DATA LINK LAYER

Error Detection and Correction

Note

Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

10-1 INTRODUCTION

Let us first discuss some issues related, directly or indirectly, to error detection and correction.

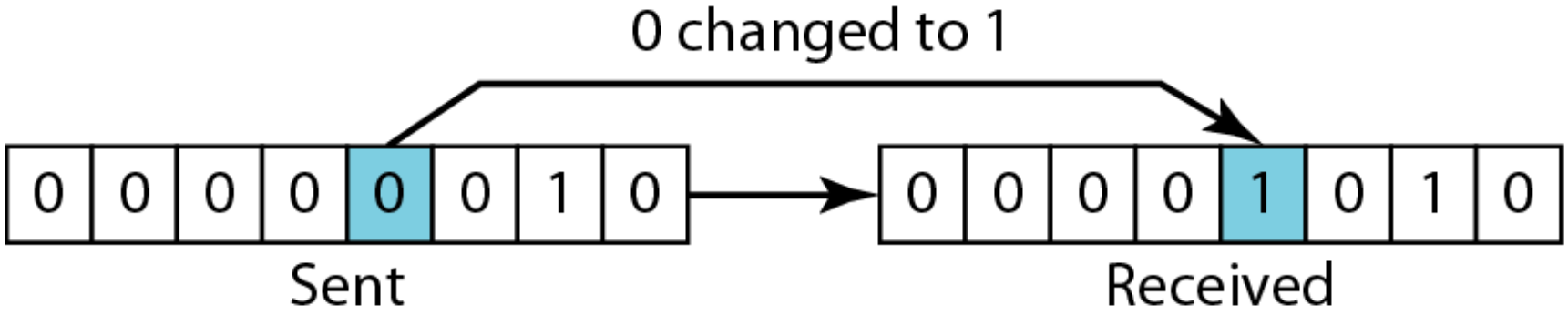
Topics discussed in this section:

Types of Errors
Redundancy
Detection Versus Correction
Forward Error Correction Versus Retransmission
Coding
Modular Arithmetic

Note

In a single-bit error, only 1 bit in the data unit has changed.

