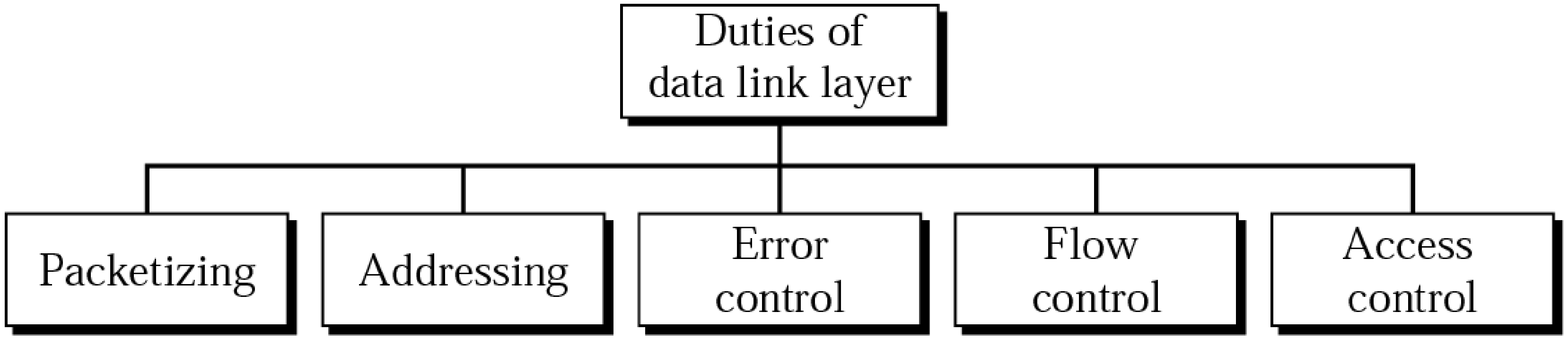
Data Communications and Networking Fourth Edition

Forouzan

Part III Datalink Layer

Data link layer duties



Chapter 10 :Error Detection and Correction
Chapter 11 Data Link Control and Protocols
Chapter 12 Multiple Access
Chapter 13 Wired LANs: Ethernet
Chapter 15 Connecting LANs, Backbone networks and virtual LAN

Chapter 10

Error Detection and Correction

Note

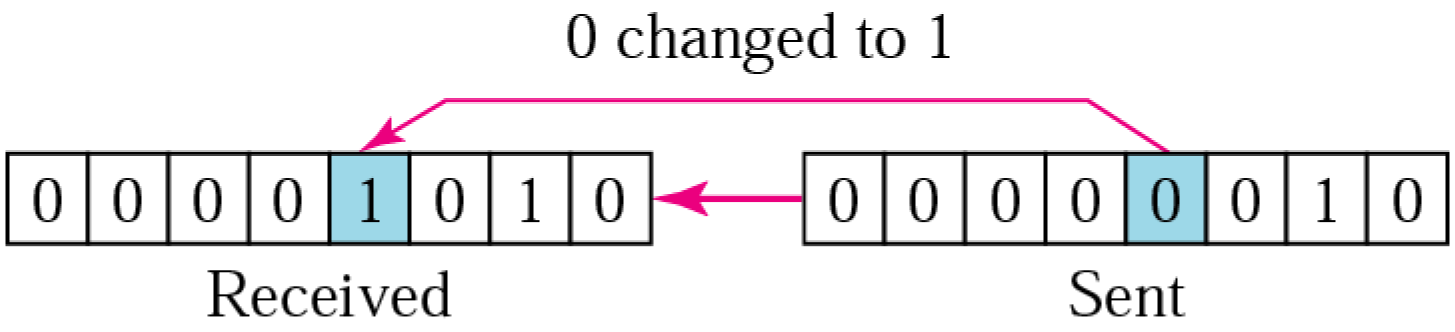
➢Network must be able to transfer data from one device to another with acceptable accuracy.

➢ Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.

Types of Error

- ***Single-Bit Error***
- ***Burst Error***

Single-bit error

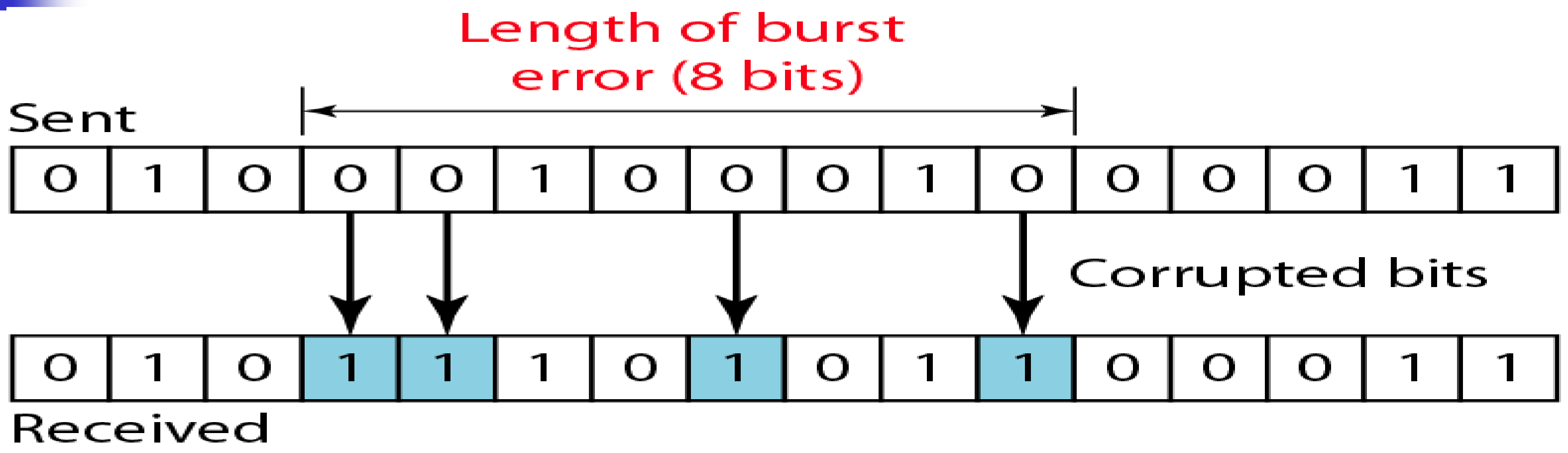


➢**In a single-bit error, only one bit in the data unit has changed. (0 to 1 or 1 to 0)**

➢ **Does the Single-bit error happen in parallel or serial transmission ? Why ?**

➢ **Single-bit error most likely to occur in parallel transmission (eg. between CPU and memory)**

