Error Control: Detection and Correction

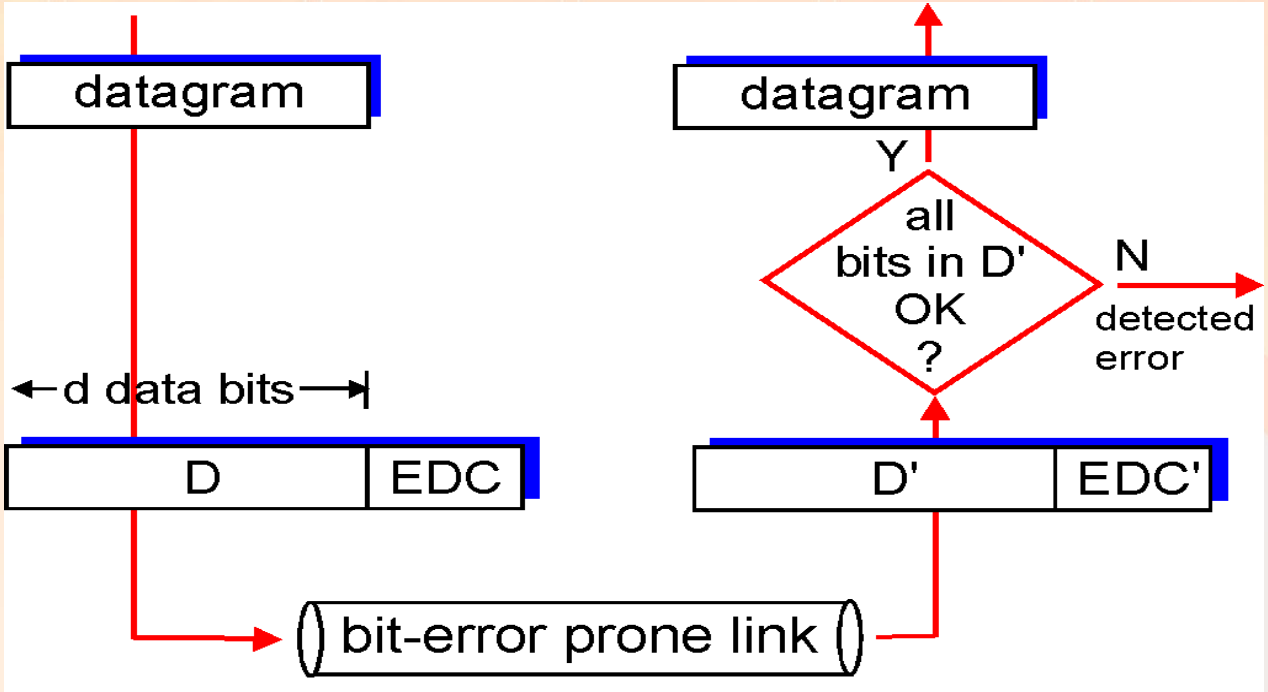
Slide Set 4

Error Detection

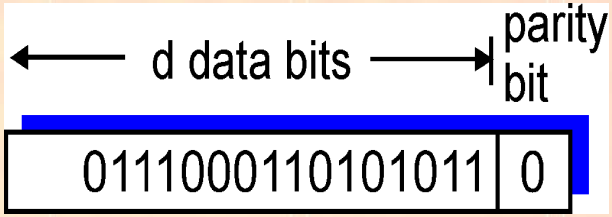
EDC = Error Detection and Correction bits (redundancy)

D = Data protected by error checking, may include header fields

- **Error detection not 100% reliable!**
  - **protocol may miss some errors, but rarely**
  - **larger EDC field yields better detection and correction**



Single Bit Parity Checking (Detect Only)



This is an example of odd parity: The parity bit is chosen (0) in such a way that the total number of 1s is odd (9)

IMPORTANT

A single bit parity check will only be able to detect 1 or and odd number of bits in error.