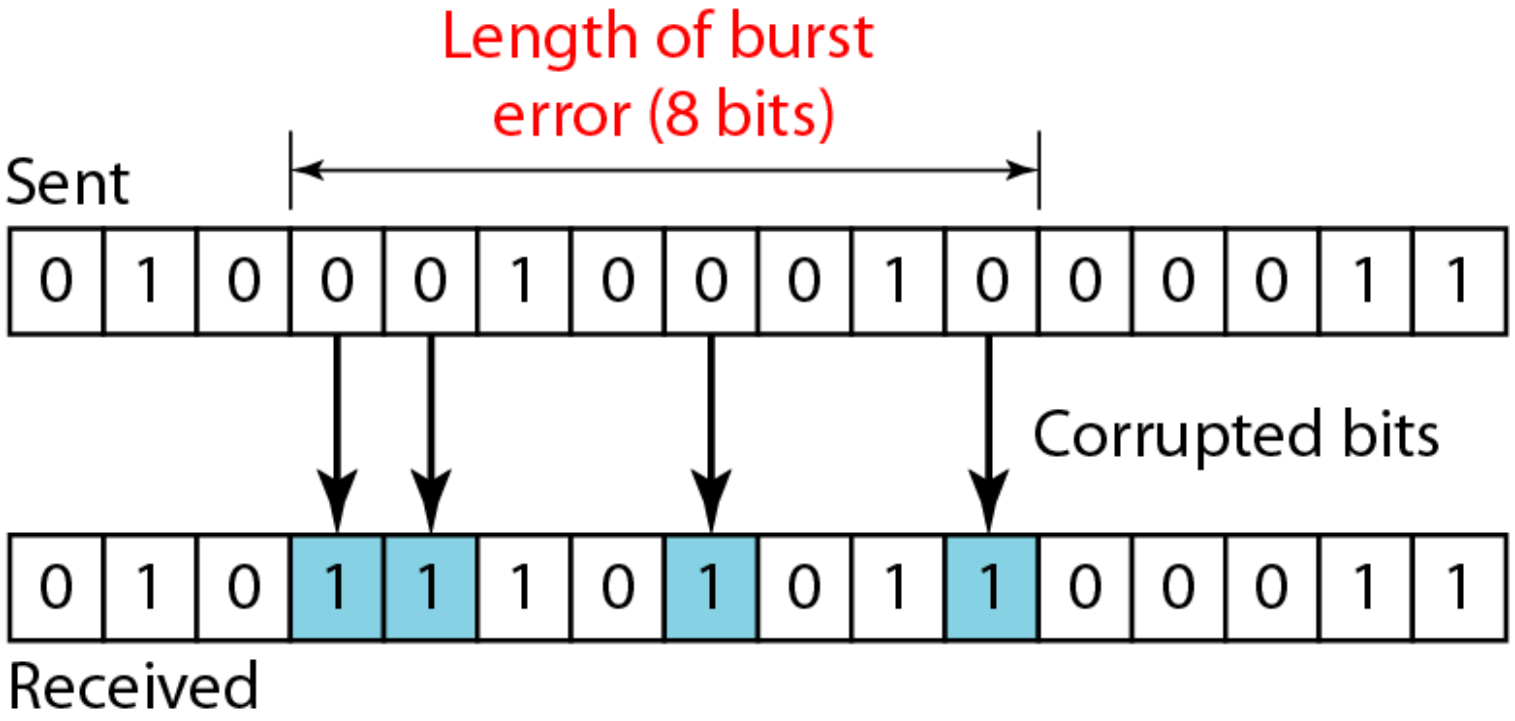
Figure 10.1 *Single-bit error*



Note

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

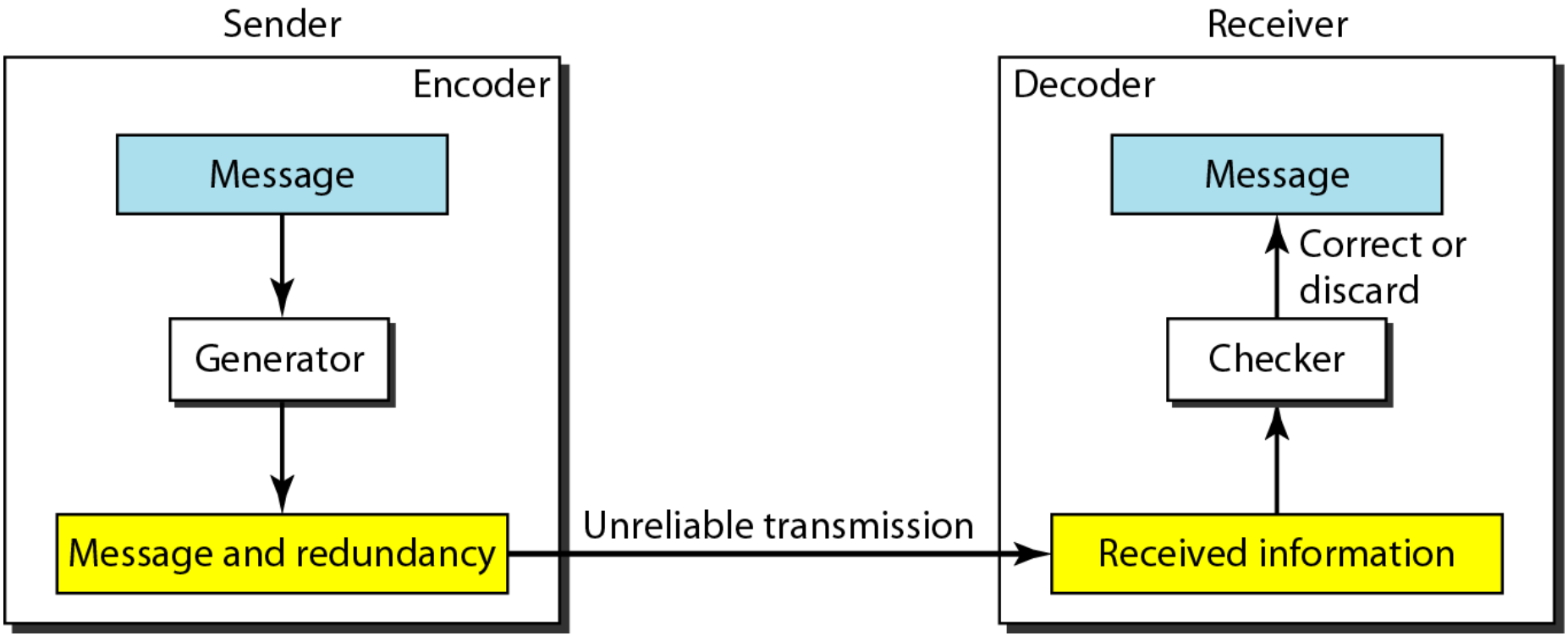
Figure 10.2 *Burst error of length 8*



Note

To detect or correct errors, we need to send extra (redundant) bits with data.

Figure 10.3 *The structure of encoder and decoder*



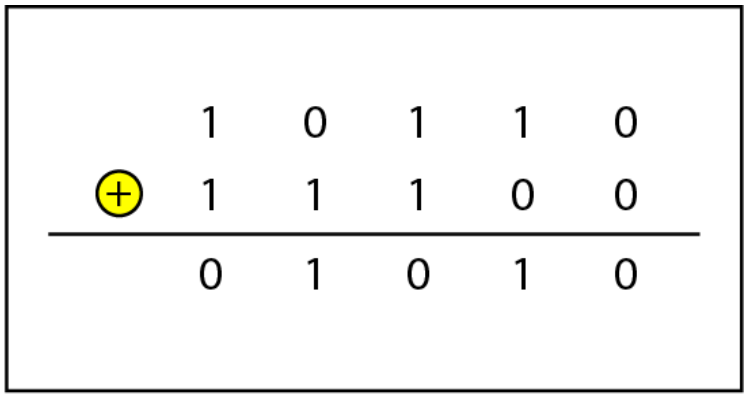
Note

In modulo-N arithmetic, we use only the integers in the range 0 to N −1, inclusive.

Figure 10.4 *XORing of two single bits or two words*

a. Two bits are the same, the result is 0.

b. Two bits are different, the result is 1.



c. Result of XORing two patterns

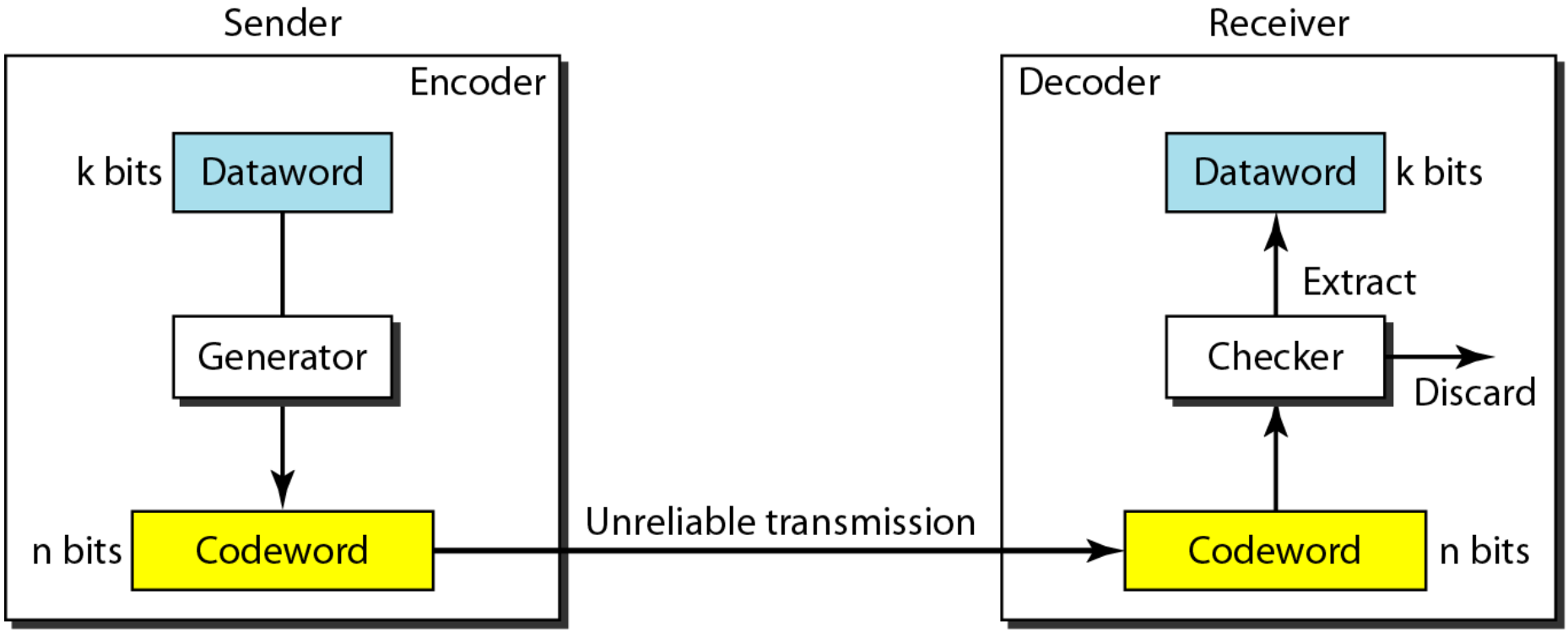
10-2 BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called datawords. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.

Topics discussed in this section:

Error Detection
Error Correction
Hamming Distance
Minimum Hamming Distance

Figure 10.6 *Process of error detection in block coding*



Example 10.2

Let us assume that k = 2 and n = 3. Table 10.1 shows the list of datawords and codewords (even parity). It is only good for detecting one bit error.