Burst error



Burst burst error means that 2 or more bits in the data unit have changed

Burst error does not necessarily mean that the errors occur in consecutive bits

Most likely to happen in a serial transmission

Number of bits affected depends on the data rate and noise duration

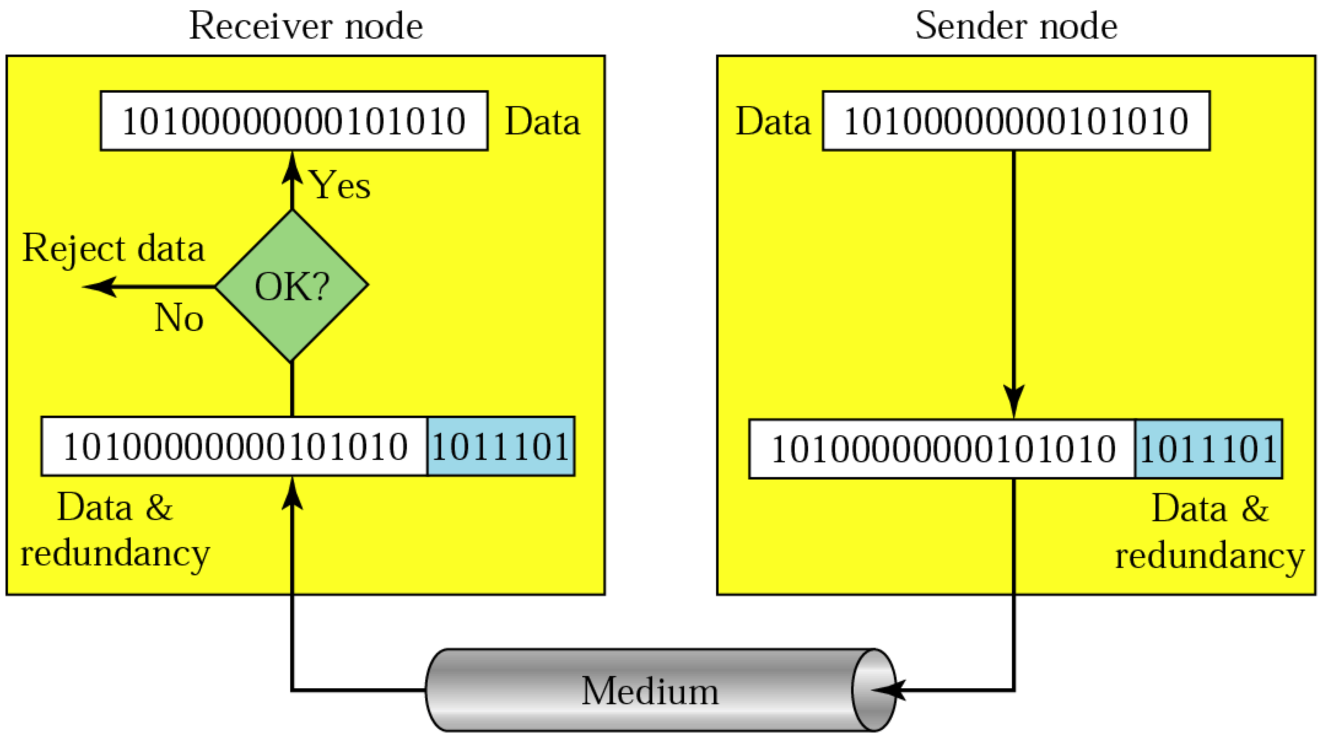
If the data is sent at 1 kbps, a noise of 1/100 s can affect 10 bits.

If the data is sent at 1 Mbps, a noise of 1/100 s can affect 10000 bits.

error

ERROR DETECTION

Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

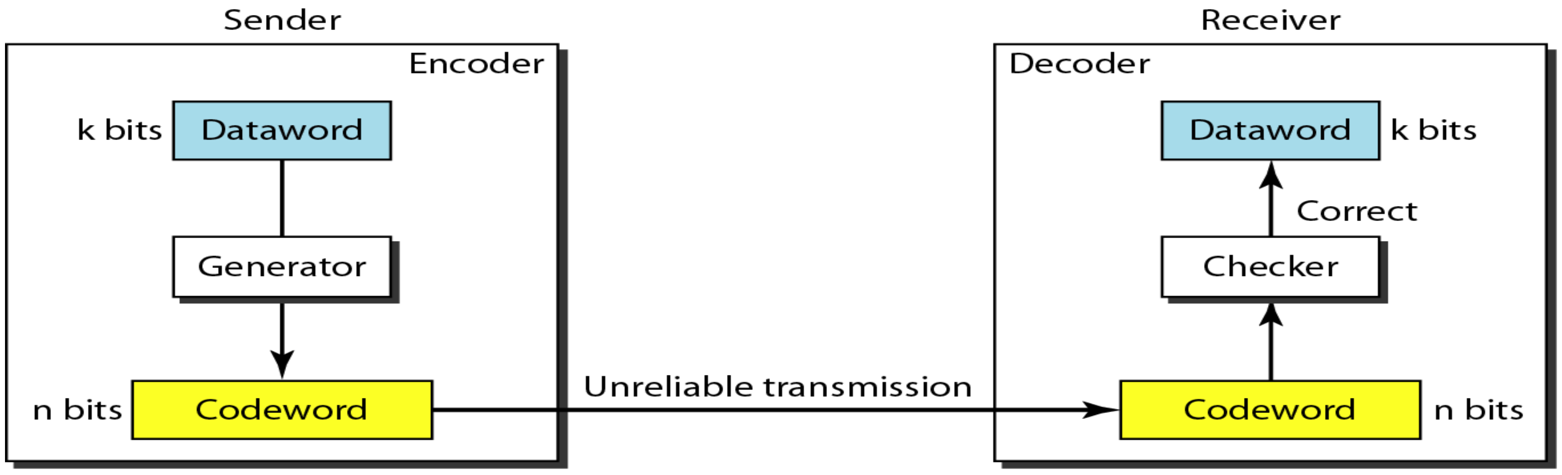


ERROR DETECTION

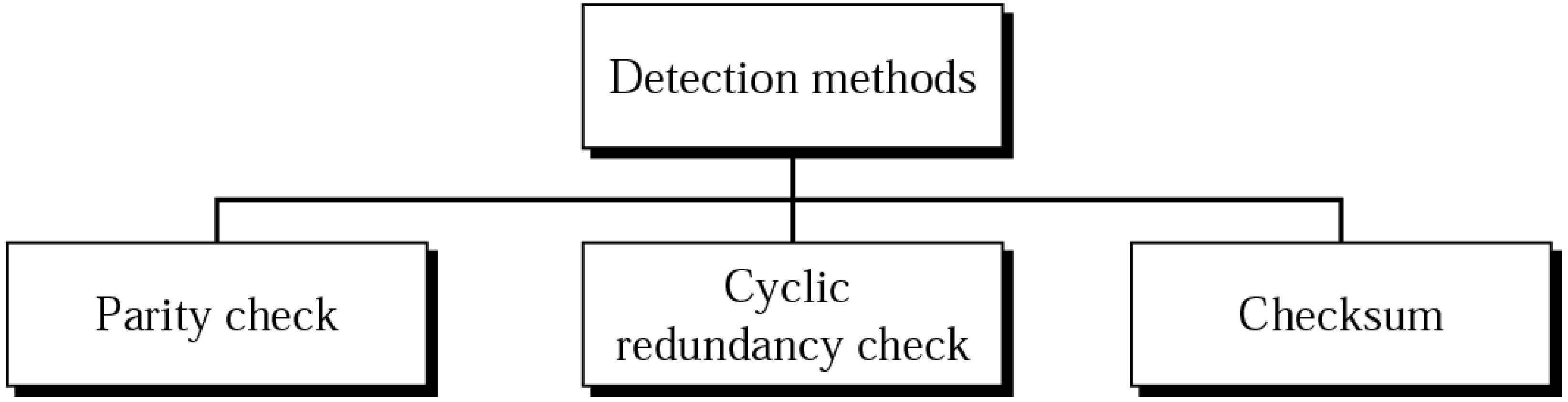
Error can be detected by using block coding:

- **Dived the message into blocks, each of k bits, called datawords.**
- Add r redundant bits to each block to make the length

  n = k + r.
- **The resulting n bits blocks are called codewords**



Error Detection methods

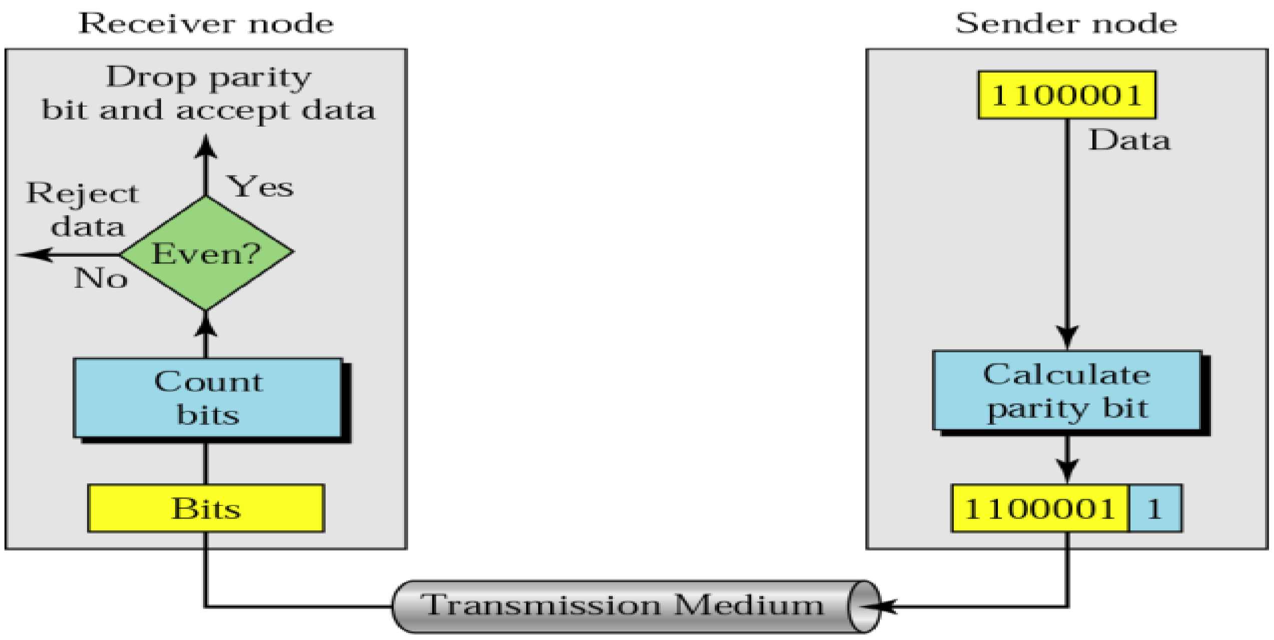


parity Check

The most common and least expensive

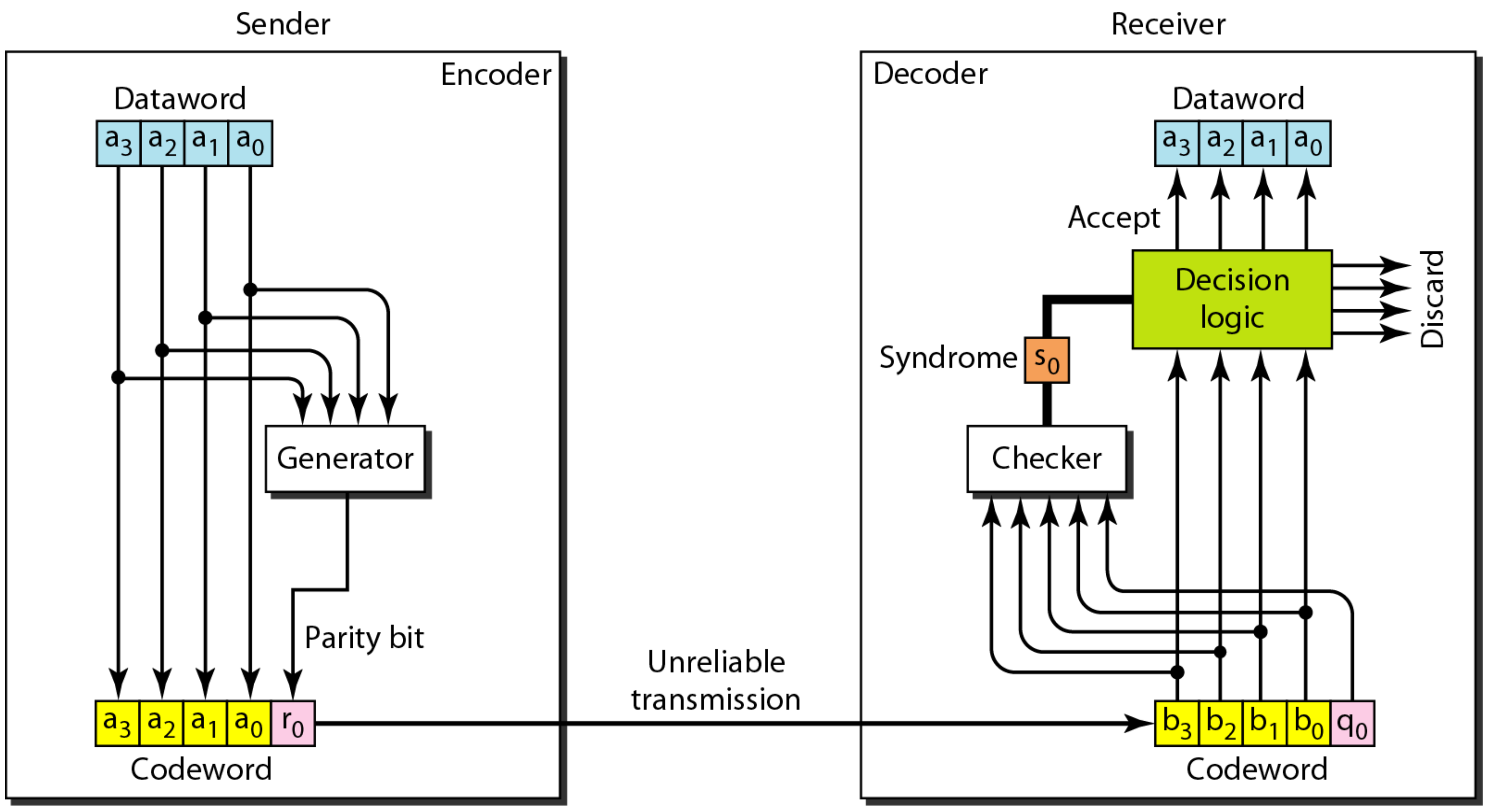
Simple or Two Dimensional

simple parity check



a parity bit (extra bit) is added to every data unit so that the total number of 1s is even or odd (for odd-parity).

Figure 10.10 *Encoder and decoder for simple parity-check code*



Error Detection methods

Suppose the sender wants to send the word world. In ASCII the five characters are coded as

w	o	r	l	d
1110111	**1101111**	**1110010**	**1101100**	**1100100**

The following shows the actual bits sent

11101110 11011110 11100100 11011000 11001001

Error Detection methods

Now suppose the word world in Example 1 is received by the receiver without being corrupted in transmission.

11101110   11011110   11100100   11011000   11001001

The receiver counts the 1s in each character and comes up with even numbers:
(6, 6, 4, 4, 4). The data are accepted.

Error Detection methods

Now suppose the word world in Example 1 is corrupted during transmission.

11111110   11011110   11101100   11011000   11001001

The receiver counts the 1s in each character and comes up with even and odd numbers:
(7, 6, 5, 4, 4).

The receiver knows that the data are corrupted, discards them, and asks for retransmission.

Performance of Simple Parity Check

Example:

We have an even-parity data unit where the total number of 1s, including the parity bit, is 6: 1000111011.

- If any 3 bits changed, the resulting parity will be odd and the error will be detected:
    - 1111111011: “ 9 ones”
    - 0110111011: “7 ones”
    - 1100010011: “ 5 ones”

- The same holds true for any odd number of errors

Performance of Simple Parity Check

Example:

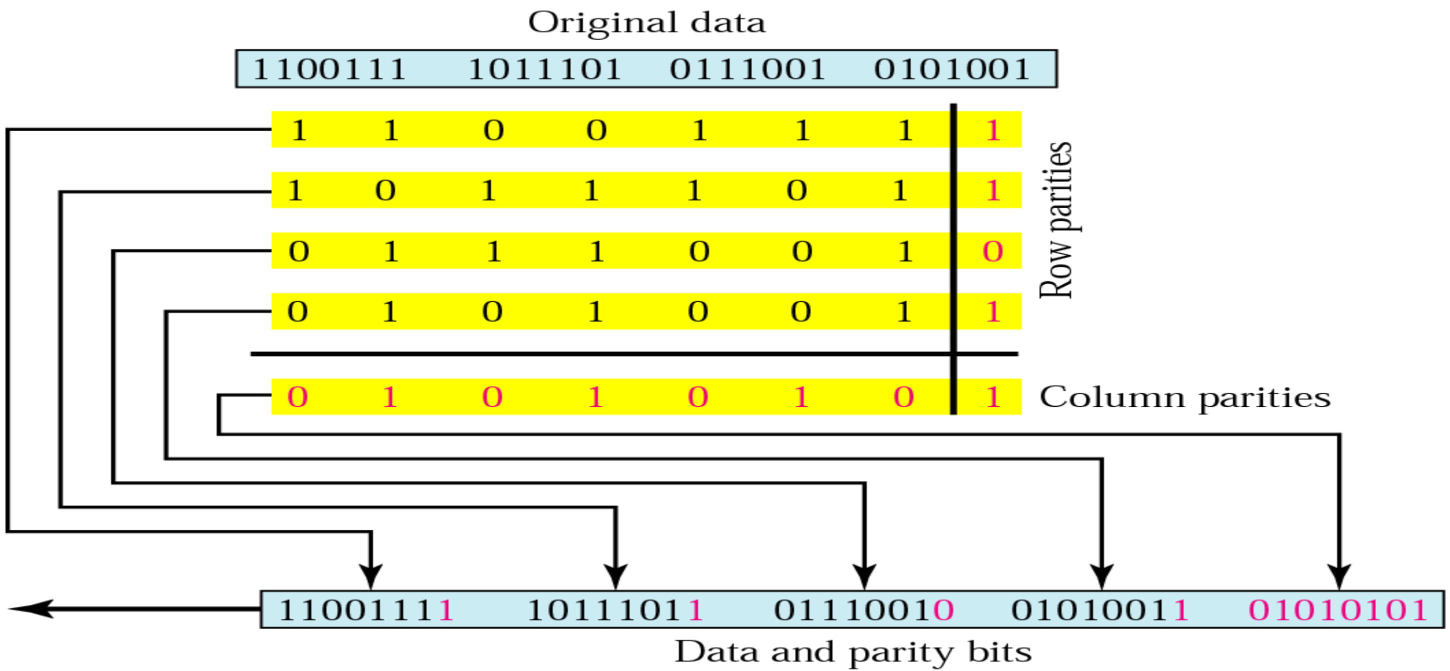
- Suppose 2 bits are changed:
    - 1110111011: ”8 ones”
    - 1100011011: ”6 ones”
- The number of 1s in the data unit is still even .The same holds true for any even number of errors
- The parity checker cannot detect errors when number of bits changed is even. The change cancel each other and the data unit will pass a parity check even though the data unit is damaged.

