

Error Control: Detection and Correction

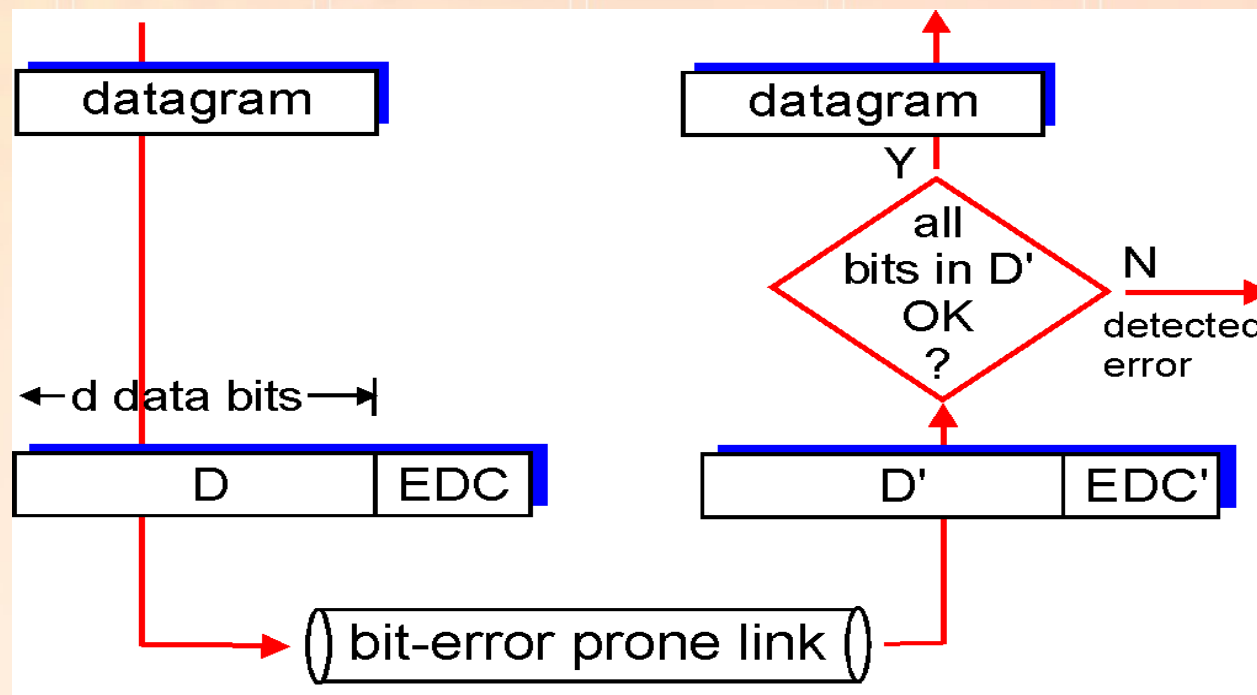
Slide Set 4

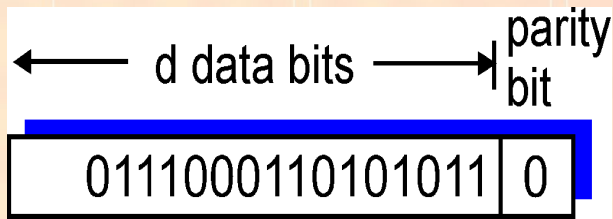


EDC = Error Detection and Correction bits (redundancy)

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





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.