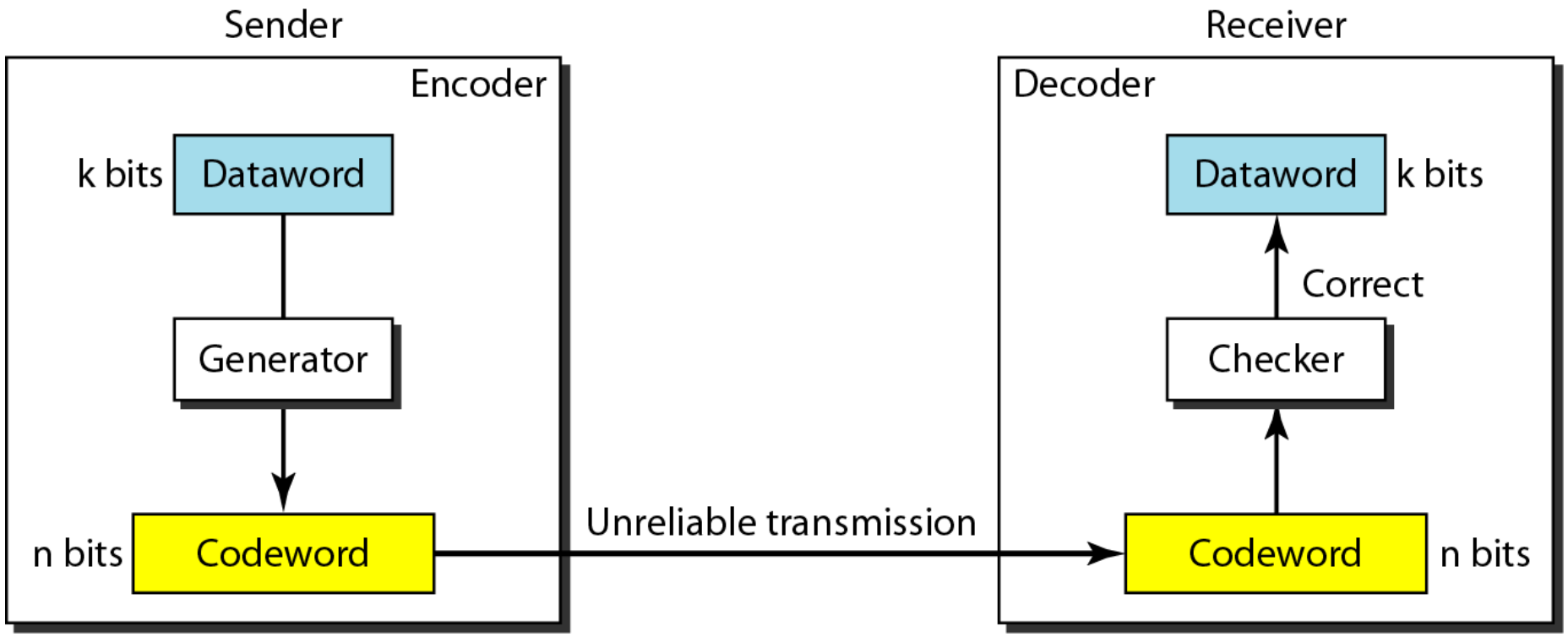
Table 10.1 *A code for error detection (Example 10.2)*

Datawords	Codewords
00	000
01	011
10	101
11	110

Note

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Figure 10.7 *Structure of encoder and decoder in error correction*



Note

The Hamming distance between two words is the number of differences between corresponding bits.

Example 10.4

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d(000, 011) is 2 because

$000 \oplus 011$ is 011 (two 1s)

2. The Hamming distance d(10101, 11110) is 3 because

$10101 \oplus 11110$ is 01011 (three 1s)

Note

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Example 10.5

Find the minimum Hamming distance of the coding scheme in Table 10.1.

Solution

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Note

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

Why?

More than s-bit error is possible to detect, but not guaranteed.

Figure 10.8 *Geometric concept for finding d_{min} in error detection*

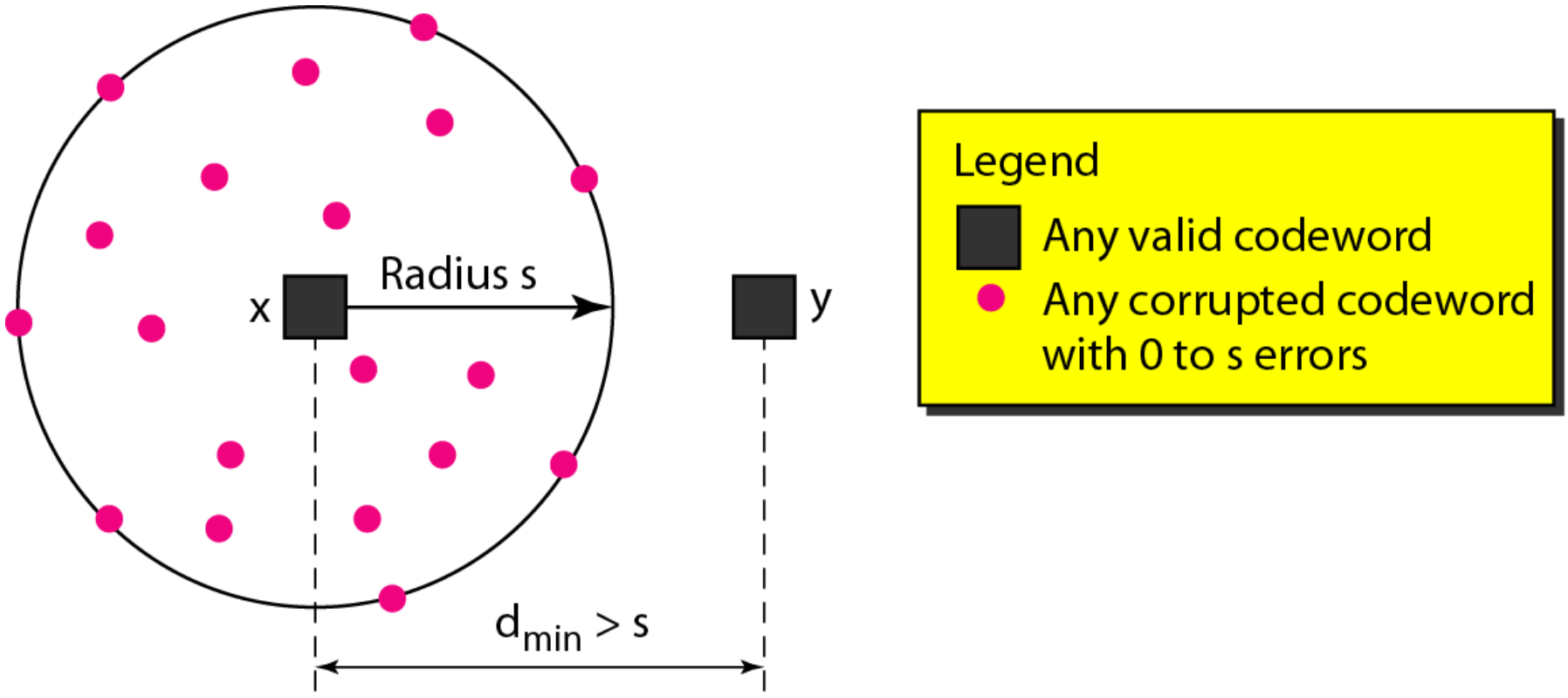
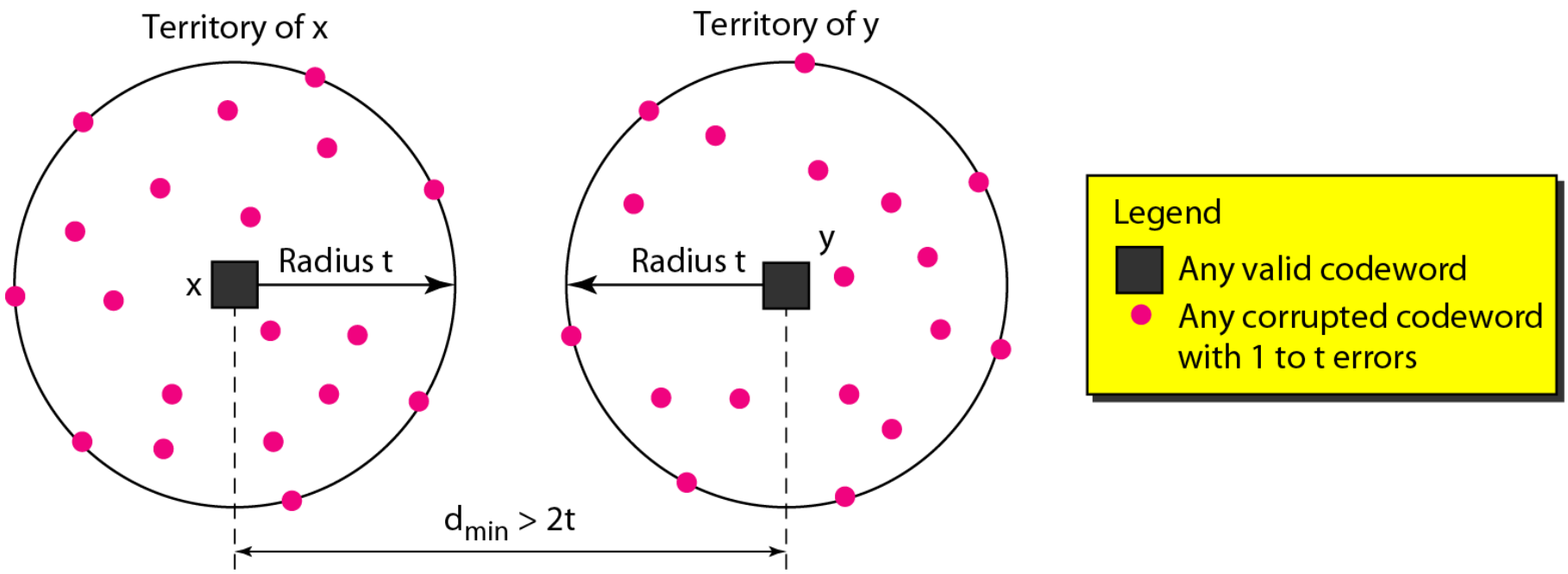