Note

Simple parity check can detect all single--bit error. It can detect burst errors only if the total number of errors in each data unit is odd.

Two-dimensional parity

- Calculate the parity bit for each data unit, then organize them into table (rows and columns)
- Calculate the parity bit for each column and create a new row (column parity)
- A redundant row of bits is added to the whole block.



Note

In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Figure 10.11 ***Two-dimensional parity-check code***

Suppose the following block is sent:

10101001   00111001   11011101   11100111   10101010

However, it is hit by a burst noise of length 8, and some bits are corrupted.

10100011   10001001   11011101   11100111   10101010

When the receiver checks the parity bits, some of the bits do not follow the even-parity rule and the whole block is discarded.(see next slide)

Figure 10.11 *Two-dimensional parity-check code*

At Sender:

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

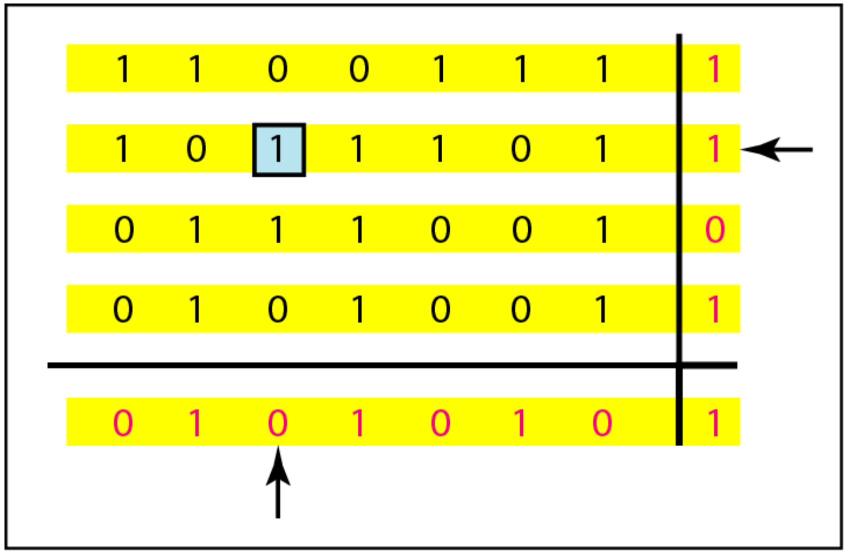
At receiver:

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

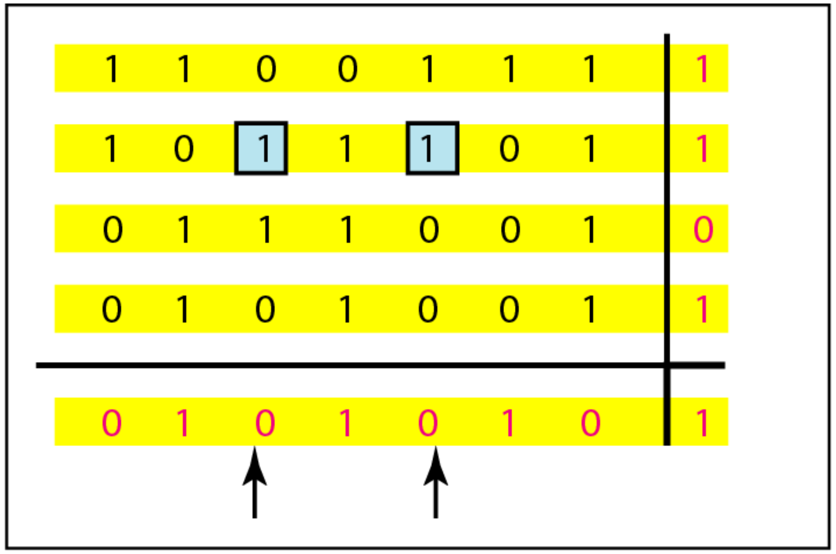
Two-dimensional Performance

- Increases the likelihood of detecting burst errors.
- A redundancy of n-bits can detect a burst error of n bits
- A burst error of more than n bits is also detected with a very high probability
- One pattern of errors still elusive: If 2 bits in one data unit are damaged and two bits in exactly the same position in anther data unite are also damaged. The error will not be detected

Figure 10.11 *Two-dimensional parity-check code*

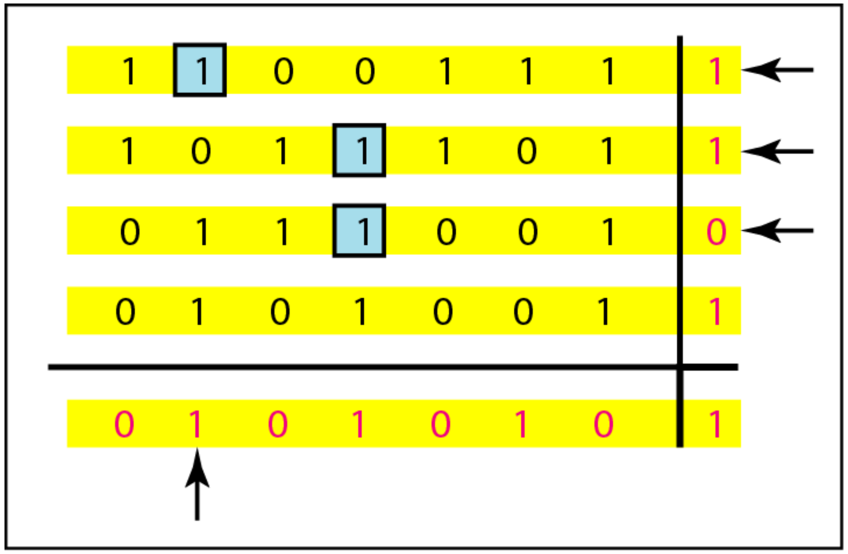


b. One error affects two parities

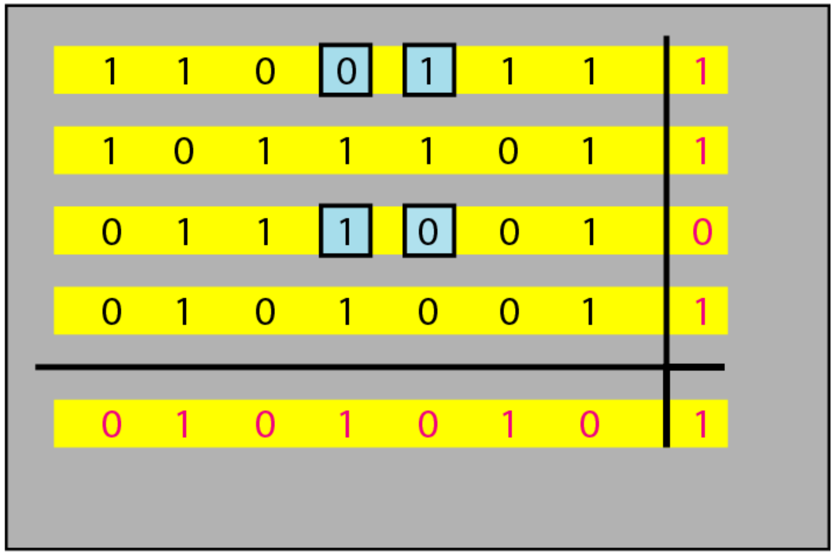


c. Two errors affect two parities

Figure 10.11 *Two-dimensional parity-check code*



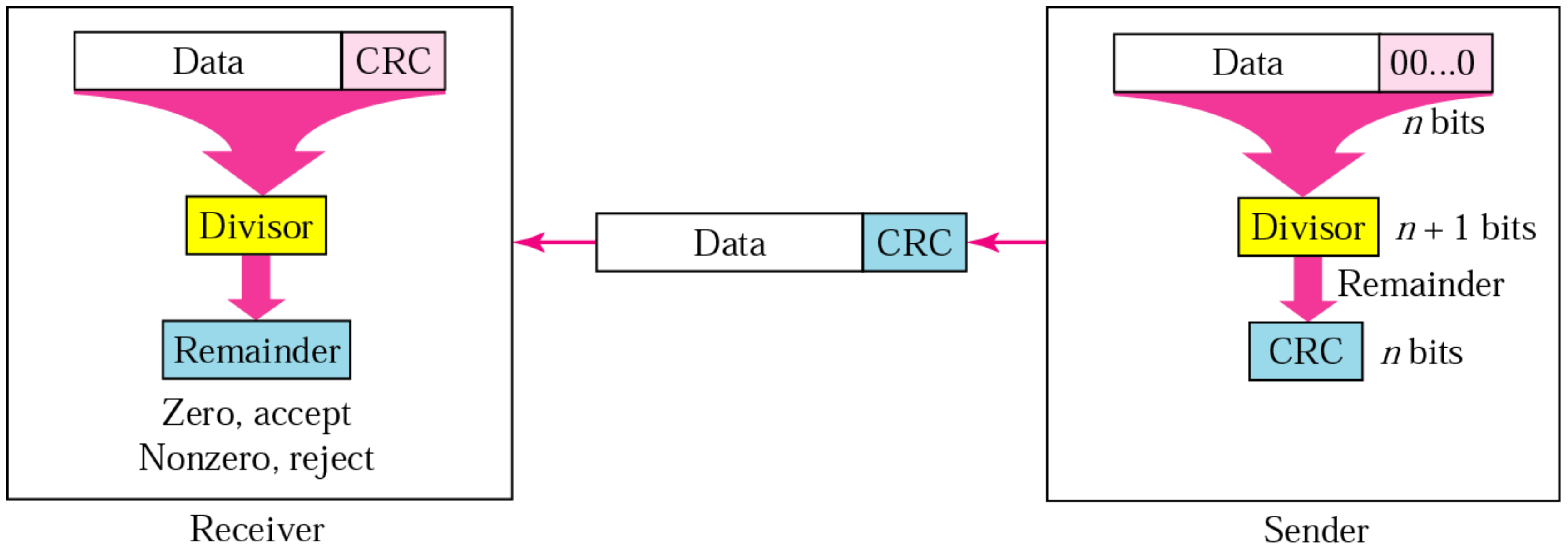
d. Three errors affect four parities



e. Four errors cannot be detected

Cyclic Redundancy Check CRC

Uses Module-2 Binary division



Two-dimensional Performance

In CRC generator (At sender)

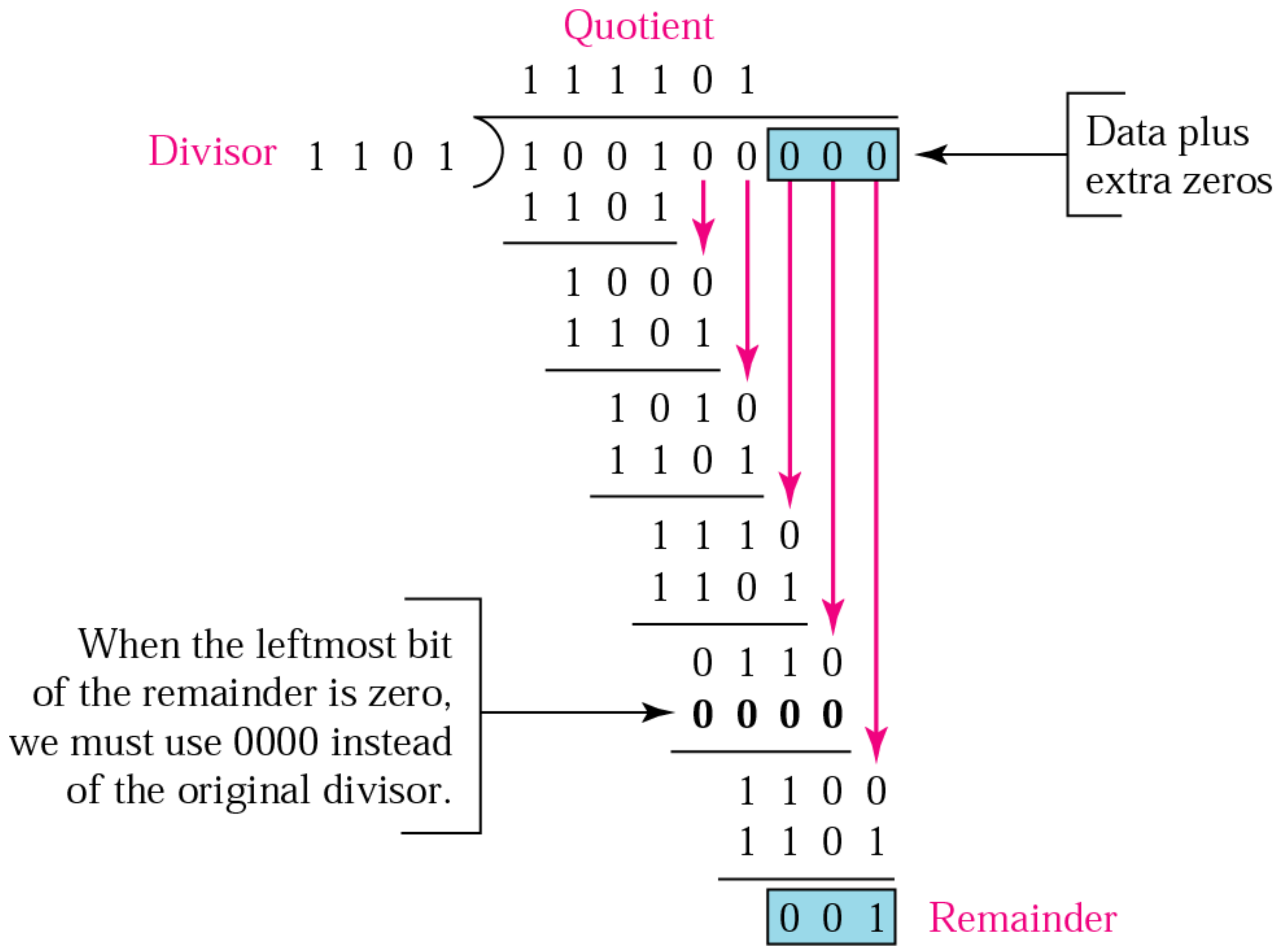
- ➢ A string of n 0s is appended to the data unit
- ➢ The number n is 1 less than the number of bits in the divisor
- ➢ Divide the data word plus appended zeros by the divisor

Use module-2 binary division:

- There is no carry when you add or subtract two digits in a column
- Addition and subtraction gives the same results
- This means: you can use XOR operation for both Addition and subtraction

- ➢ The remainder resulting from the division is the CRC
- ➢ The CRC of n bits replaces the appended 0s at the end of the data unit.
- ➢ Appending CRC to the end of the data must make resulting bit sequence divisible by the divisor

Figure 10.15 Binary division in a CRC generator



Two-dimensional Performance

In CRC generator (At receiver)

- **After receiving the data appended with the CRC, it does the same module -2 division**

- **If the remainder is all 0s the CRC dropped and the data are accepted (the data is correct)**

- **If the remainder is not equal zero, the received stream of bits is discarded and data must be resent (the data is corrupted)**

Figure 10.15 *Division in CRC encoder*

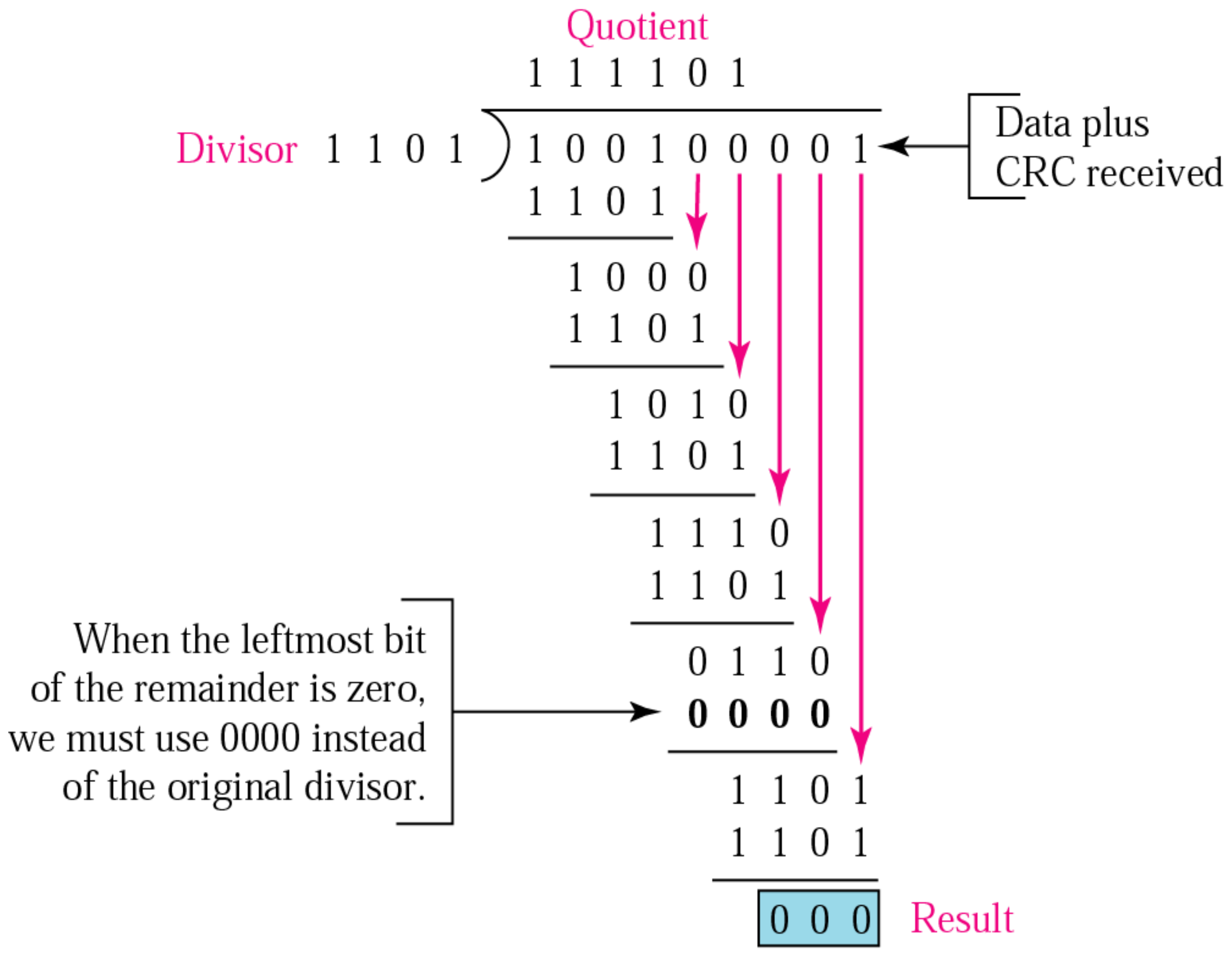


Figure 10.15 ***Division in CRC encoder***

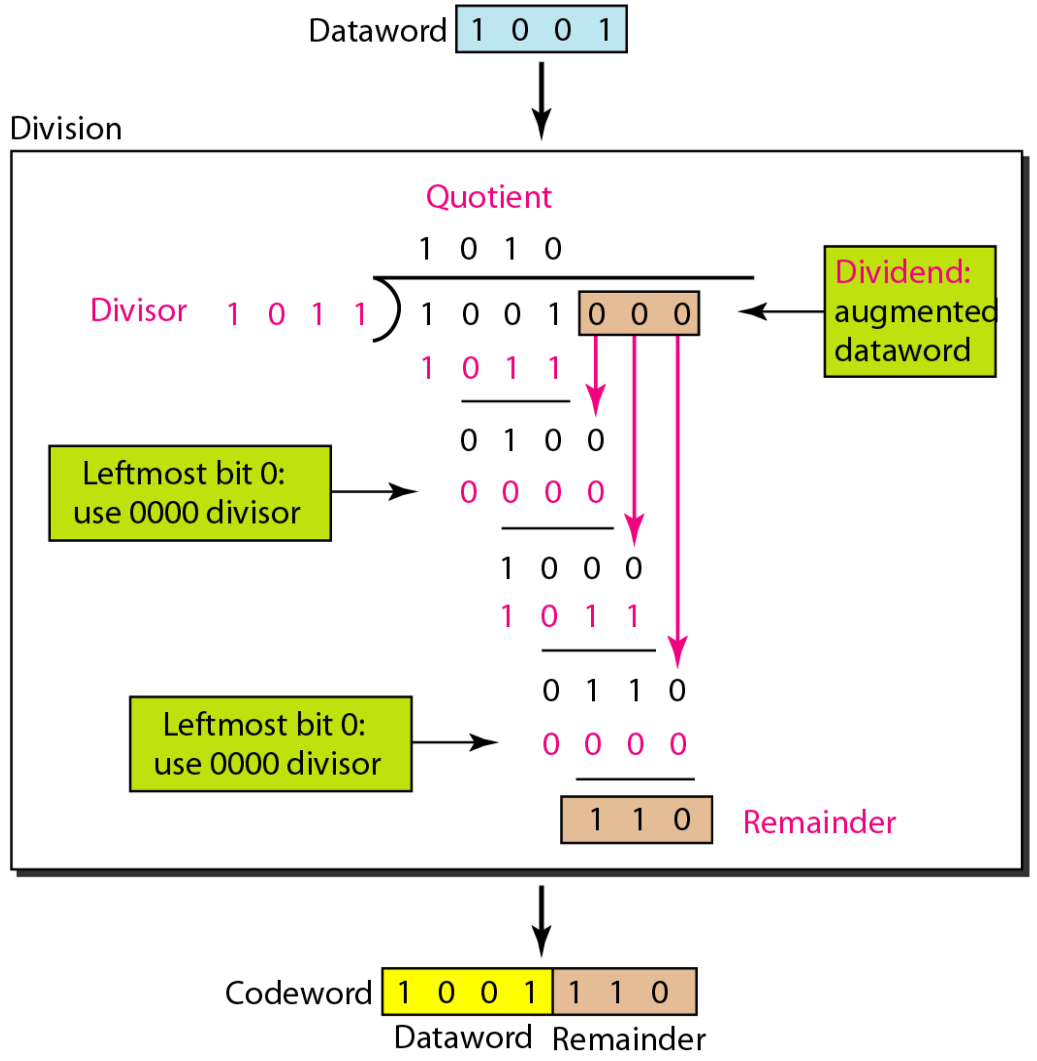
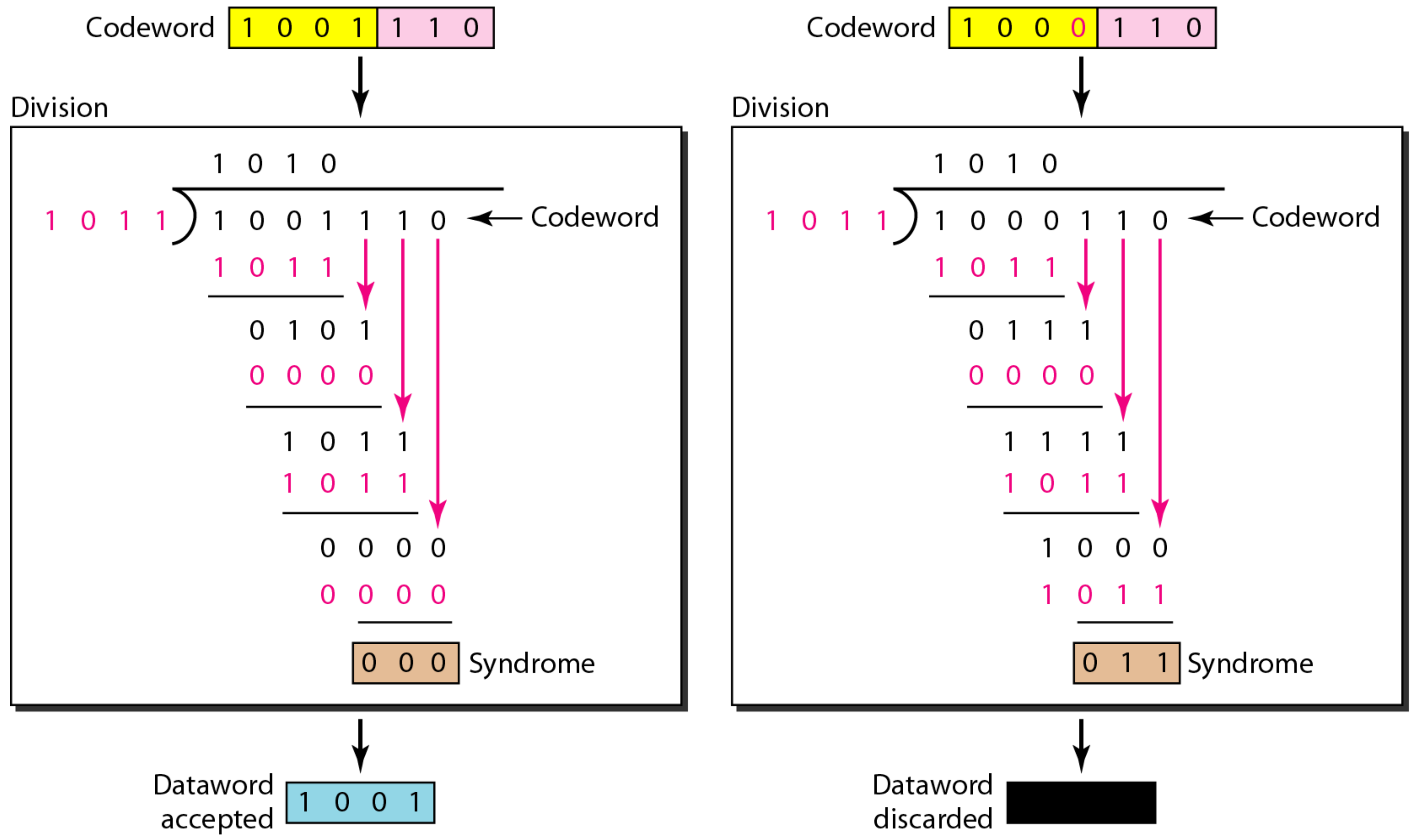


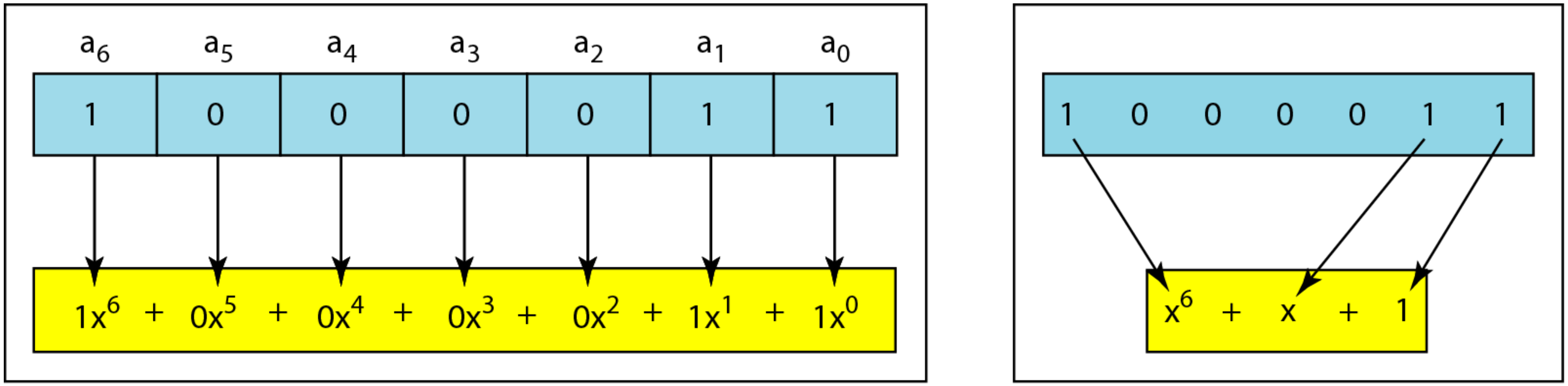
Figure 10.16 *Division in the CRC decoder for two cases*



Using Polynomials

- The divisor in CRC generator is most often represented not as string of 1s and 0s, but as an algebraic **polynomial.**
- We can use a polynomial to represent a binary word.
- Each bit from right to left is mapped onto a power term.
- The rightmost bit represents the “0” power term. The bit next to it the “1” power term, etc.
- If the bit is of value zero, the power term is deleted from the expression.

Figure 10.21 *A polynomial to represent a binary word*

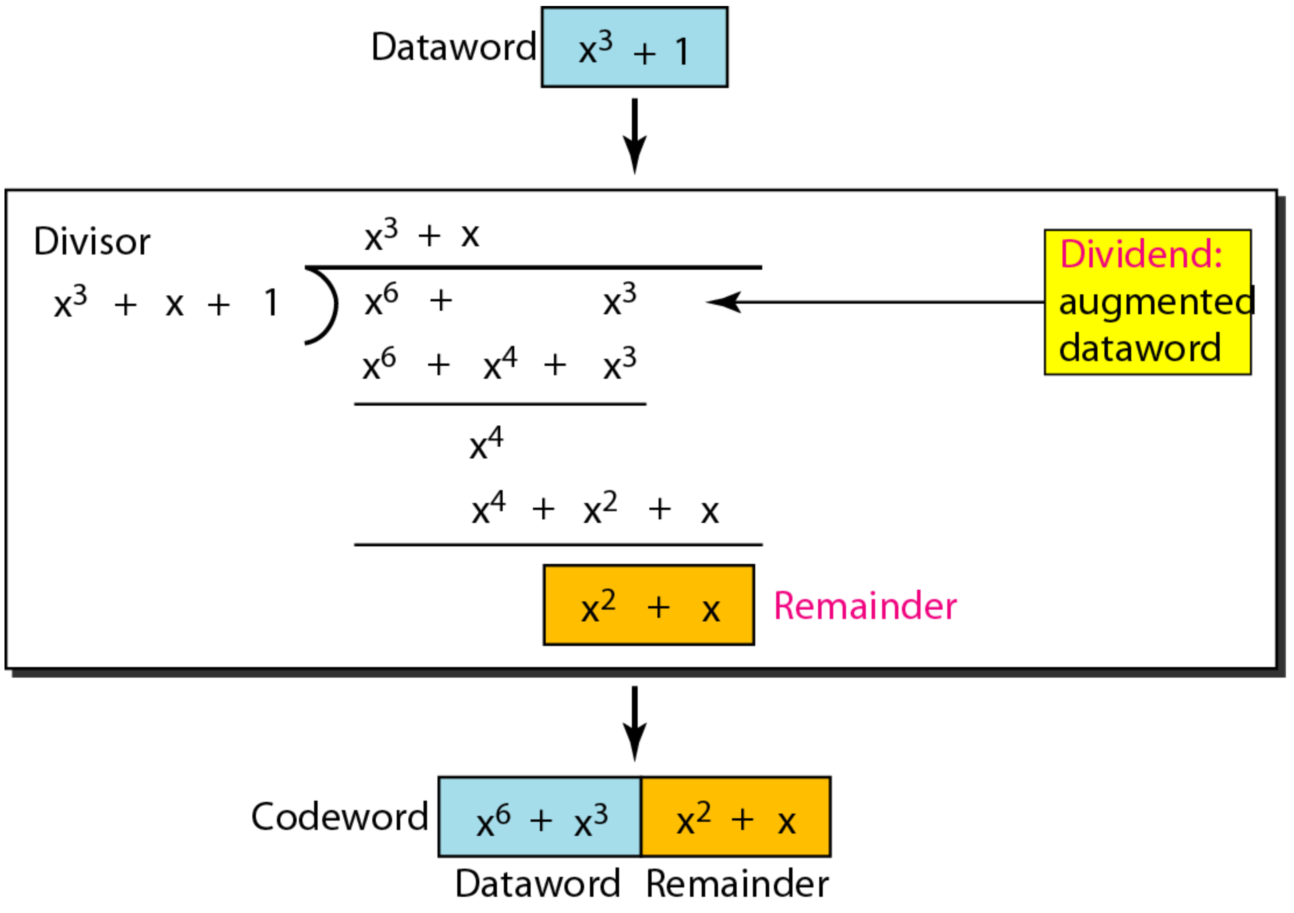


a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$
1 0 0 0 0 1 1
$x^6 + x + 1$

a. Binary pattern and polynomial

b. Short form

Figure 10.22 *CRC division using polynomials*



Note

In a cyclic code,

If $s(x) \neq 0$, one or more bits is corrupted.

