PARSHWANATH CHARITABLE TRUST'S

A.P. SHAH INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Data Science

Semester: V **Subject:** Computer Network **Academic Year:** 2023-24

Module -2

Error Detection and Correction (Hamming Code, CRC, Checksum)

Network designers have developed two basics strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been .The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses **Error – correcting codes** and the latter uses **Error- detecting codes**.

The **Error – correcting and Error- detecting methods are**

1. PARITY METHOD
2. LRC METHOD (Longitudinal redundancy check)
3. CRC METHOD (Cyclic redundancy check)
4. HAMMING CODE METHOD

PARITY METHOD

- appends a parity bit to the end of each word in the frame

- Even parity is used for asynchronous Transmission

- Odd parity is used for synchronous Transmission

Ex	Character code	even parity	odd parity
1.	1100100	1100100 1	1100100 0
2.	0011000	0011000 0	0011000 1

If one bit or any odd no bits is erroneously inverted during Transmission, the Receiver will detect an error. How ever if two or even no of bits are inverted an undetected error occurs.

Ex 3. The Transmitted data is 10011010. The received data is 11011010.

Let both the transmitter and receiver are agreed on EVEN parity.

Now an error will be detected, since the no of ones received are ODD

4. The Transmitted data is 10011010. The received data is 01011010

The received data is wrong even though the no of ones are EVEN.

Science two bits are inverted error can't be detected.

Longitudinal Redundancy Check(LRC)

The frame is viewed as a block of characters arranged in 2-dimensions. To each character is appended a parity bit. In addition a parity bit is generated for each bit position across all characters i.e., an additional character is generated in which the Ith bit of the character is parity bit for the Ith bit of all other characters in the block. This can be expressed mathematically using exclusive OR(+) operation. The parity bit at the end of each character of row parity

$$R_j = b_{1j} + b_{2j} + \text{------} b_{nj}$$

Where R_j=Parity bit of jth character

b_{ij}=ith bit in jth character

This equation generates even parity.

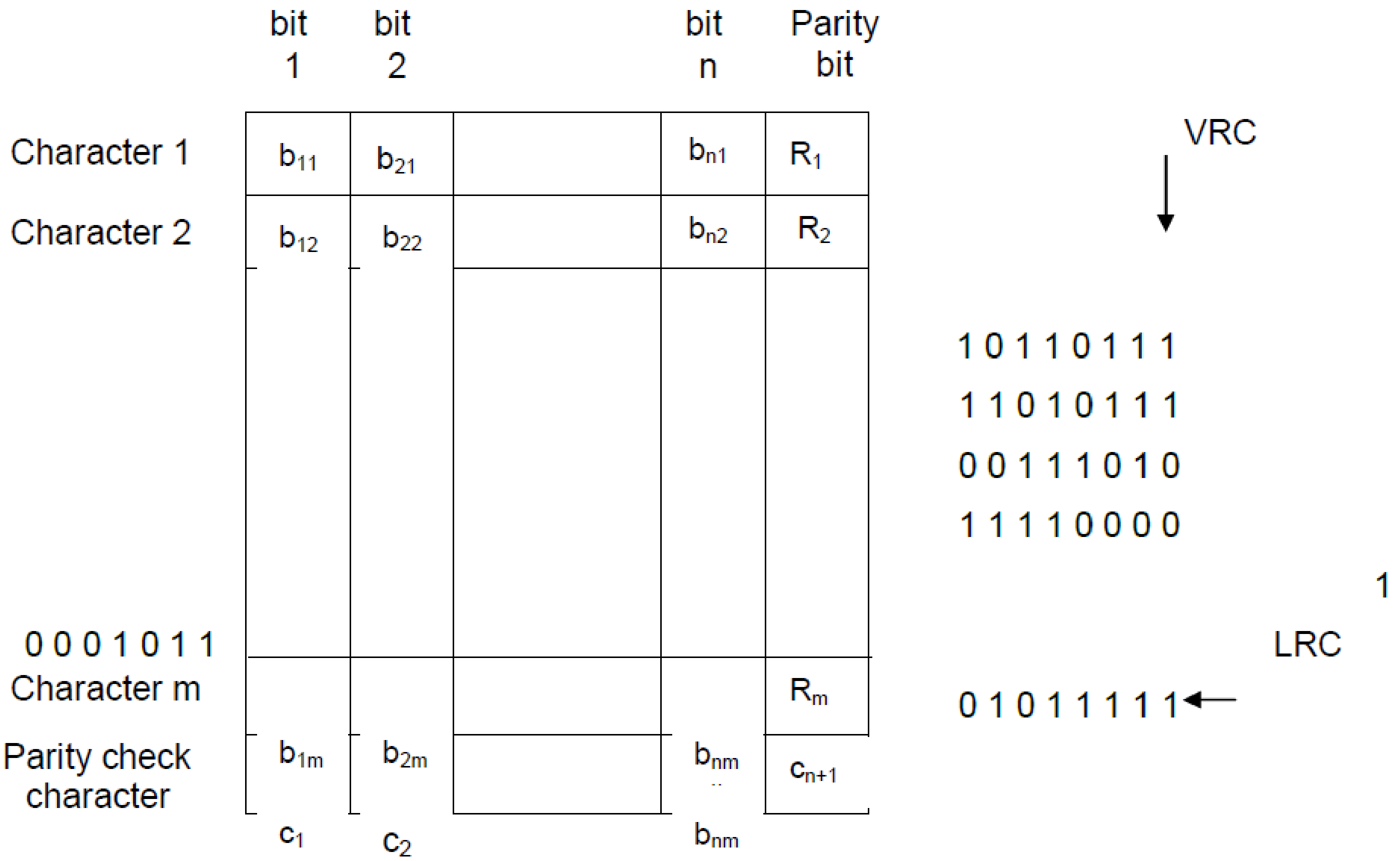
$$C_i = b_{i1} + b_{i2} + \text{-----} + b_{in}$$

Where C_i=ith bit of parity check character

m=number of characters in a frame

In this format the parity bits at the end of each character are referred to as

The Vertical Redundancy Check (VRC) and the Parity check character is referred to as the Longitudinal Redundancy Check (LRC).



	bit 1	bit 2		bit n	Parity bit
Character 1	b_{11}	b_{21}		b_{n1}	R_1
Character 2	b_{12}	b_{22}		b_{n2}	R_2
0 0 0 1 0 1 1 Character m					R_m
Parity check character	b_{1m} c_1	b_{2m} c_2		b_{nm} b_{nm}	c_{n+1}

VRC

1 0 1 1 0 1 1 1
1 1 0 1 0 1 1 1
0 0 1 1 1 0 1 0
1 1 1 1 0 0 0 0

1

LRC

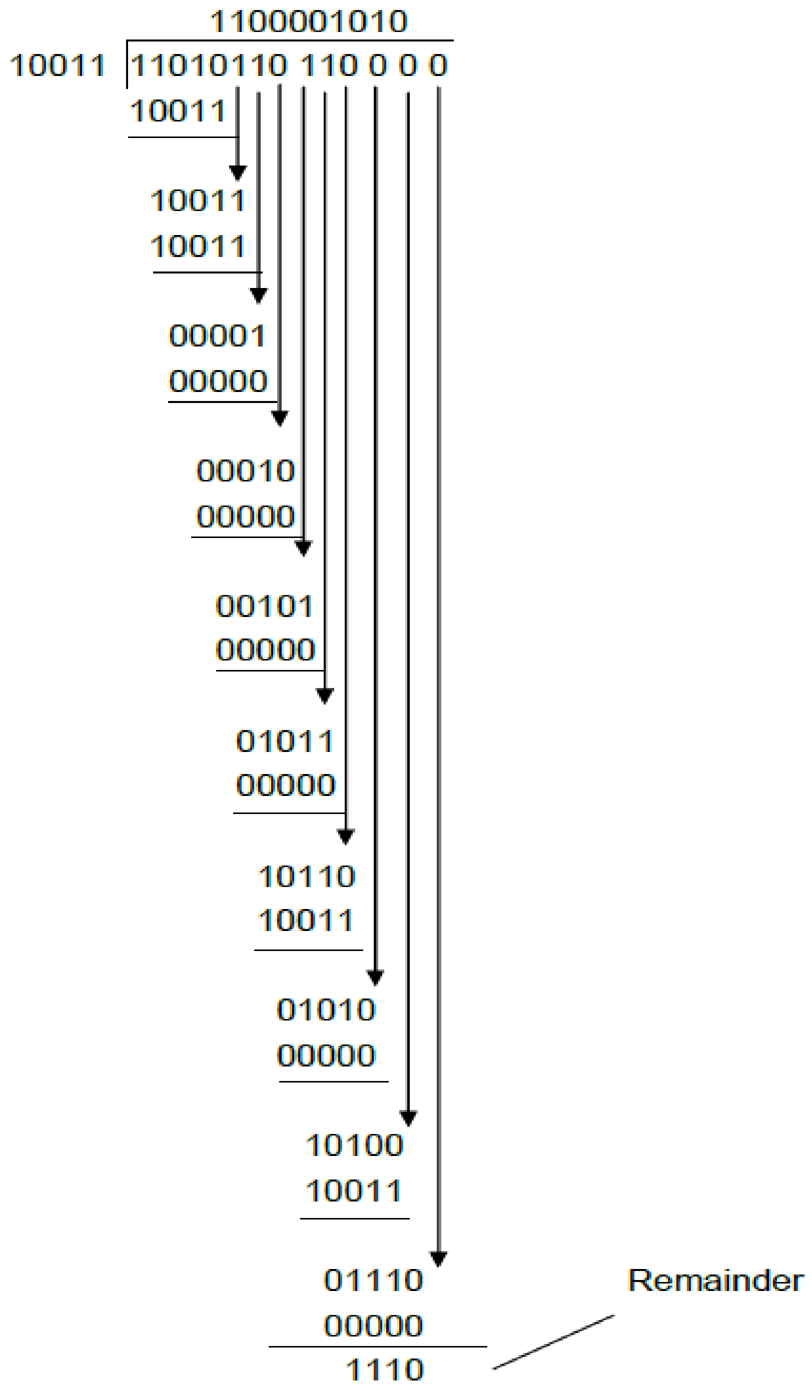
0 1 0 1 1 1 1 1

CRC Method

1. The frame is expressed in the form of a Polynomial F(x).0 1 1 1 1 1 1 0
2. Both the sender and receiver will agree upon a generator polynomial G(x) in advance.
3. Let 'r' be the degree of G(x).Append 'r' zero bits to the lower – order end of frame now it contains m+r bits.
4. Divide the bit string by G(x) using Mod 2 operation.
5. Transmitted frame [T(x)] = frame + remainder
6. Divide T(x) by G(x) at the receiver end. If the result is a zero, then the frame is transmitted correctly. Ex. Frame: 1101011011

Generator: 10011

Message after appending 4 zero bits: 11010110000



```
                1100001010
10011 | 11010110 110 0 0 0
        10011
          10011
          10011
            00001
            00000
              00010
              00000
                00101
                00000
                  01011
                  00000
                    10110
                    10011
                      01010
                      00000
                        10100
                        10011
                          01110
                          00000
                           1110        Remainder
```

Transmitted frame: 11010110111110

```
               1100001010
        10011 |11010110 111 1 1 0
               10011
               -----
                10011
                10011
                -----
                 00001
                 00000
                 -----
                  00010
                  00000
                  -----
                   00101
                   00000
                   -----
                    01011
                    00000
                    -----
                     10111
                     10011
                     -----
                      01001
                      00000
                      -----
                       10011
                       10011
                       -----
                        00000          Remainder
                        00000
                       -------
                         0000
```

Since the remainder is zero there is no error in the transmitted frame.