Figure 10.9 *Geometric concept for finding d_{min} in error correction*



Note

To guarantee correction of up to *t* errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Example 10.9

A code scheme has a Hamming distance $d_{min} = 4$. What is the error detection and correction capability of this scheme?

Solution

This code guarantees the detection of up to three errors ($s = 3$), but it can correct up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes should have an odd minimum distance (3, 5, 7, . . .).

10-3 LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called linear block codes. A linear block code is a code in which the XOR (addition modulo-2) of two valid codewords creates another valid codeword.

Topics discussed in this section:

Minimum Distance for Linear Block Codes
Some Linear Block Codes

Note

In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

Example 10.10

Let us see if the two codes we defined in Table 10.1 belong to the class of linear block codes.

The scheme in Table 10.1 is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword. For example, the XORing of the second and third codewords creates the fourth one.

Datawords	Codewords
00	000
01	011
10	101
11	110

Example 10.11

Note

In a linear block code, the minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.

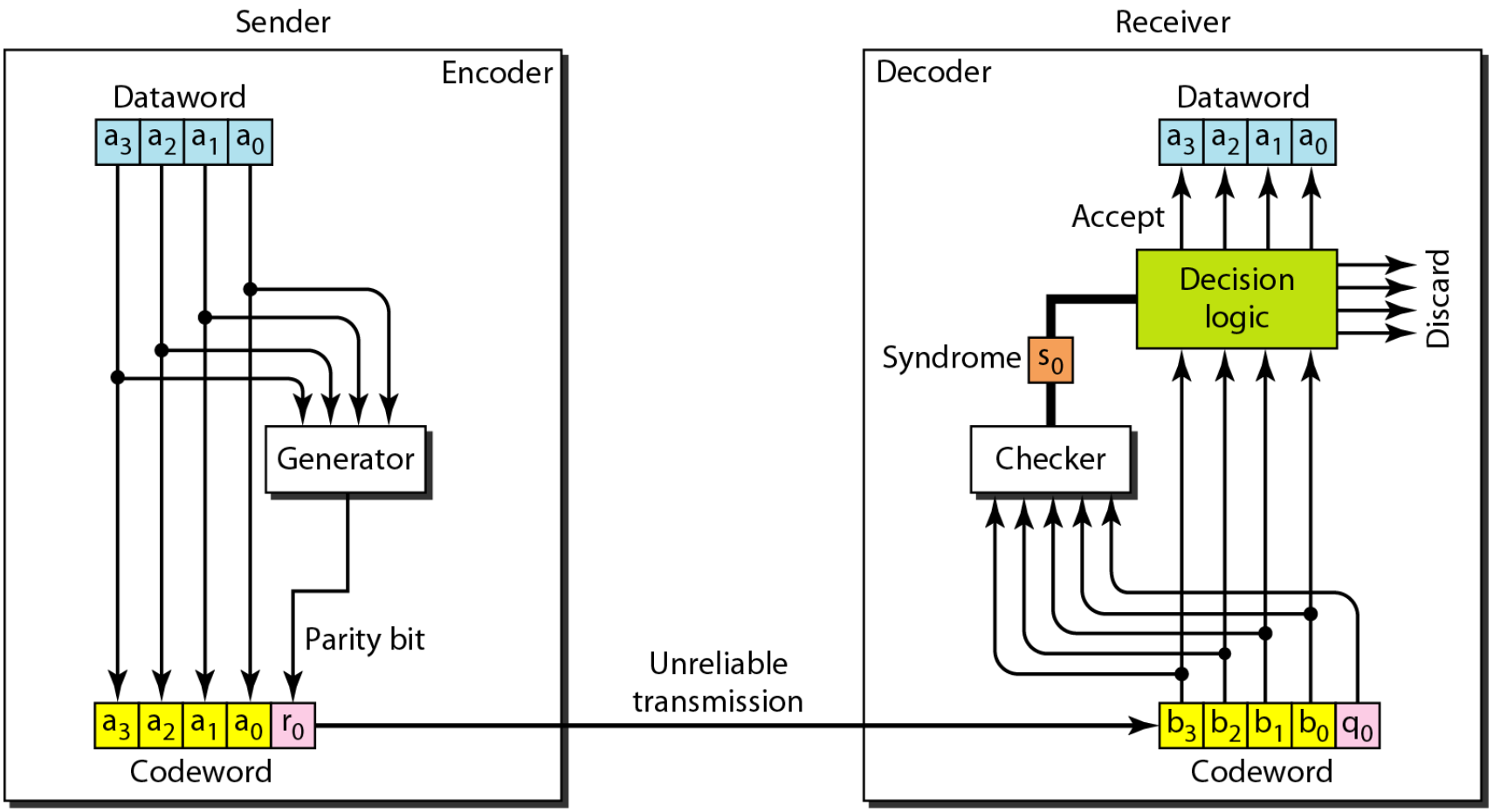
Note

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.

Table 10.3 ***Simple parity-check code C(5, 4)***

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

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



Parity-check code

- $r0 = a3 + a2 + a1 + a1$ (modulo-2)
- Syndrome (calculated by the receiver)
- $s0 = b3 + b2 + b1 + b0 + q0$ (modulo-2)