If $s(x) = 0$, either

a. No bit is corrupted. or

b. Some bits are corrupted, but the decoder failed to detect them.

CRC Performance

CRC is very effective error detection method:

1. **CRC Can detect all burst errors that affect an odd number of bits.**

2. CRC Can detect all burst errors of length less than or equal to the degree of the polynomial

3. CRC can detect, with very high probability, burst errors of length greater than the degree of the polynomial.

Figure 10.22 *CRC division using polynomials*

The CRC-12 x12 + x11 + x3 + x + 1 which has a degree of 12, will detect

1.all burst errors affecting an odd number of bits,

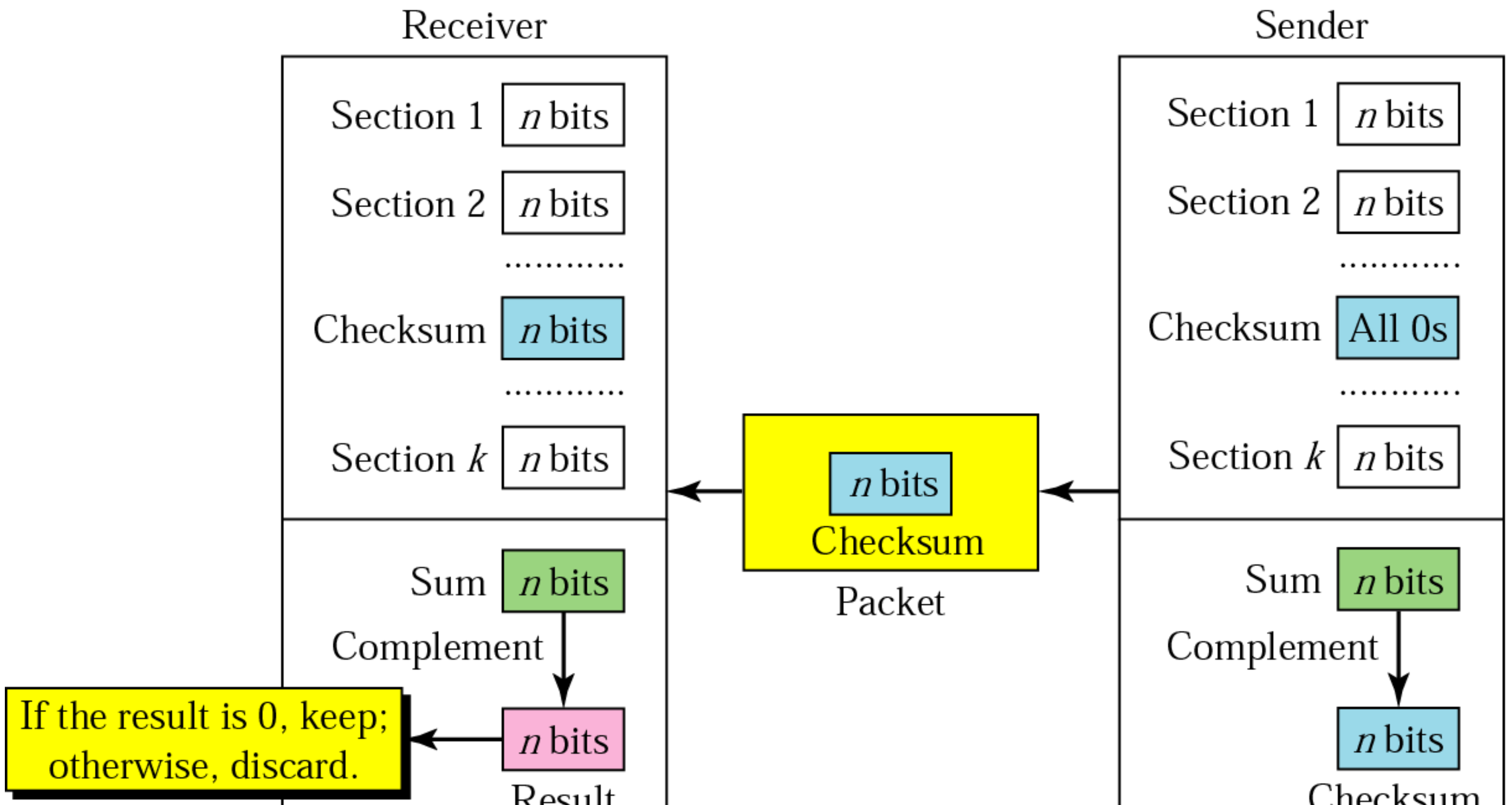
2.will detect all burst errors with a length less than or equal to 12, and

3. will detect, 99.97 percent of the time, burst errors with a length of 12 or more.

10-5 CHECKSUM

The last error detection method we discuss here is called the checksum. The checksum is used in the Internet by several protocols although not at the data link layer.(Checksum used for example in network layer). However, we briefly discuss it here to complete our discussion on error checking

CHECKSUM


Receiver
Section 1
n bits
Section 2
n bits
Checksum
n bits
Section k
n bits
Sum
n bits
Complement
n bits
Result
If the result is 0, keep; otherwise, discard.
n bits
Checksum
Packet
Sender
Section 1
n bits
Section 2
n bits
Checksum
All 0s
Section k
n bits
Sum
n bits
Complement
n bits
Checksum

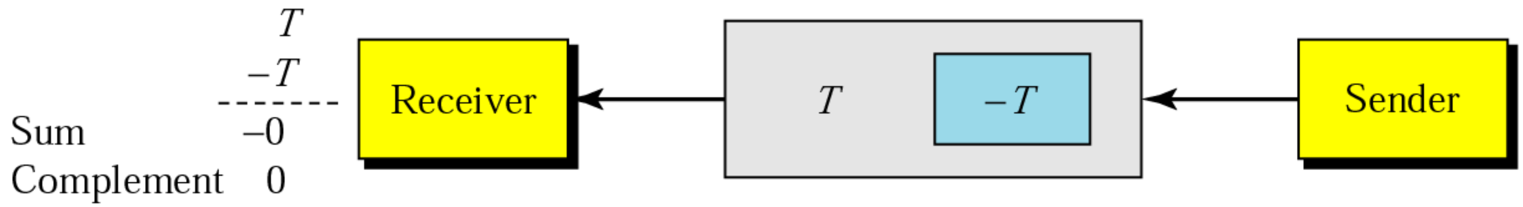
Data unit and checksum

The sender follows these steps:

1. The unit is divided into k sections, each of n bits.
2. All sections are added using one's complement to get the sum.
3. The sum is complemented and becomes the checksum.
4. The checksum is sent with the data

The receiver follows these steps:

1. The unit is divided into k sections, each of n bits.
2. All sections are added using one's complement to get the sum.
3. The sum is complemented.
4. If the result is zero, the data are accepted: otherwise, rejected



Example

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.
10101001   00111001

The numbers are added using one's complement

```
          1 0 1 0 1 0 0 1
          0 0 1 1 1 0 0 1
        ___________________

Sum       1 1 1 0 0 0 1 0
```

Checksum   00011101
The pattern sent is 10101001 00111001 00011101

Example 7

Now suppose the receiver receives the pattern sent in Example 7 and there is no error.
10101001 00111001 00011101

When the receiver adds the three sections, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

```
            1 0 1 0 1 0 0 1
            0 0 1 1 1 0 0 1
            0 0 0 1 1 1 0 1
        -------------------
Sum         1 1 1 1 1 1 1  1
```

Complement 00000000 means ***that the pattern is OK.***

Example

Now suppose there is a burst error of length 5 that affects 4 bits.

10101111 11111001 00011101

When the receiver adds the three sections, it gets

```
                 1 0 1 0 1 1 1 1
                 1 1 1 1 1 0 0 1
                 0 0 0 1 1 1 0 1
              -------------------
Partial Sum    1 1 1 0 0 0 1 0 1
Carry                          1
              -------------------
 Sum             1 1 0 0 0 1 1 0
```

Complement 00111001 the pattern is corrupted.

Performance of checksum

Almost detects all errors involving odd numbers of bits or even.

if one or more bits of a segment are damaged and the corresponding bit or bits of opposite value in a second segment are also damaged, then the sum of columns will not change. Receiver will not be able to detect the error