```

Since, the remainder contains non zero bits, the data received has an error.