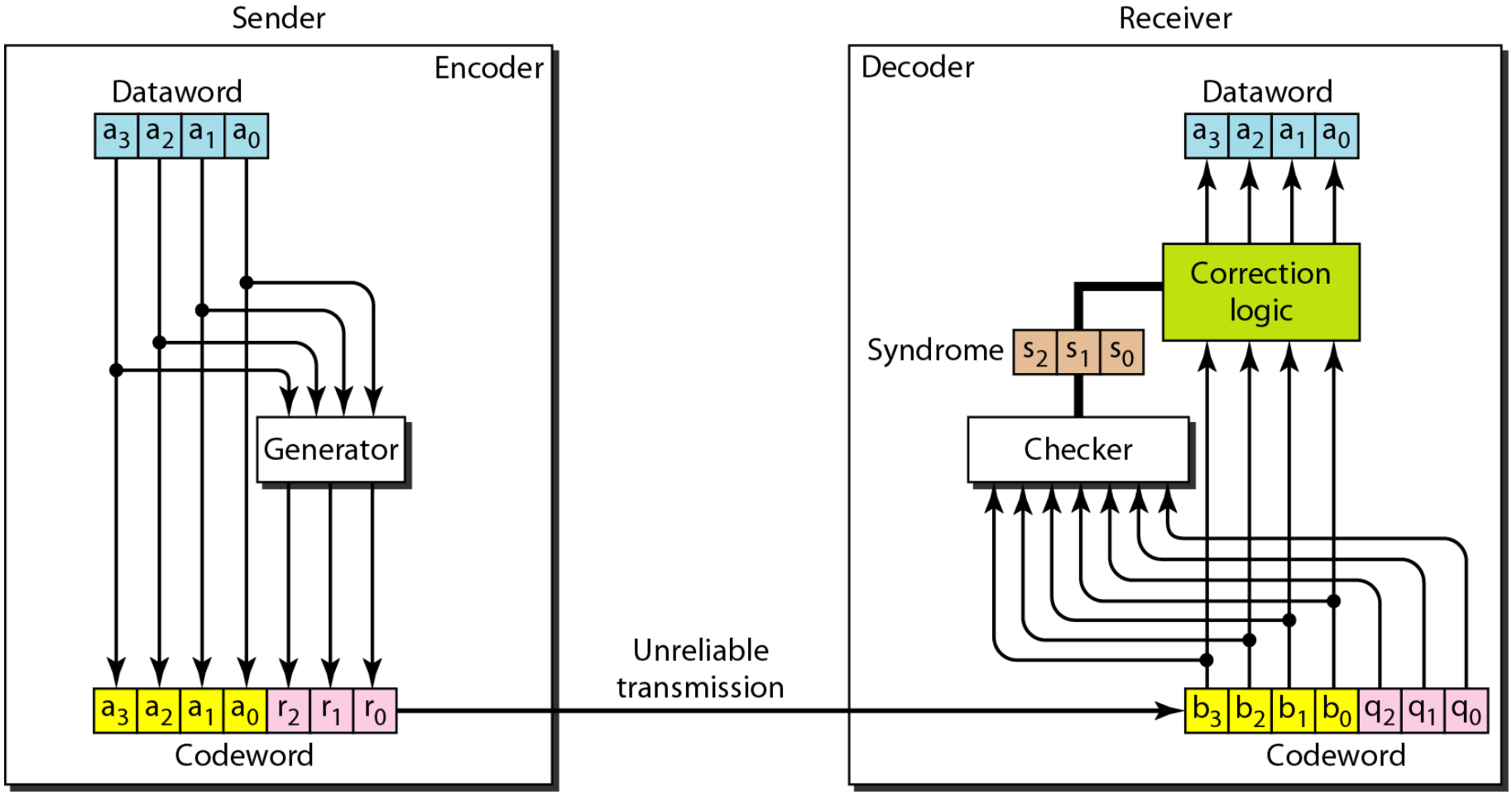
Note

A simple parity-check code can detect an odd number of errors.

Table 10.4 ***Hamming code C(7, 4)*** ***C(n,k)*** $d_{min}=3$

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Figure 10.12 *The structure of the encoder and decoder for a Hamming code*



Hamming Code

- Parity checks are created as follow (using modulo-2)
  - *r0 = a2 + a1 + a0*
  - *r1 = a3 + a2 + a1*
  - *r2 = a1 + a0 + a3*

Hamming Code

- The checker in the decoder creates a 3-bit syndrome (s2s1s0).
- In which each bit is the parity check for 4 out of the 7 bits in the received codeword:
- $s0 = b2 + b1 + b0 + q0$
- $s1 = b3 + b2 + b1 + q1$
- $s2 = b1 + b0 + b3 + q2$
- The equations used by the checker are the same as those used by the generator with the parity-check bits added to the right-hand side of the equation.

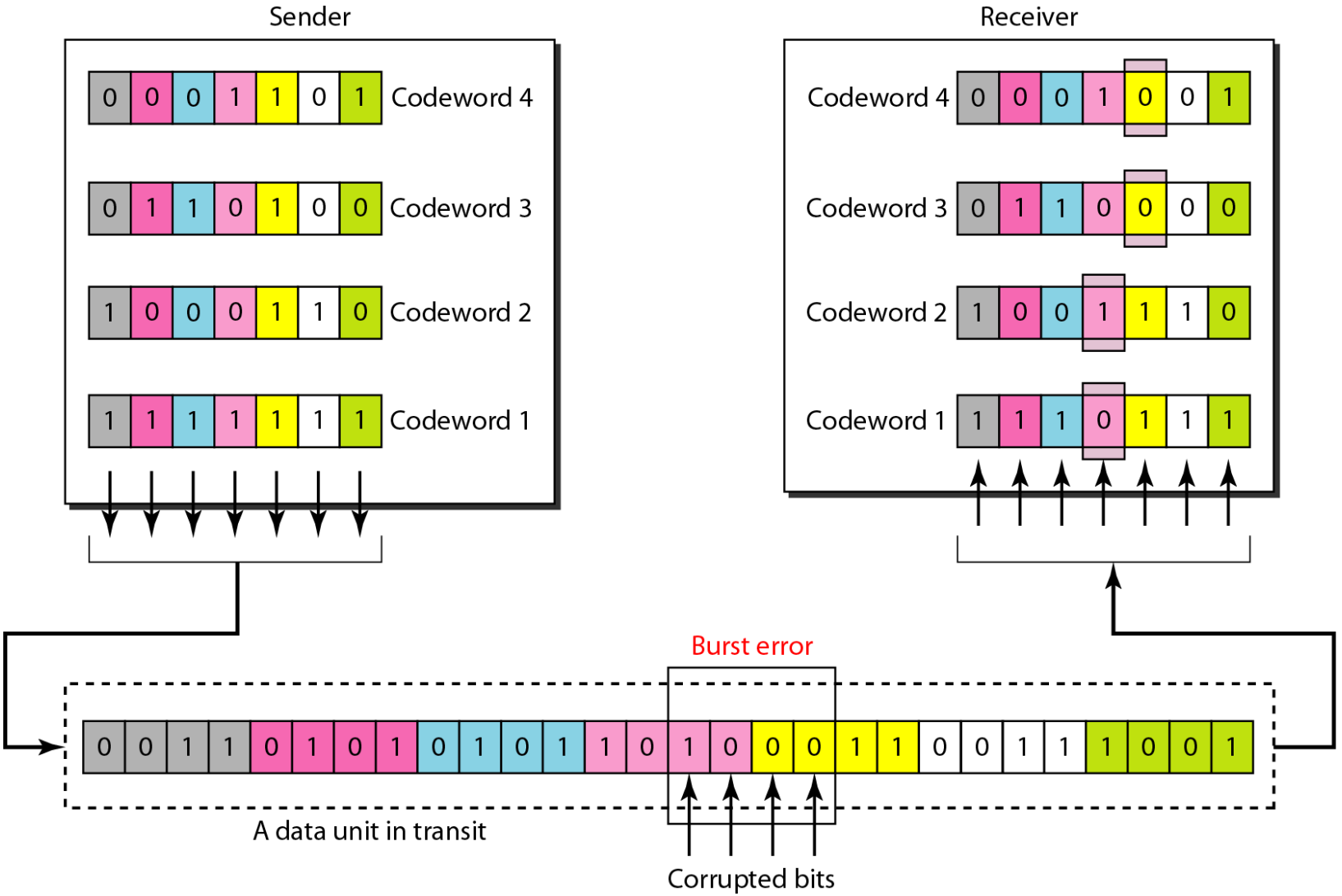
Table 10.5 ***Logical decision made by the correction logic analyzer***

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Hamming code C(7, 4) can :

- ***detect up to 2-bit error*** $(d_{min} - 1)$
- ***can correct up to 1 bit error*** $(d_{min} - 1)/2$

Figure 10.13 *Burst error correction using Hamming code*



Split burst error between multiple codewords

10-4 CYCLIC CODES

***Cyclic codes** are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.*

Topics discussed in this section:

Cyclic Redundancy Check
Hardware Implementation
Polynomials
Cyclic Code Analysis
Advantages of Cyclic Codes
Other Cyclic Codes

Table 10.6 *A CRC code with C(7, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 10.14 *CRC encoder and decoder*