HAMMING CODES

Hamming codes provide another method for error correction. Error bits, called Hamming bits, are inserted into message bits at random locations. It is believed that the randomness of their locations reduces the odds that these Hamming bits themselves would be in error. This is based on a mathematical assumption that because there are so many more message bits compared with Hamming bits, there is a greater chance for a message bit to be in error than for a Hamming bit to be wrong. Determining the placement and binary value of the Hamming bits can be implemented using hardware, but it is often more practical to implement them using software. The number of bits in a message (M) are counted and used to solve the following equation to determine the number of Hamming bits (H) to be used:

$2^H \geq M + H + 1$

Once the number of Hamming bits is determined, the actual placement of the bits into the message is performed. It is important to note that despite the random nature of the Hamming bit placements, the exact sample placements must be known and used by both the transmitter and receiver. Once the Hamming bits are inserted into their positions, the numerical values of the bit positions of the logic 1 bits in the original message are listed. The equivalent binary numbers of these values are added in the same manner as used in previous error methods by discarding all carry results. The sum produced is used as the states of the Hamming bits in the message. The numerical difference between the Hamming values transmitted and that produced at the receiver indicates the bit position that contains a bad bit, which is then inverted to correct it.

Ex. The given data

10010001100101(14- bits)

The number of hamming codes

$2^H \geq M + H + 1$

H = ? M = 14 to satisfy this equation H should be 5 i.e. 5 hamming code bits should be incorporated in the data bits.

1 0 0 1 0 0 0 1 1 0 H 0 H 1 H 0 H 1 H

Now count the positions where binary 1's are present. Add using mod 2 operation (Ex-OR). The result will give the Hamming code at the transmitter end.

1's position	Binary equivalent
2 -	0 0 0 1 0
6 -	0 0 1 1 0
11 -	0 1 0 1 1
12 -	0 1 1 0 0
16 -	1 0 0 0 0
19 -	1 0 0 1 1
Hamming code =	0 0 0 0 0

This Hamming code will be incorporated at the places of 'H' in the data bits and the data will be transmitted.

How to find out there is an error in the data?

Let the receiver received the 12th bit as zero. The receiver also finds out the Hamming code in the same way as transmitter.

1's position	Binary equivalent
2 -	0 0 0 1 0
6 -	0 0 1 1 0
11 -	0 1 0 1 1
16 -	1 0 0 0 0
19 -	1 0 0 1 1
Hamming code at the receiver	0 1 1 0 0

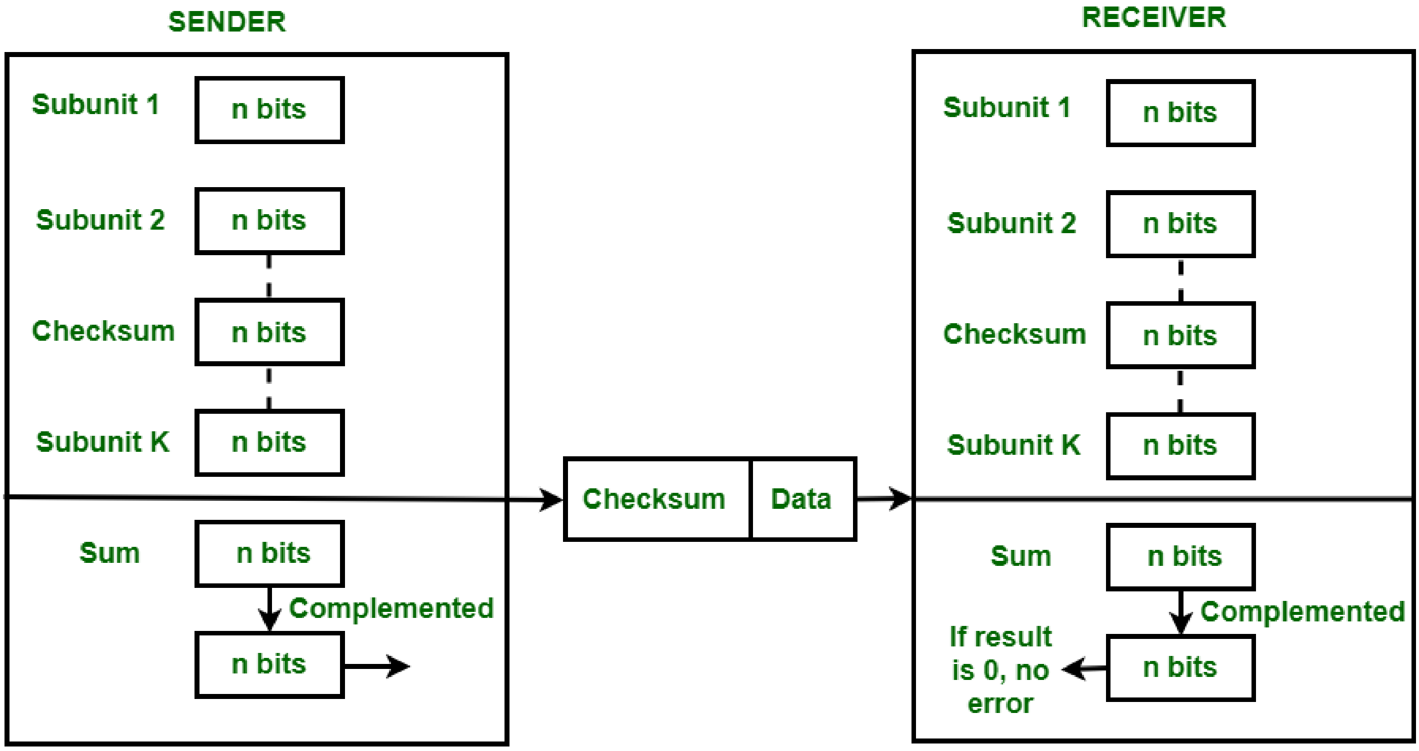
Hamming code at the Tx	0 0 0 0 0
Hamming code at the Rx	0 1 1 0 0
	0 1 1 0 0

The decimal equivalent for the binary is **12** so error is occurred at 12th place.

CHECKSUM

Checksum is the error detection method used by upper layer protocols and is considered to be more reliable than LRC, VRC and CRC. This method makes the use of **Checksum Generator** on Sender side and **Checksum Checker** on Receiver side.

At the Sender side, the data is divided into equal subunits of n bit length by the checksum generator. This bit is generally of 16-bit length. These subunits are then added together using one's complement method. This sum is of n bits. The resultant bit is then complemented. This complemented sum which is called checksum is appended to the end of original data unit and is then transmitted to Receiver.

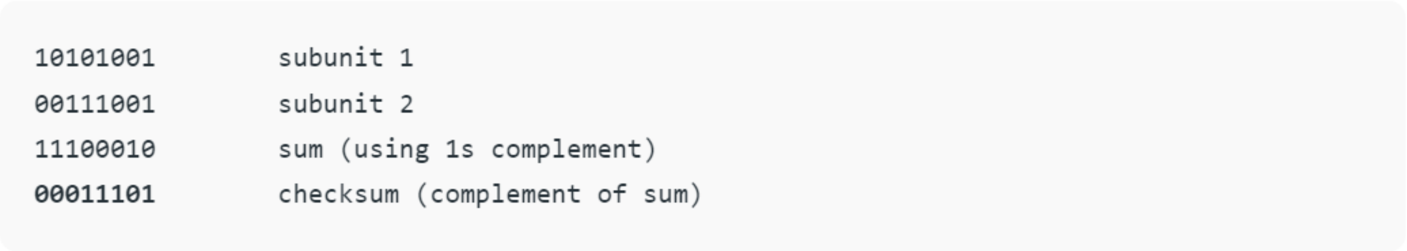


The Receiver after receiving data + checksum passes it to checksum checker. Checksum checker divides this data unit into various subunits of equal length and adds all these subunits. These subunits also contain checksum as one of the subunits. The resultant bit is then complemented. If the complemented result is zero, it means the data is error-free. If the result is non-zero it means the data contains an error and Receiver rejects it.

Example –

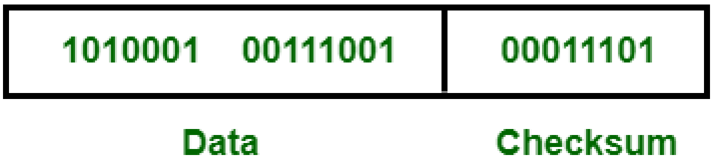
If the data unit to be transmitted is 10101001 00111001, the following procedure is used at Sender site and Receiver site.

Sender Site :

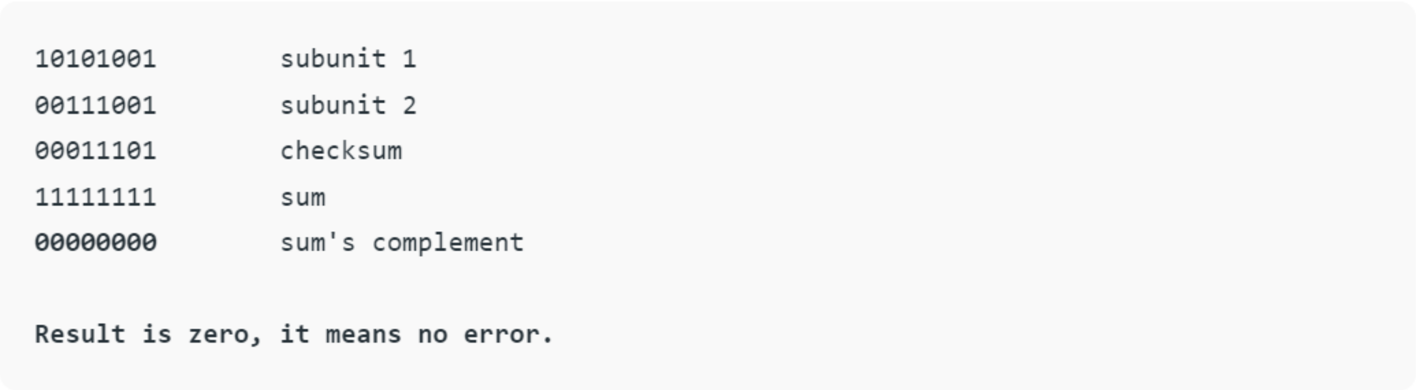


```
10101001        subunit 1
00111001        subunit 2
11100010        sum (using 1s complement)
00011101        checksum (complement of sum)
```

Data transmitted to Receiver is –



Receiver Site :



```
10101001        subunit 1
00111001        subunit 2
00011101        checksum
11111111        sum
00000000        sum's complement

Result is zero, it means no error.
```

Advantage :

The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

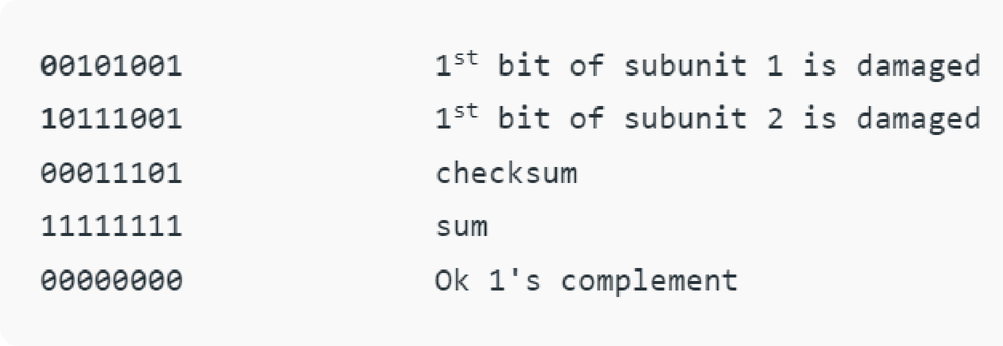
Disadvantage :

The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

Example –

If the data transmitted along with checksum is 10101001 00111001 00011101. But the data received at destination is **0**0101001 **1**0111001 00011101.

Receiver Site :



```
00101001            1st bit of subunit 1 is damaged
10111001            1st bit of subunit 2 is damaged
00011101            checksum
11111111            sum
00000000            Ok 1's complement
```

Although data is corrupted, the error is undetected.