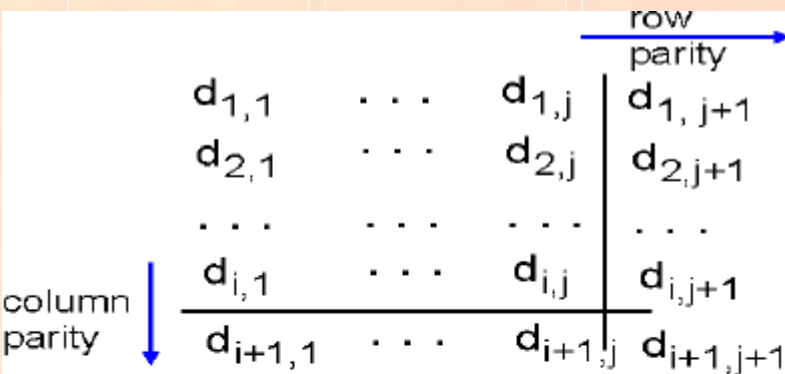
Even Parity Scheme:

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).



```

1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
-----
0 0 1 0 1 | 0
    
```

no errors

```

1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
-----
0 0 1 0 1 | 0
    
```

parity error

*correctable
single bit error*

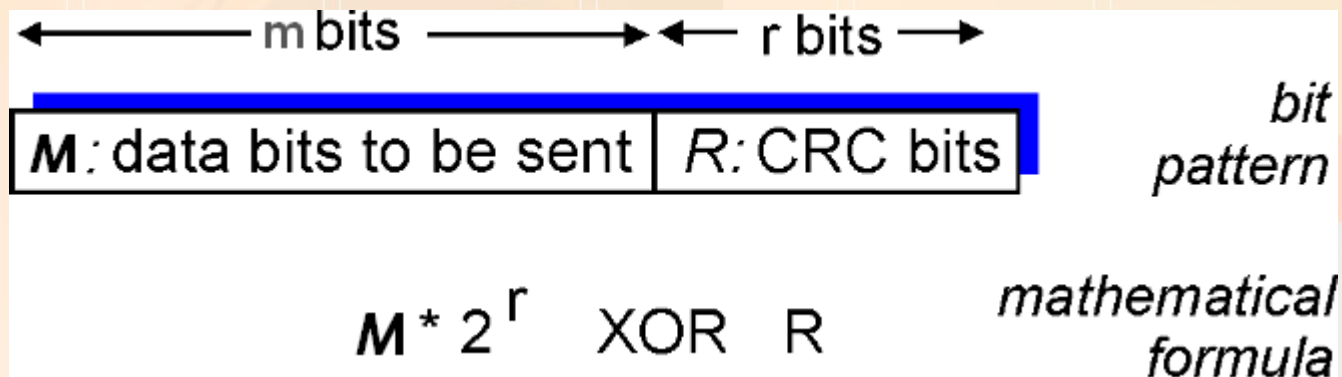
In this technique, the data bits are rearranged in an $n \times m$ matrix. Ideally a square matrix for higher efficiency. The parity bit is chosen for each row and column in the matrix. An additional parity bits can be used for checking the parity bits themselves but this is optional.

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.

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

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



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

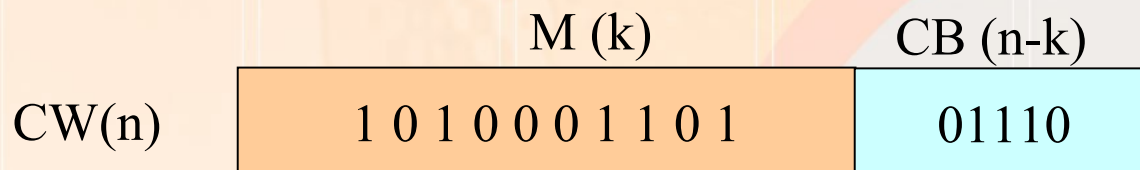
The FCS bits are used at Receiver for error detection