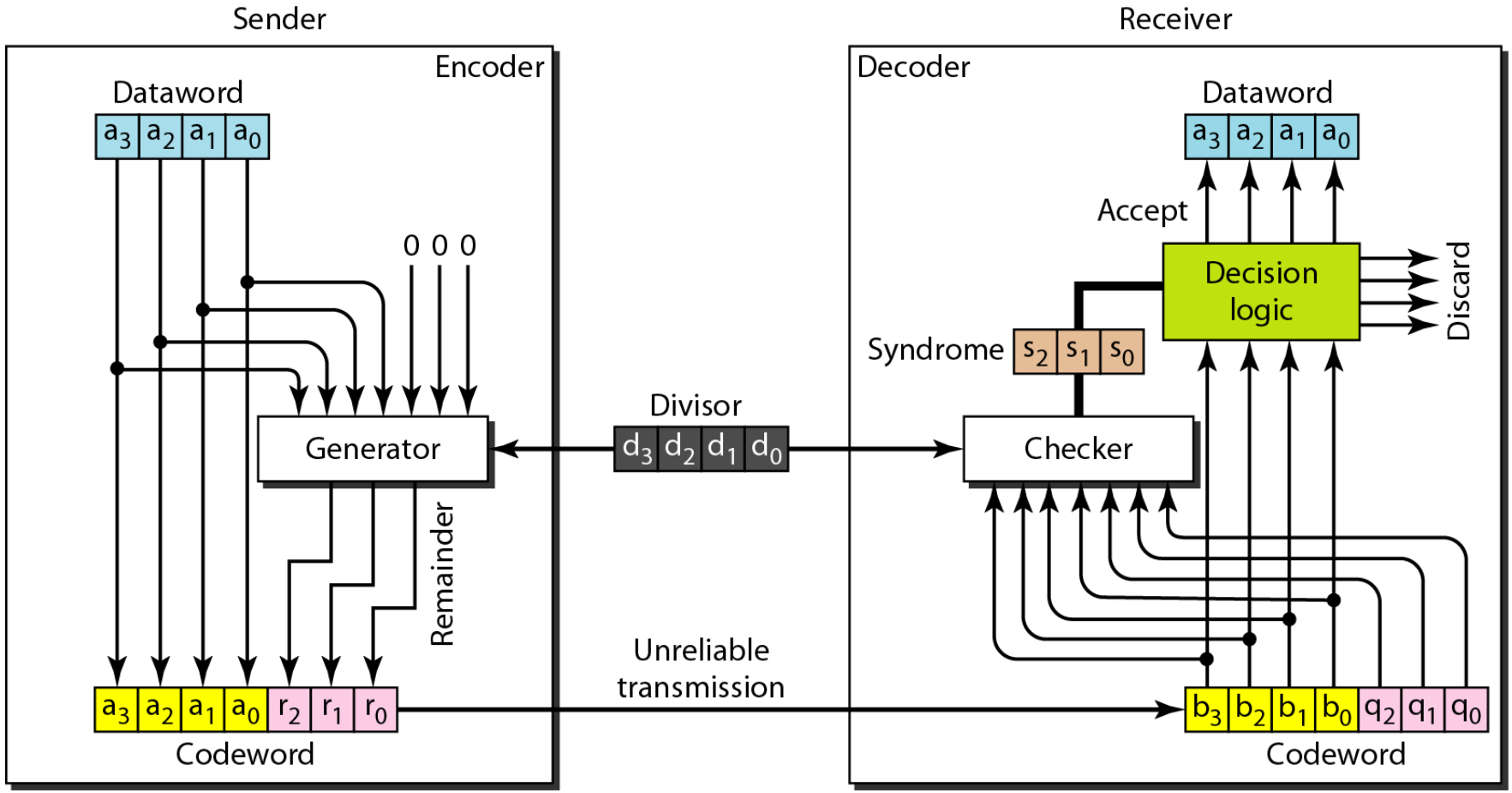


Figure 10.15 *Division in CRC encoder*

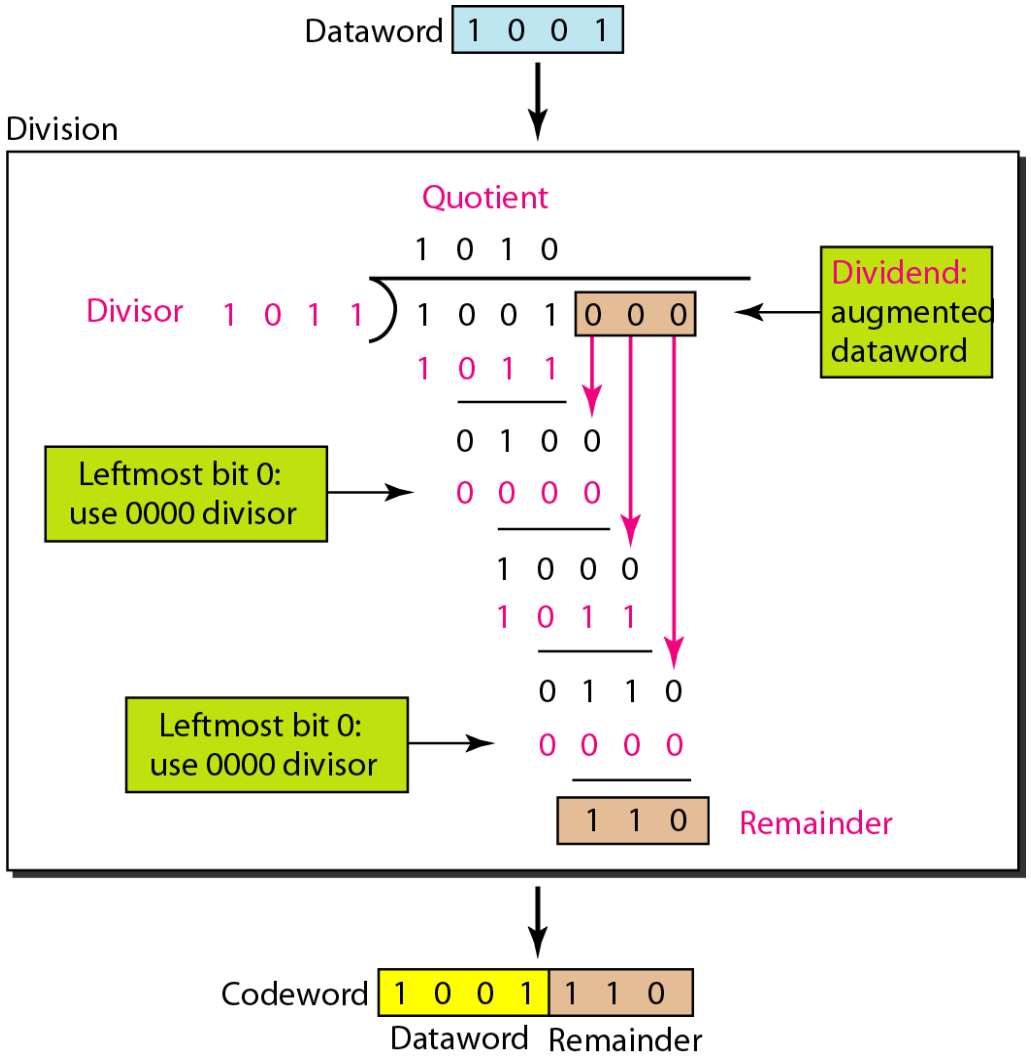