Even Parity Scheme:

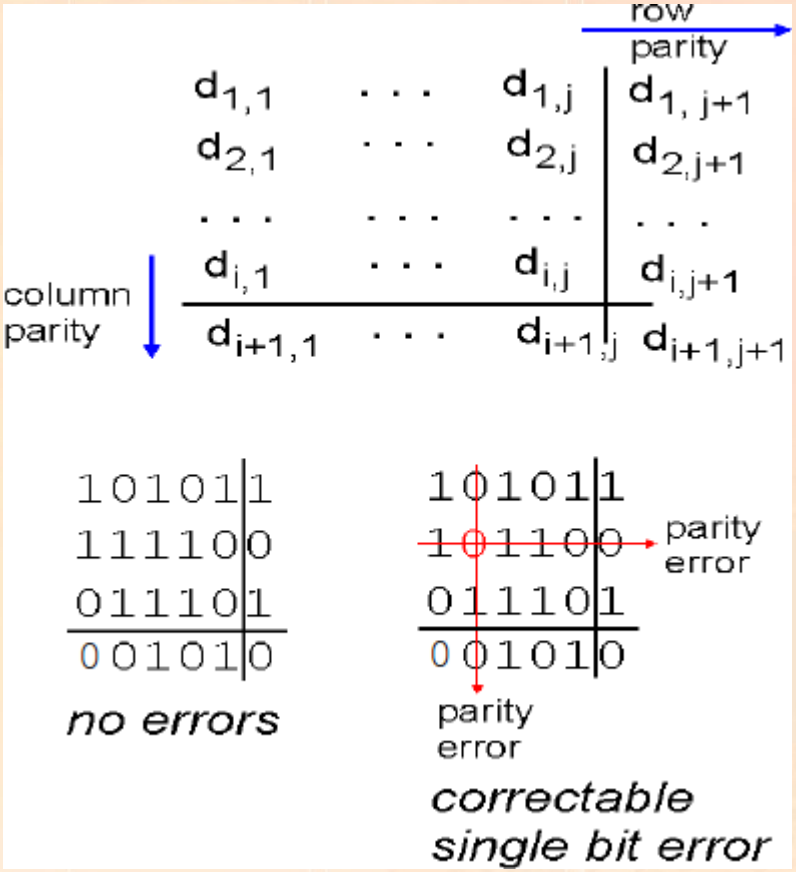
Parity bit chosen so that total number of 1s including the parity bit is **EVEN.**

Odd Parity Scheme:

Parity bit chosen so that total number of 1s including the parity bit is **ODD**.

It is highly efficient since a single parity bit is needed for any length of data bits (Message M).

2-D Bit Parity Checking (Detect & Correct)



This is an example of even parity: The parity bits are chosen in such a way that the total number of 1s is either a column or row is even

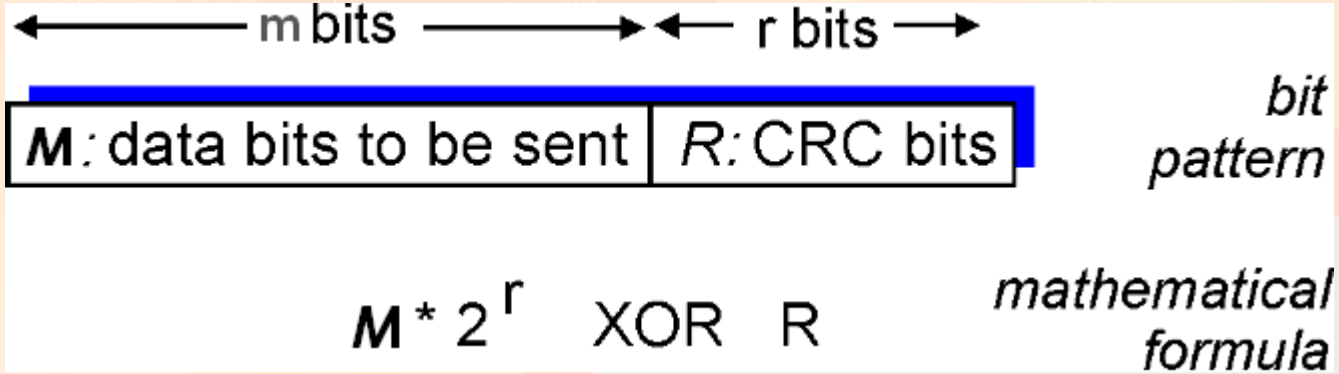
In this technique, the data bits are rearranged in an nxm matrix. Ideally a square matrix for higher efficiency. The parity bit is chosen for each row and column in the matrix. An additional parity bits can be used for checking the parity bits themselves but this is optional.