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

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

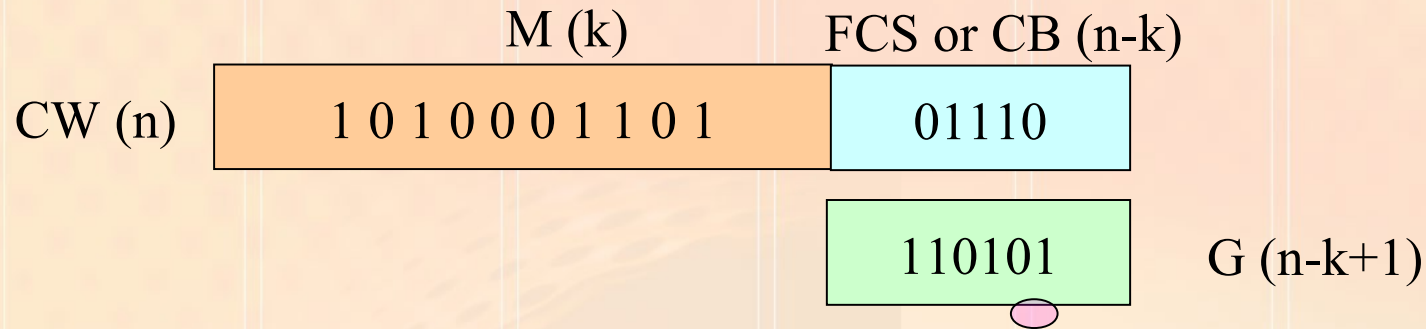
$(n - k)$ = the number of added bits as the **FCS** field or Code Bits (CB)

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

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

$$0 \text{ XOR } 0 = 0$$

$$0 \text{ XOR } 1 = 1$$

$$1 \text{ XOR } 0 = 1$$

$$1 \text{ XOR } 1 = 0$$

Examples:

$$1011 \text{ XOR } 0101 = 1110$$

$$1001 \text{ XOR } 1101 = 0100$$

■ Given

□ $M = 1010001101$

□ $G = 110101$ (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

■ Solution (continued):

□ Multiply $2^{(n-k)} \times M$

■ $2^{(5)} \times 1010001101 = 101000110100000$

■ This is a simple shift to the left by five positions and inserting (n-k) zeroes.

□ Divide $2^{(n-k)} \times M / G$ (see next slide for details)

■ $101000110100000 \div 110101$ yields:

□ Quotient $Q = 11010110$

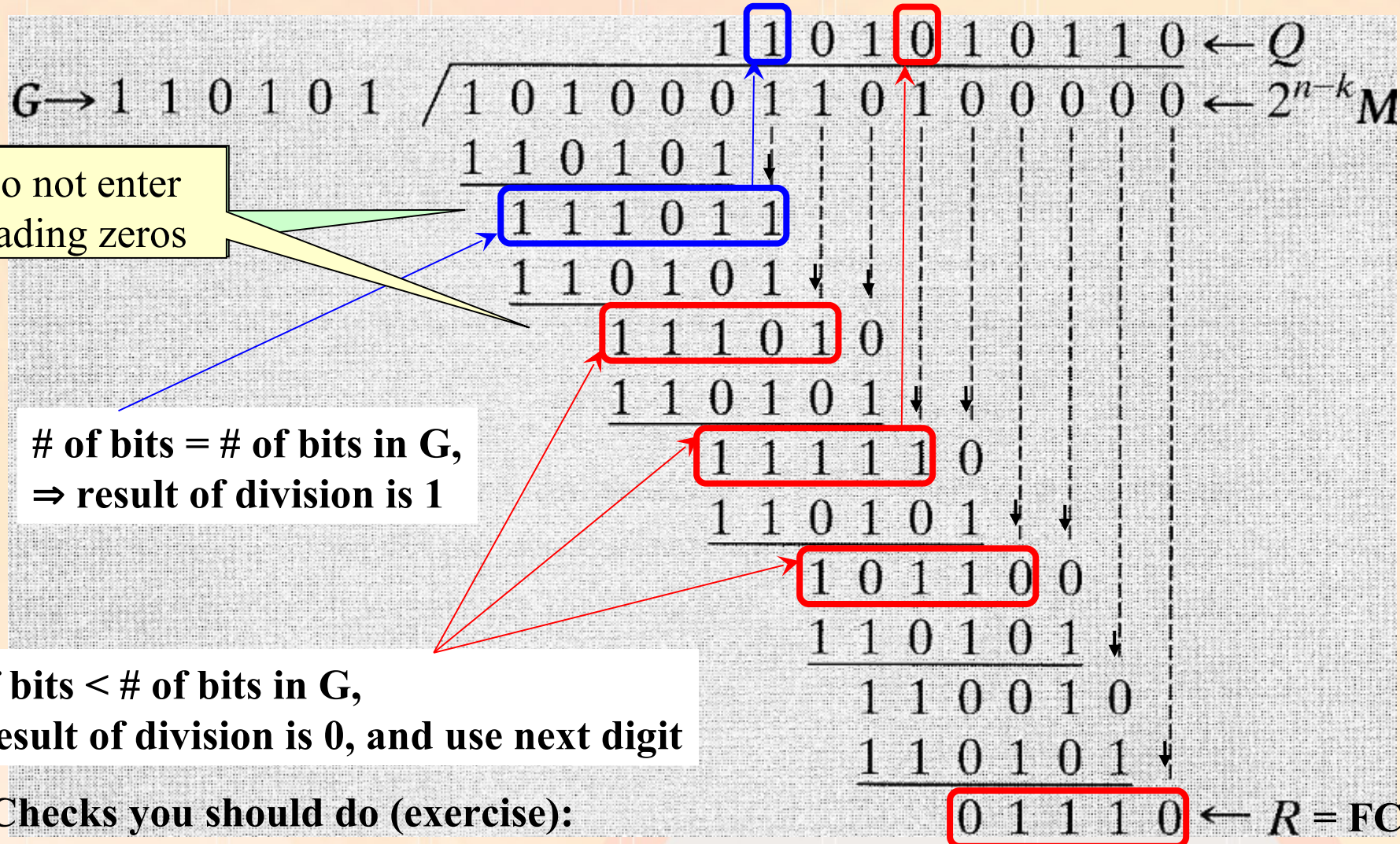
□ Remainder $R = 01110$

□ So, FCS = R = 01110 : Append it to M to get the full frame CW to be transmitted

□ CW = 101000110101110

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(101000110101110) divides G (110101) exactly (i.e. with zero remainder)

01110 ← R = FCS = CB

Example – Modulo-2 Arith. Method

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

```

                                10110110
110011 / 1110001100000
          110011 : : : : :
          -----
           101111 : : : : :
           110011 : : : : :
           -----
            111000 : : : : :
            110011 : : : : :
            -----
             101100 : : : : :
             110011 : : : : :
             -----
              111110 : : : : :
              110011 : : : : :
              -----
               CRC = 11010
```

CW to transmit is? Answer: 11100011**11010**

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 $n = 0, 1, 2, 3, \dots$) 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	?	?

We reserve **4 boxes**: 1,2,4 and 8 for the code bits, and insert the message bits in the remaining boxes. There are 13 boxes in all that will represent the 13 bits in the codeword.

- 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.

Modulo-2 addition yields:

$$\begin{array}{r} 13: \quad 1101 \\ 11: \quad 1011 \\ 3: \quad \underline{0011} \\ \hline \underline{0101} \quad = \text{Code Bits} \end{array}$$

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

At the RECEIVER:

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

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

13: 1101

11: 1011

4: 0100

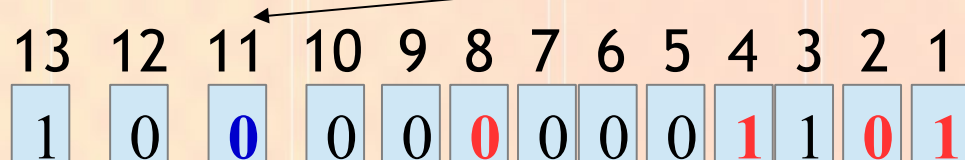
3: 0011

1: 0001

0000

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

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
	<hr/>
	1011

Since addition is NOT 0, it implies that an 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

- 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.