Figure 10.16 *Division in the CRC decoder for two cases*

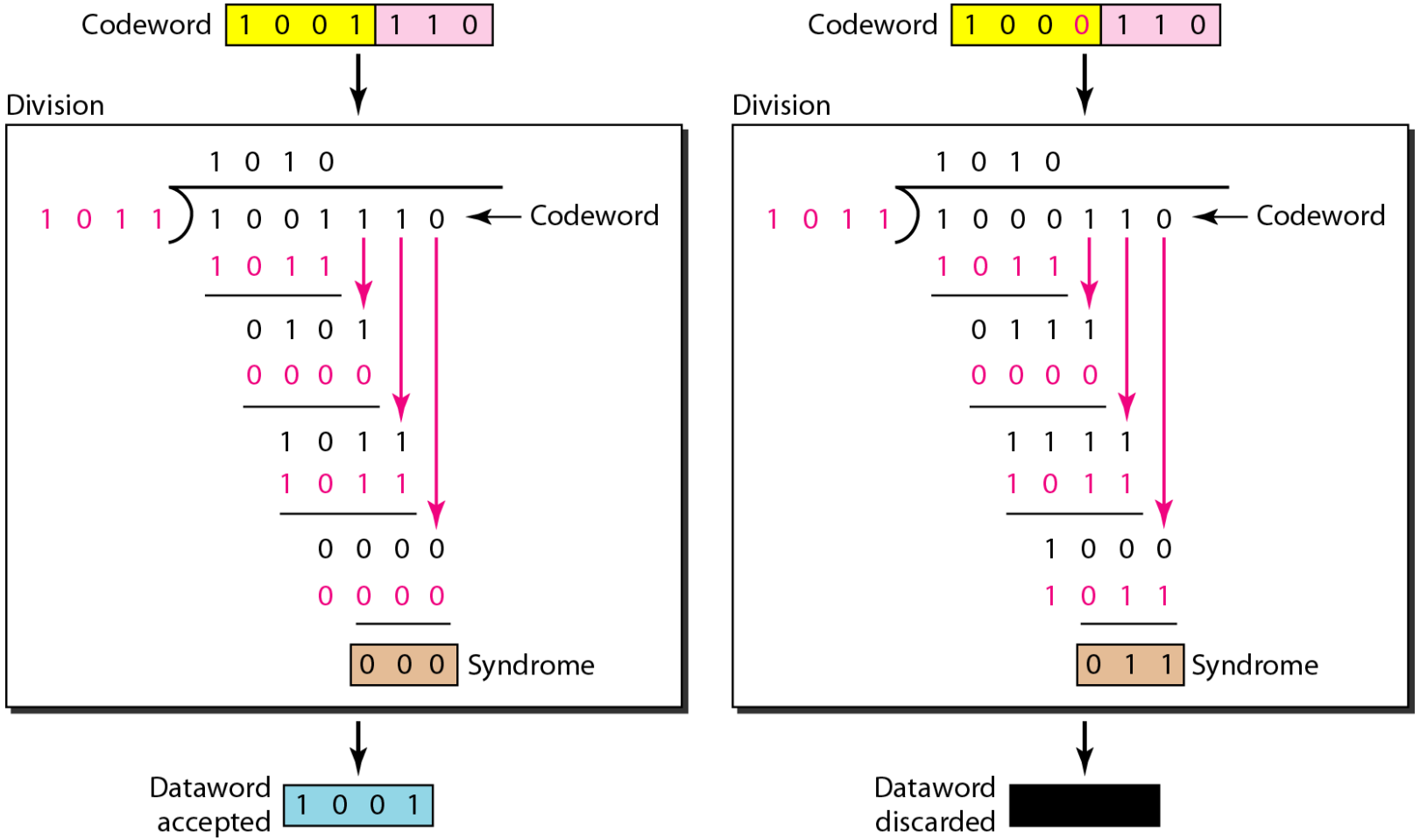
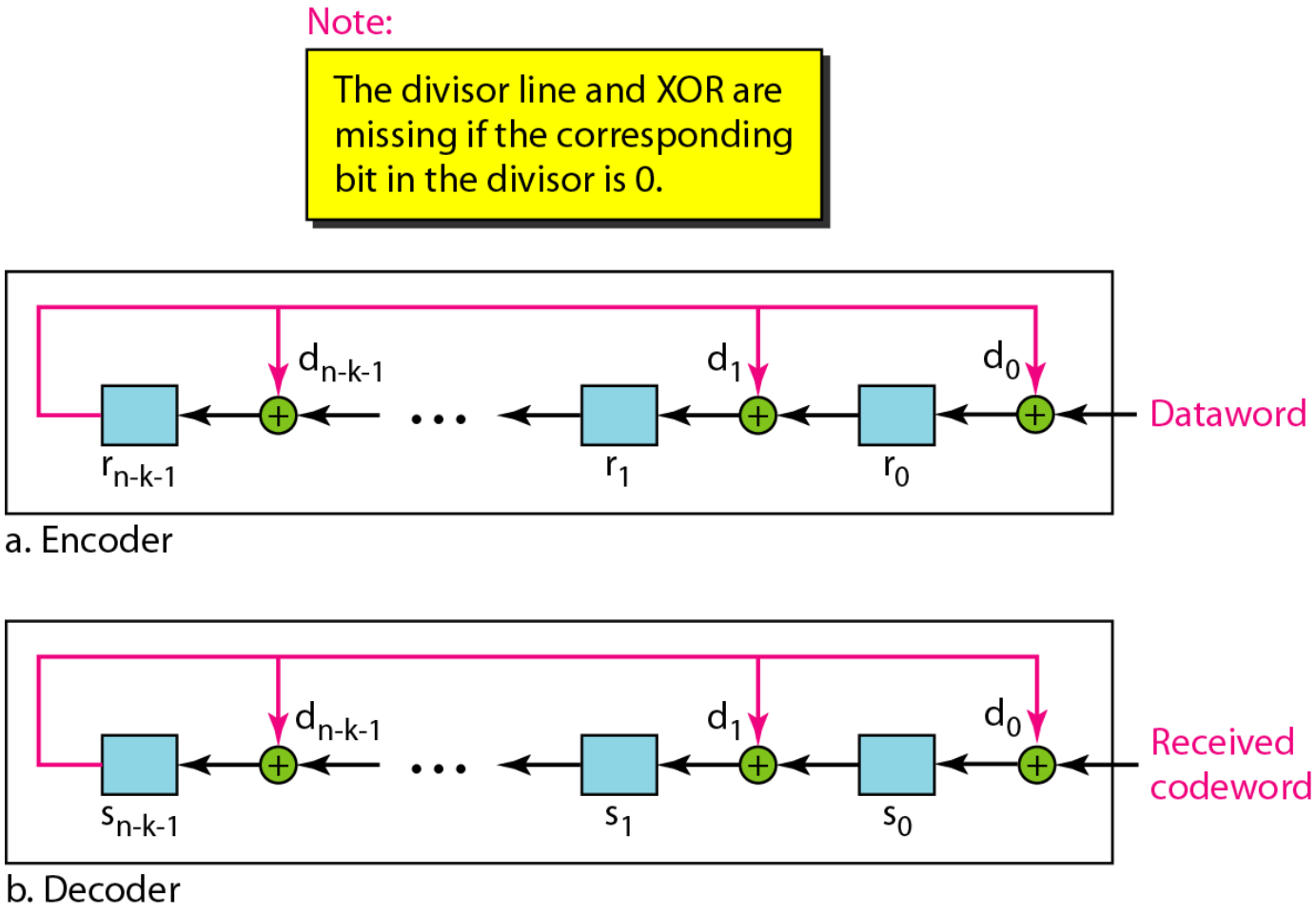