IMPORTANT

A 2-D bit parity check will only be able to detect 1 or and odd number of bits in error in either a column or a row but can also correct single bit errors if they occur in different rows and columns.

Cyclic Redundancy Check (Detect Only)

- view Message bits, M, as a binary number
- choose r+1 bit pattern divisor (generator), G
- goal: choose r CRC bits, R, such that
  - <M,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <M,R> by G. If non-zero remainder occurs: error detected!



Example of CRC in an Ethernet frame

CRC appends **redundant** bits to the frame trailer called Frame Check Sequence (FCS)

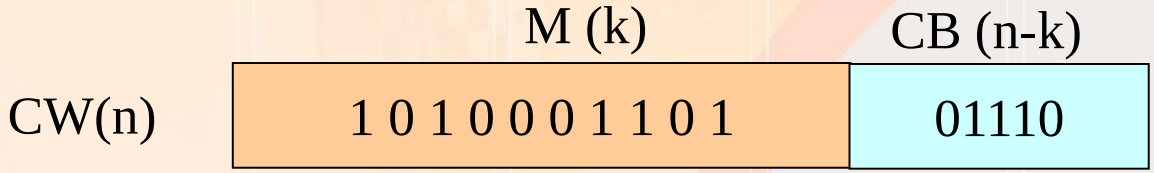
The FCS bits are used at Receiver for error detection

In a given frame containing a total of n bits, we define:

k = the number of original data bits (Message M)

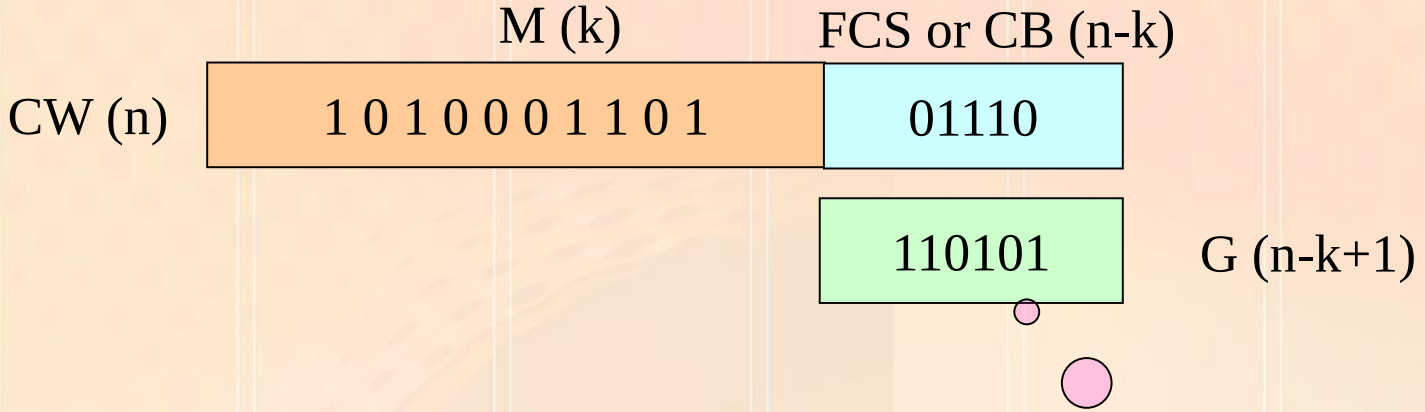
(n – k) = the number of added bits as the FCS field or Code Bits (CB)

So, the total frame length is k + (n – k) = n bits or Code Word (CW)



CRC Generation

CRC generation at the sender is all about finding the FCS, given the data (M) and a divisor (G) that makes CW exactly divisible by G (i.e. with 0 remainder)



There are three equivalent ways to see how the CRC code is generated:

Modulo-2 Arithmetic Method

Polynomial Method (not covered)

Digital Logic Method (not covered)

What is F that makes T divide P exactly? i.e. with no remainder

Modulo-2 Arithmetic

- *In modulo 2 arithmetic addition and substruction are identical to EXCLUSUVE OR (XOR) operation.*
- *Multiplication and division are the same as in base-2 arithmetic without carries in addition or borrows in substraction.*

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Examples:

1011 XOR 0101 = 1110

1001 XOR 1101 = 0100

CRC Error Detection Process

Given k-bit data (M), the Sender generates an (n – k)-bit FCS field (CB) such that the *total* n-bit frame (CW) is exactly divisible by a predefined (n-k+1) bit divisor (G) (i.e. gives a zero remainder)

In general, the received frame (CW') may or may not be identical to the sent frame (CW).

Let the received frame be (CW’)

Only in error-free transmissions that we have CW’ = CW

Receiver divides (CW’) by the same known divisor (G) and checks if there is any remainder, if division yields a remainder then the frame is erroneous

If the division yields zero remainder then the frame is error-free unless many erroneous bits in CW’ resulted in a new exact division by G.

This is extremely unlikely but possible, causing an undetected error!

Example – Modulo-2 Arithmetic Method

- Given
  - M = 1 0 1 0 0 0 1 1 0 1
  - G = 1 1 0 1 0 1 (i.e. $x^5+x^4+x^2+1$)

At the Sender (source) side

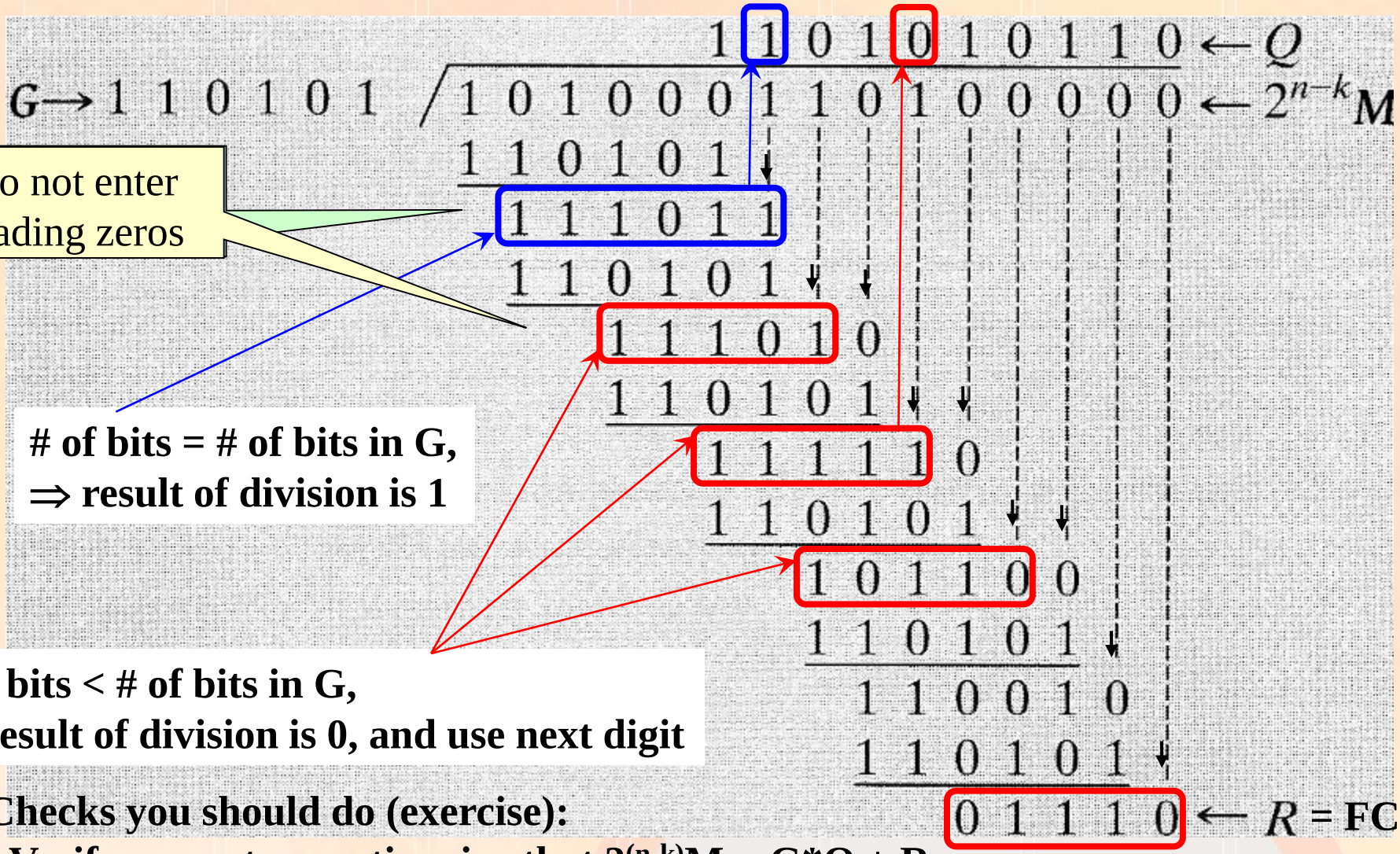
- Find the FCS field
- Solution:
  - First we note that:
    - The size of the data block M is k = 10 bits
    - The size of G is (n – k + 1) = 6 bits
      - Hence the FCS length is n – k = 5
      - Total size of the frame CW is n = 15 bits

Example – Modulo-2 Arithmetic Method

- Solution (continued):
  - Multiply $2^{(n-k)}$ x M
    - $2^{(5)}$ x 1 0 1 0 0 0 1 1 0 1 = 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0
    - This is a simple shift to the left by five positions and inserting (n-k) zeroes.
  - Divide $2^{(n-k)}$ x M / G (see next slide for details)
    - 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 ÷ 1 1 0 1 0 1 yields:
      - **Quotient Q = 1 1 0 1 0 1 0 1 1 0**
      - **Remainder R = 0 1 1 1 0**
  - So, FCS = R = 0 1 1 1 0: Append it to M to get the full frame CW to be transmitted
  - CW = 1 0 1 0 0 0 1 1 0 1 0 1 1 1 0

M FCS

Example – Modulo-2 Arith. Method



Do not enter leading zeros

# of bits = # of bits in G,
⇒ result of division is 1

# of bits < # of bits in G,
⇒ result of division is 0, and use next digit

Checks you should do (exercise):

- **Verify correct operation, i.e. that $2^{(n-k)}M = G*Q + R$**
- **Verify that the obtained CW(1010001101011110) divides G (110101) exactly (i.e. with zero remainder)**

Example – Modulo-2 Arith. Method

For G = 110011 & M = 11100011, find the CRC

CW to transmit is? Answer: 11100011<u>11010</u>

Hamming Code (Detect & Correct)

Hamming Code is an error control technique where the redundant bits (CB) are spread at strategic position within the message bits (M).

- The position of these redundant bits are always at position 2^n (where n =0,1,2,3,...) i.e. position 1,2,4,8,...
- The number of redundant bits needed depends on the number of bits in the message (M).
- It is usually expressed as a function ***H(CW,M)*** e.g **H(11,7)** i.e. 7 message bits and 4 Code Bits (CB) yielding an 11-bit Codeword (CW)

Hamming Code: Code Bits Generation

At the SENDER:

Suppose M = 101000001 (9 bits)

13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	0	?	0	0	0	?	1	?	?

The following equation should hold $2^{CB} \geq M+CB+1$

We reserve **4 boxes**: 1,2,4 and 8 for the code bits (CB), and insert the message bits (M) in the remaining boxes. There should be 9+4 = 13 boxes in all which represents the 13 bits in the codeword (CW).

- To obtain the values of the code bits (the 4 boxes with interrogation marks), we perform a **modulo-2 addition** of all the box positions containing a '1' bit.

- In modulo-2 addition, we count the number of '1's in each column respectively. If the number of '1's is even, the addition yield 0 else if the number of '1's is odd, the addition yields 1.

Hamming Code: Code Bits Generation

Modulo-2 addition yields:

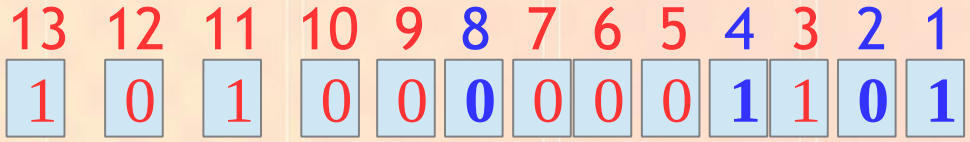
```
13:  1101
11:  1011
3 :  0011
     0101    = Code Bits
```

These become the code bits and are substituted back in the interrogation mark boxes. The transmitted codeword therefore becomes:

13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	0	**0**	0	0	0	**1**	1	**0**	**1**

Hamming Code: Error Checking

At the RECEIVER:



A modulo-2 addition is performed on the received codeword with all the box positions containing a '1':

13: 1101

11: 1011

4: 0100

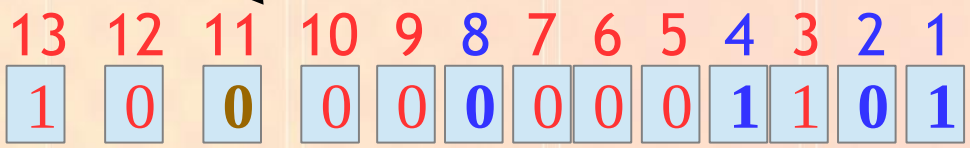
3: 0011

1: 0001

0000 **Since** addition is 0, it implies that no errors have taken place.

Hamming Code: Error Correction

Assume that at the RECEIVER, bit number 11 is in error:



A modulo-2 addition is performed on the received codeword with all the box positions containing a '1':

13: 1101
4: 0100
3: 0011
1: 0001
1011

Since addition is NOT 0, it implies that a error has taken place. The bit position in error is given by the result of the addition **i.e. 1011 = 11th bit.** So to correct, we simply invert the bit value.

Hamming Code

- It is always assumed that the code bits are not corrupted during transmission.
- Hamming code can only detect and correct **1 bit** in error in the message M.
- The efficiency of Hamming Code increases as the number of bits in the message becomes larger.

Summary

- Single parity bit checking can **only detect** 1 or an odd number of bits in error in the message M. It has the highest efficiency as it needs only one code bit irrespective of the length of the message M.
- 2-Dimensional parity bit checking can **detect and correct** 1 or more errors as long as 1 or an odd number of bits in error occur in different rows and/or columns.
- CRC can **only detect** any number of bits in error in the message M. The number of code bits needed is always one bit less than the divisor irrespective of the length of the message M.
- Hamming code can **detect and correct** a single bit in error in the message M. The number of code bits needed increases with the the length of the message M.