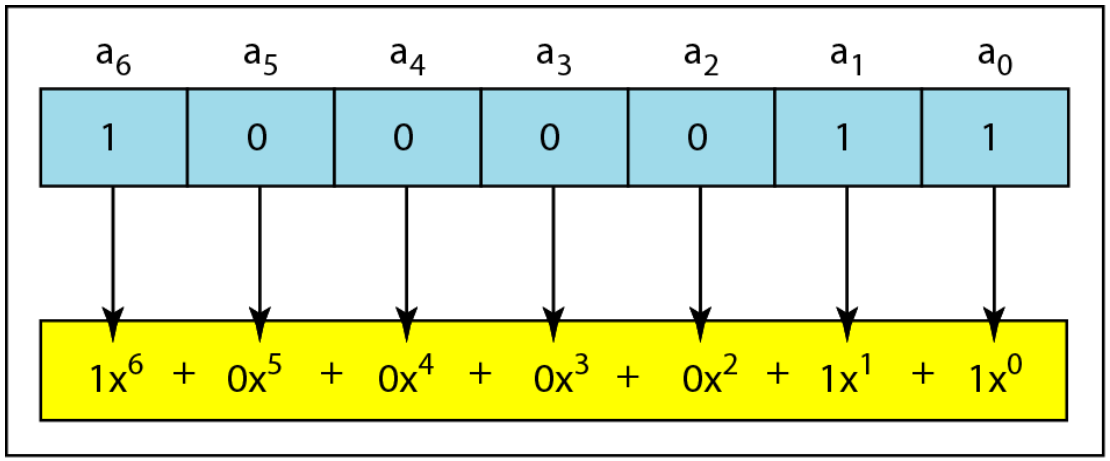
Figure 10.20 *General design of encoder and decoder of a CRC code*



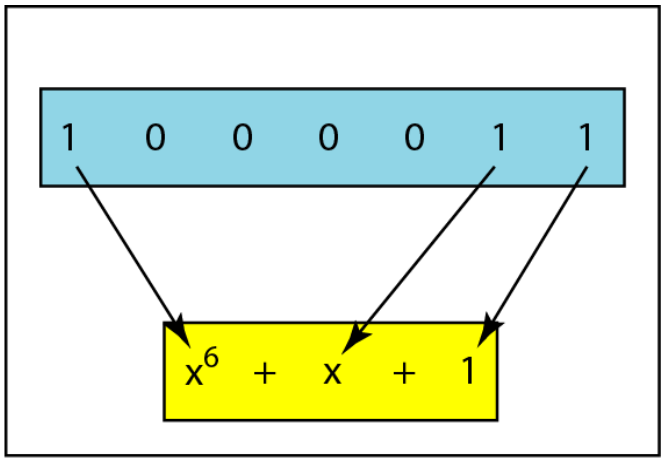
Note

The divisor in a cyclic code is normally called the generator polynomial or simply the generator.

Figure 10.21 *A polynomial to represent a binary word*



a. Binary pattern and polynomial



b. Short form

Table 10.7 *Standard polynomials*

Name	*Polynomial*	*Application*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

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. However, we briefly discuss it here to complete our discussion on error checking

Topics discussed in this section:

Idea
One's Complement
Internet Checksum

Example 10.18

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Example 10.19

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

Example 10.20

How can we represent the number 21 in one's complement arithmetic using only four bits?

Solution

The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to the four rightmost bits. We have (0101 + 1) = 0110 or 6.

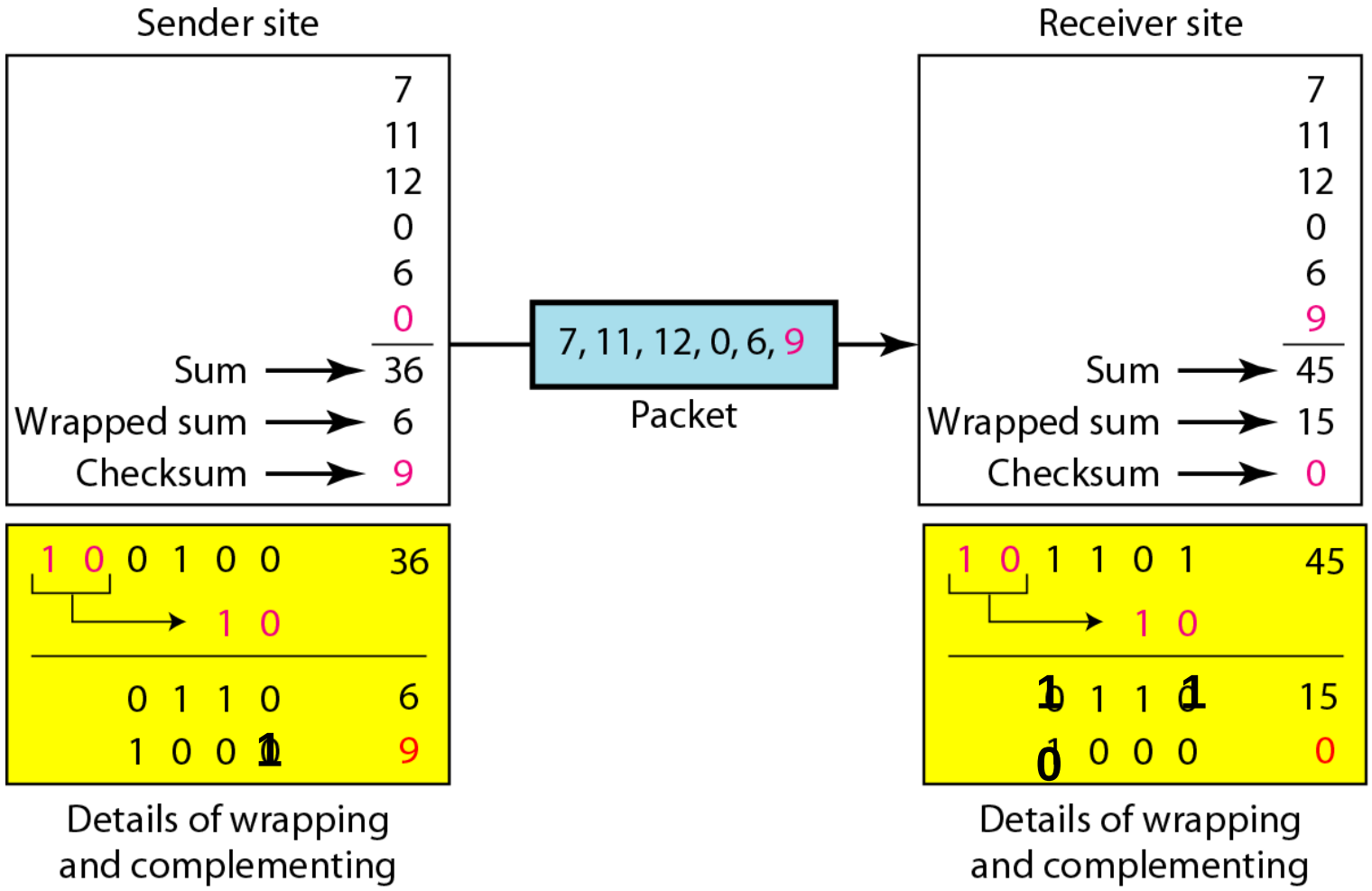
Example 10.21

How can we represent the number −6 in one's complement arithmetic using only four bits?

Solution

In one's complement arithmetic, the negative or complement of a number is found by inverting all bits. Positive 6 is 0110; negative 6 is 1001. If we consider only unsigned numbers, this is 9. In other words, the complement of 6 is 9.

Figure 10.24 *Example 10.22*



Note

Sender site:

1. **The message is divided into 16-bit words.**
2. **The value of the checksum word is set to 0.**
3. **All words including the checksum are added using one's complement addition.**
4. **The sum is complemented and becomes the checksum.**
5. **The checksum is sent with the data.**

Note

Receiver site:

1. **The message (including checksum) is divided into 16-bit words.**
2. **All words are added using one's complement addition.**
3. **The sum is complemented and becomes the new checksum.**
4. **If the value of checksum is 0, the message is accepted; otherwise, it is rejected.**

Internet Checksum Example

- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers