Data Communications and Networking Fourth Edition

Forouzan

Chapter 10

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

Figure 10.1 *Single-bit error*

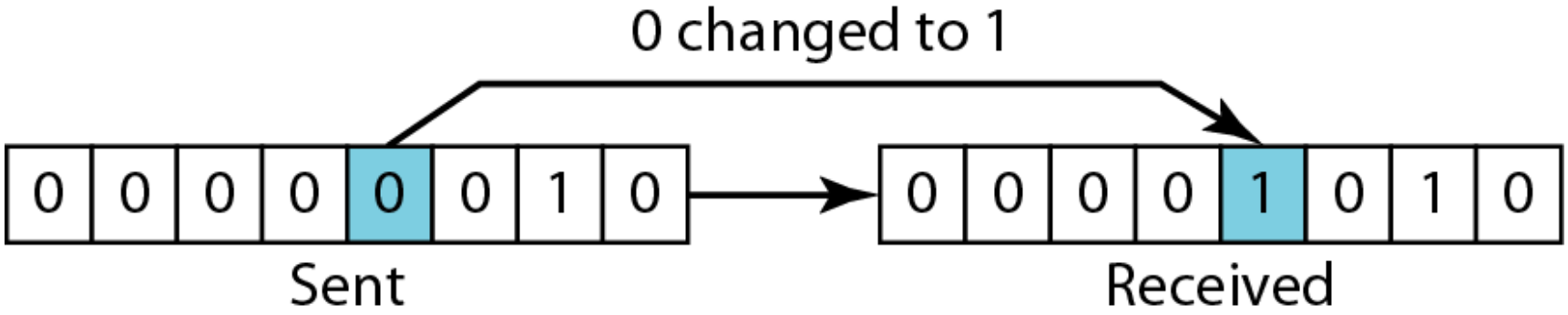


Figure 10.2 *Burst error of length 8*

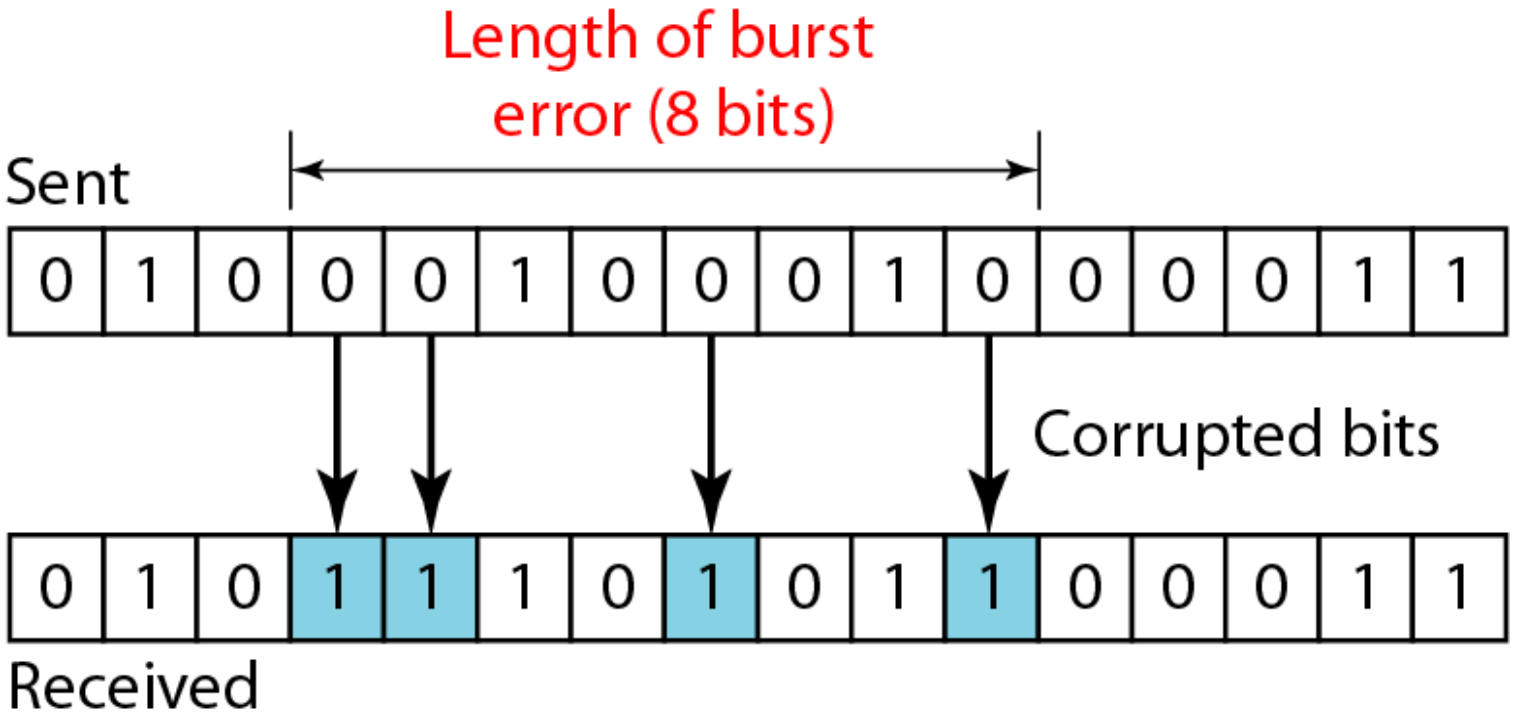
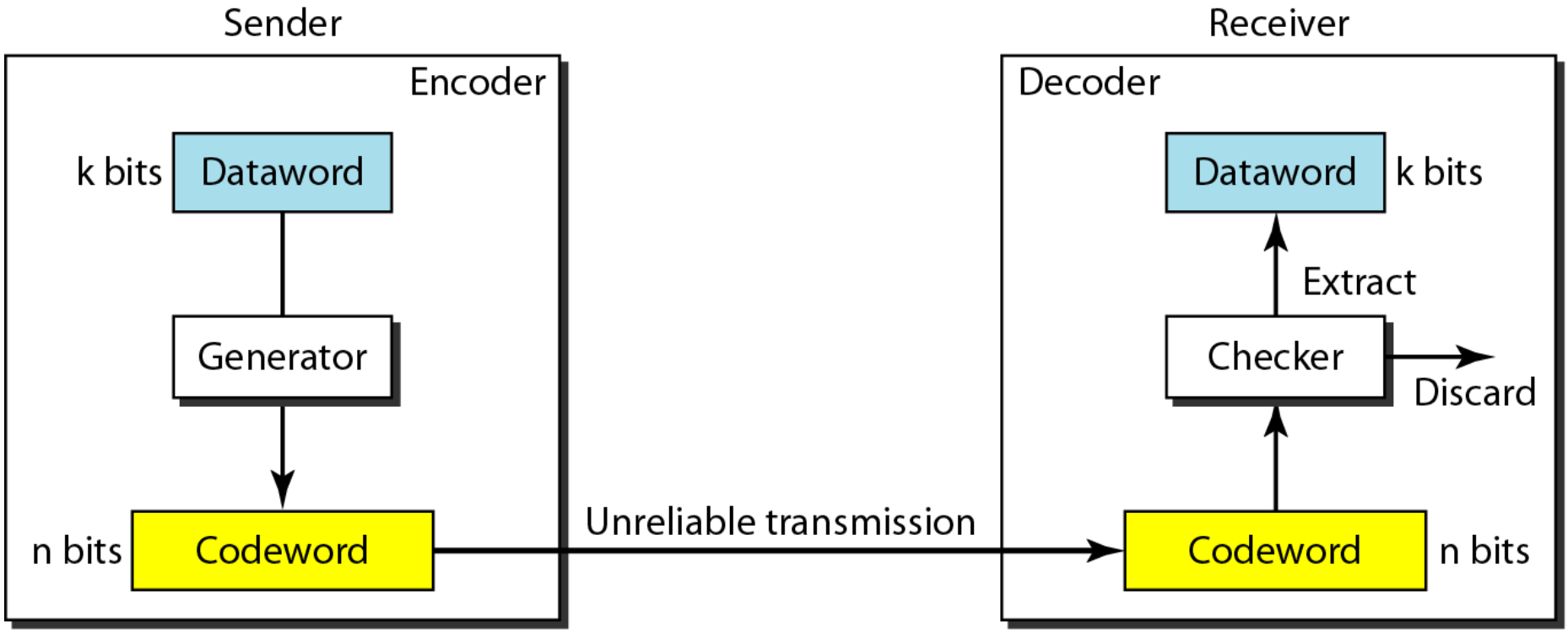


Figure 10.3 *The structure of encoder and decoder*



To detect or correct errors, we need to send redundant bits

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

Example 10.1

The 4B/5B block coding discussed in Chapter 4 is a good example of this type of coding. In this coding scheme, $k = 4$ and $n = 5$. As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.

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

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

What if we want to send 01? We code it as 011. If 011 is received, no problem.
What if 001 is received? Error detected.
What if 000 is received? Error occurred, but not detected.

Let's add more redundant bits to see if we can correct error.

Table 10.2 *A code for error correction (Example 10.3)*

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Let's say we want to send 01. We then transmit 01011. What if an error occurs and we receive 01001. If we assume one bit was in error, we can correct.

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.*

Example 10.6

Find the minimum Hamming distance of the coding scheme in Table 10.2.

Solution

We first find all the Hamming distances.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Note

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

Example 10.7

The minimum Hamming distance for our first code scheme (Table 10.1) is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Example 10.8

Our second block code scheme (Table 10.2) has $d_{min} = 3$. This code can detect up to two errors. Again, we see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.

However, some combinations of three errors change a valid codeword to another valid codeword. The receiver accepts the received codeword and the errors are undetected.

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 need to 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 exclusive OR (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

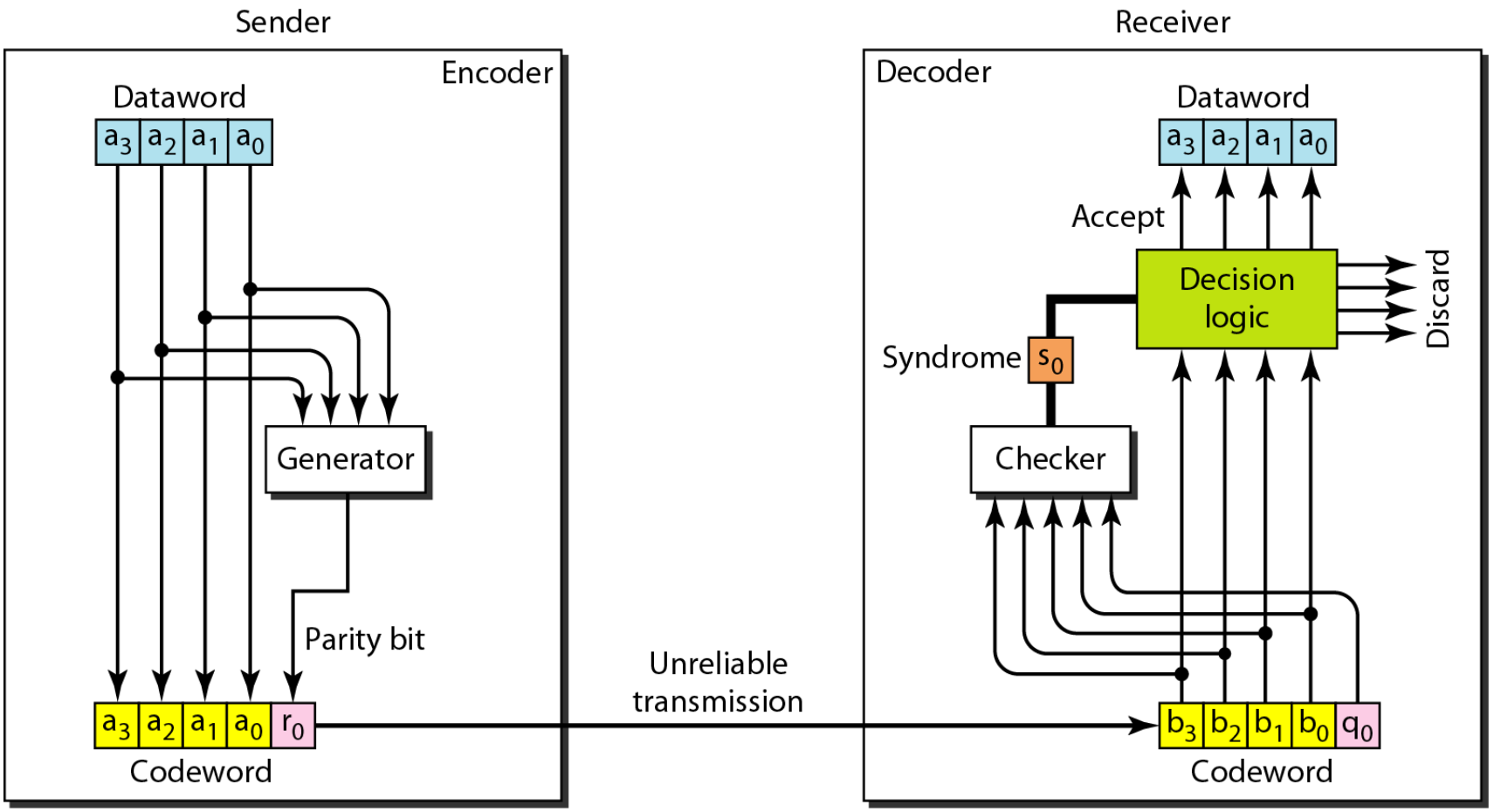
Note

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*



Example 10.12

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver. We examine five cases:

1. *No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.*
2. *One single-bit error changes a_1. The received codeword is 10011. The syndrome is 1. No dataword is created.*
3. *One single-bit error changes r_0. The received codeword is 10110. The syndrome is 1. No dataword is created.*

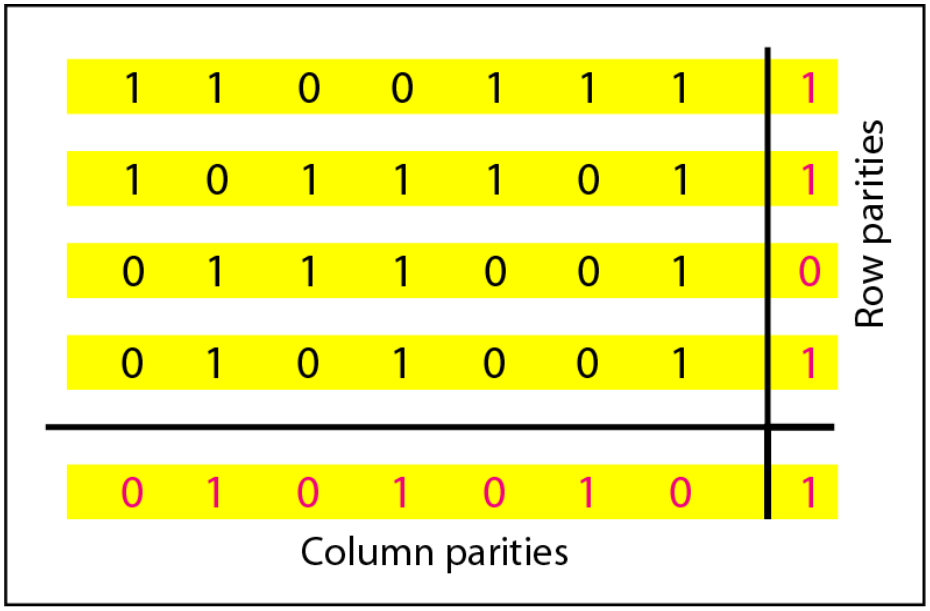
Example 10.12 (continued)

4. An error changes r_0 and a second error changes a_3. The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value.
5. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Note

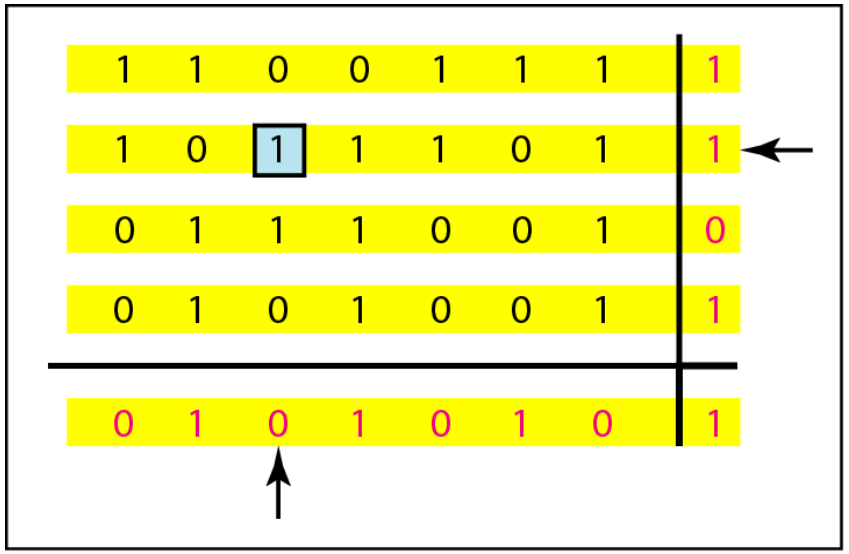
A simple parity-check code can detect an odd number of errors.

Figure 10.11 *Two-dimensional parity-check code*

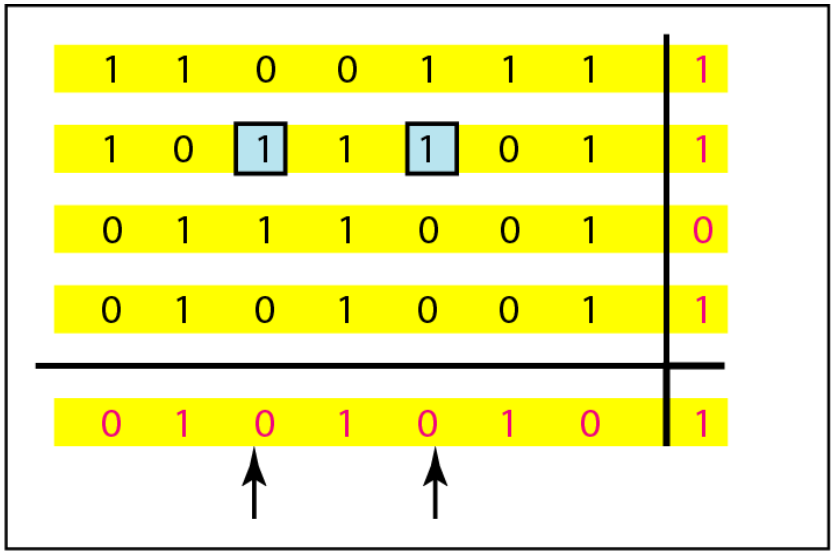


a. Design of row and column parities

Figure 10.11 *Two-dimensional parity-check code*

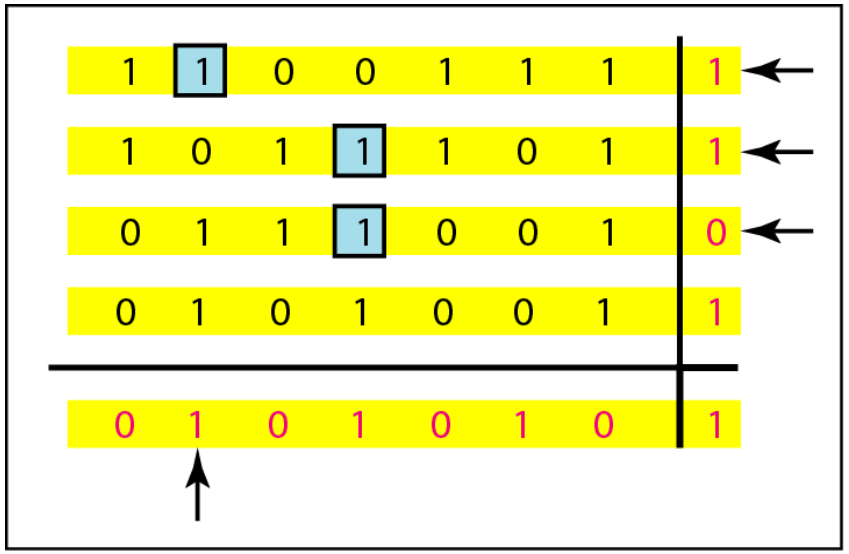


b. One error affects two parities

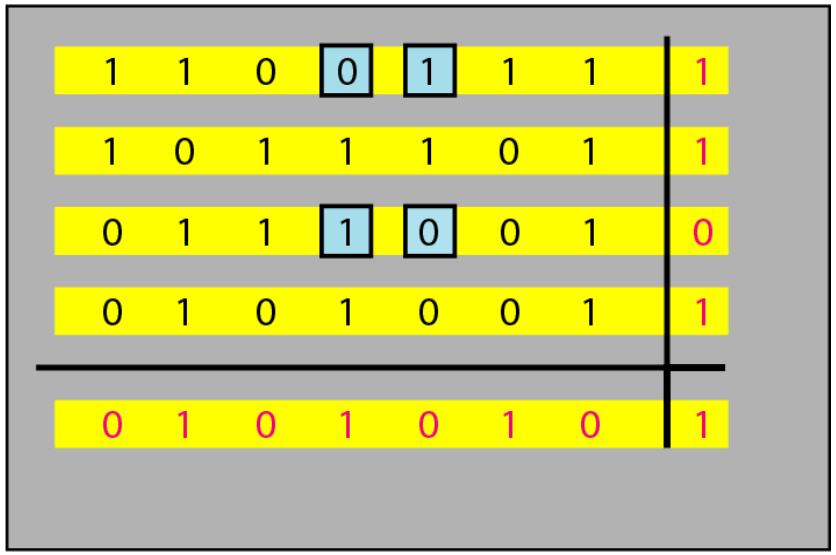


c. Two errors affect two parities

Figure 10.11 *Two-dimensional parity-check code*



d. Three errors affect four parities



e. Four errors cannot be detected

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

Cyclic Redundancy Checksum

The CRC error detection method treats the packet of data to be transmitted as a large polynomial.

The transmitter takes the message polynomial and using polynomial arithmetic, divides it by a given generating polynomial.

The quotient is discarded but the remainder is “attached” to the end of the message.

Cyclic Redundancy Checksum

The message (with the remainder) is transmitted to the receiver.

The receiver divides the message and remainder by the same generating polynomial.

If a remainder not equal to zero results, there was an error during transmission.

If a remainder of zero results, there was no error during transmission.

More Formally

- M(x) - original message treated as a polynomial
- To prepare for transmission:
- Add r 0s to end of the message (where r = degree of generating polynomial)
- Divide $M(x)x^r$ by generating polynomial P(x) yielding a quotient and a remainder Q(x)+R(x)/P(x).

More Formally

- Add (XOR) remainder R(x) to $M(x)x^r$ giving $M(x)x^r+R(x)$ and transmit.
- Receiver receives message ($M(x)x^r+R(x)$) and divides by *same* P(x).
- If remainder is 0, then there were no errors during transmission.
- (Any expression which has exactly P(x) as a term is evenly divisible by P(x).)

Table 6-4 *Error detection performance of cyclic redundancy checksum*

Type of Error	Error Detection Performance
Single bit errors	100 percent
Double bit errors	100 percent, as long as the generating polynomial has at least three 1s (they all do)
Odd number of bits in error	100 percent, as long as the generating polynomial contains a factor $x + 1$ (they all do)
An error burst of length $< r+1$	100 percent
An error burst of length $= r+1$	probability $= 1 - (½)^{(r-1)}$
An error burst of length $> r+1$	probability $= 1 - (½)^{r}$

Common CRC Polynomials

- **CRC-12:** $x^{12} + x^{11} + x^3 + x^2 + x + 1$
- **CRC-16:** $x^{16} + x^{15} + x^2 + 1$
- **CRC-CCITT:** $x^{16} + x^{15} + x^5 + 1$
- **CRC-32:** $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- **ATM CRC:** $x^8 + x^2 + x + 1$

CRC Example

- **Given a pretend $P(x) = x^5 + x^4 + x^2 + 1$ and a message M(x) = 1010011010, calculate the remainder using long hand division and a shift register.**

10-5 CHECKSUM

The last error detection method we discuss here is called the checksum, or arithmetic 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. Another way to find the complement of a number in one's complement arithmetic is to subtract the number from $2^n - 1$ ($16 - 1$ in this case).

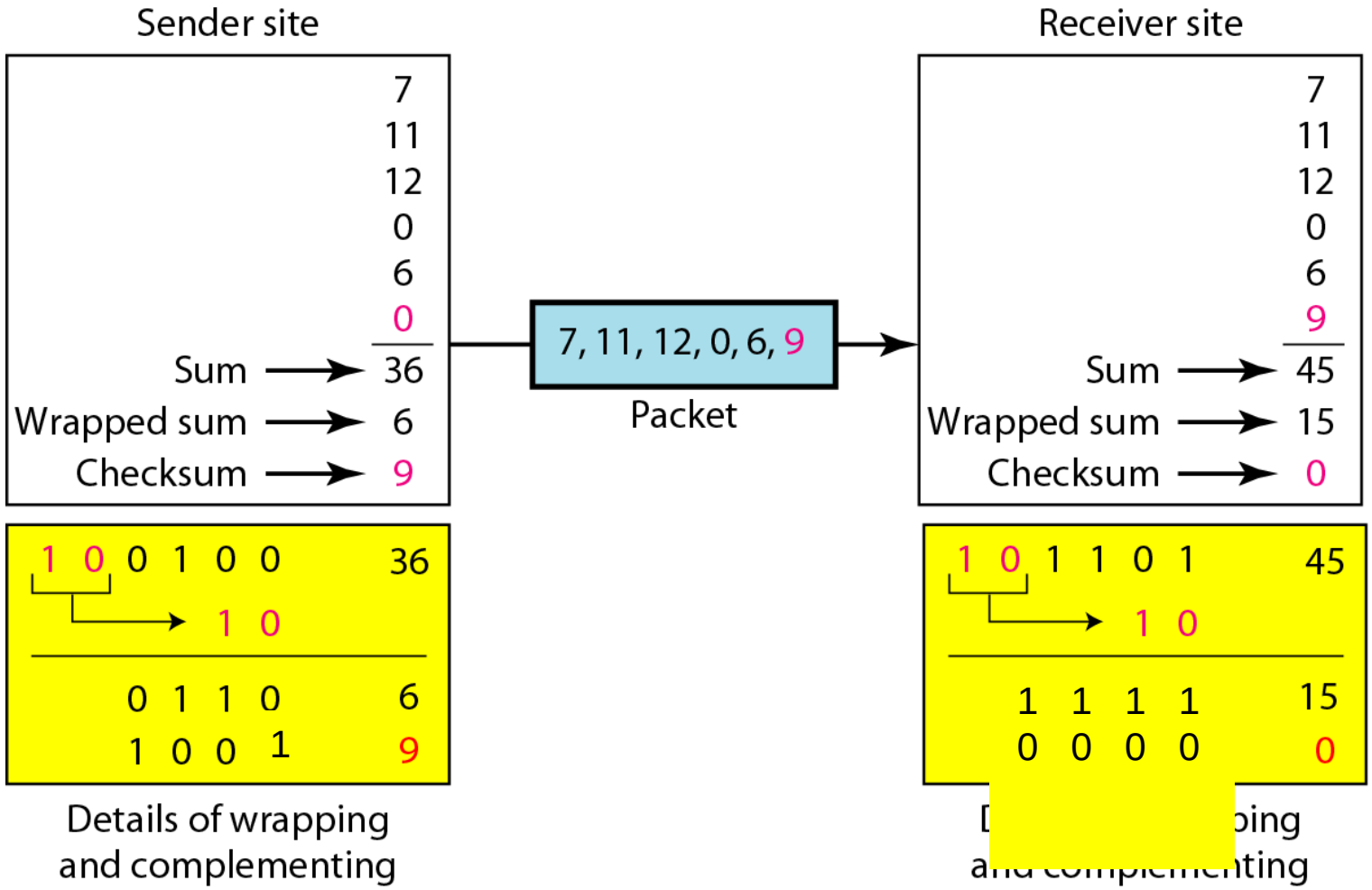
Example 10.22

Let us redo Exercise 10.19 using one's complement arithmetic. Figure 10.24 shows the process at the sender and at the receiver. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then complemented, resulting in the checksum value 9 (15 − 6 = 9). The sender now sends six data items to the receiver including the checksum 9.

Example 10.22 (continued)

The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.

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

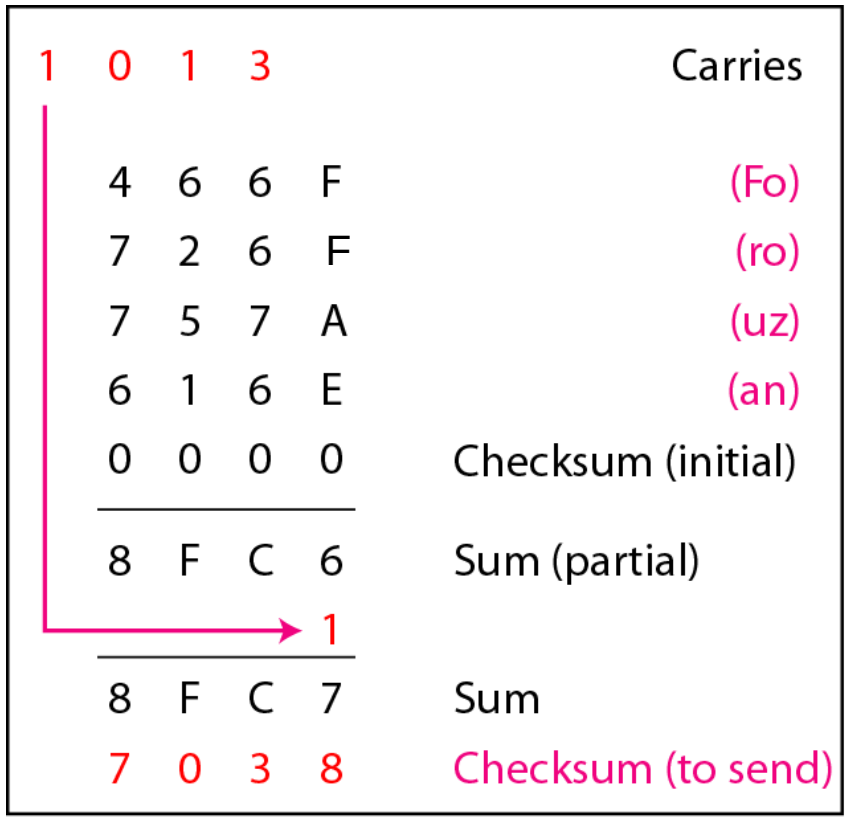
Receiver site:

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.

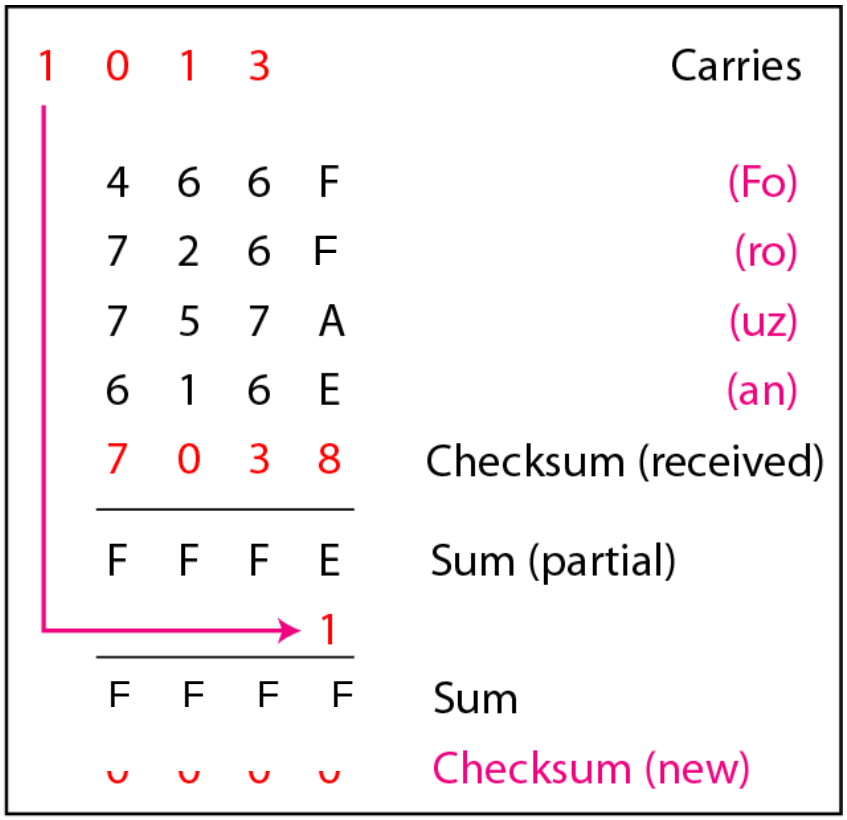
Example 10.23

Let us calculate the checksum for a text of 8 characters ("Forouzan"). The text needs to be divided into 2-byte (16-bit) words. We use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as 0x46 and o is represented as 0x6F. Figure 10.25 shows how the checksum is calculated at the sender and receiver sites. In part a of the figure, the value of partial sum for the first column is 0x36. We keep the rightmost digit (6) and insert the leftmost digit (3) as the carry in the second column. The process is repeated for each column. Note that if there is any corruption, the checksum recalculated by the receiver is not all 0s. We leave this an exercise.

Figure 10.25 *Example 10.23*



a. Checksum at the sender site



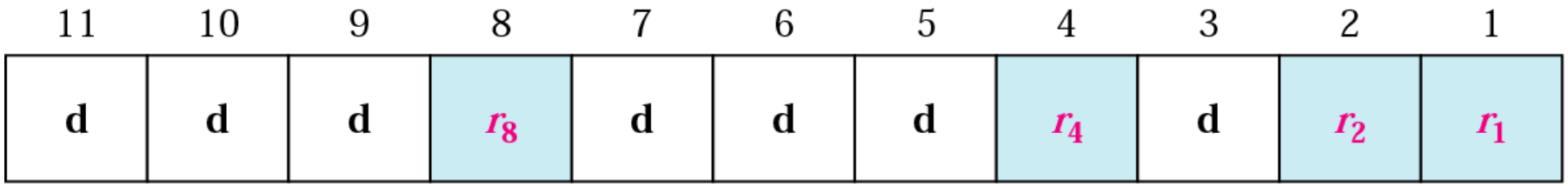
a. Checksum at the receiver site

Error Correcting Codes
or
Forward Error Correction (FEC)

FEC is used in transmission of radio signals, such as those used in transmission of digital television (Reed-Solomon and Trellis encoding) and 4D-PAM5 (Viterbi and Trellis encoding)

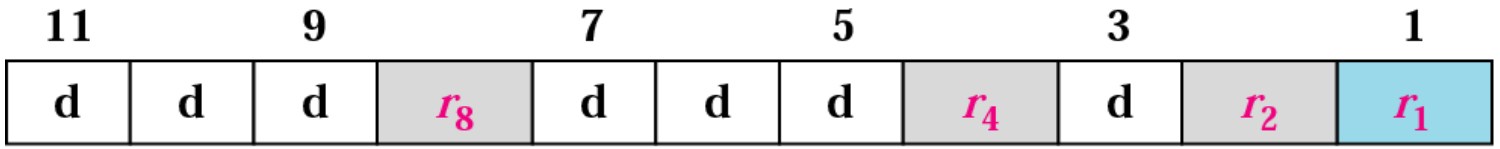
Some FEC is based on Hamming Codes

Let's examine a Hamming Code

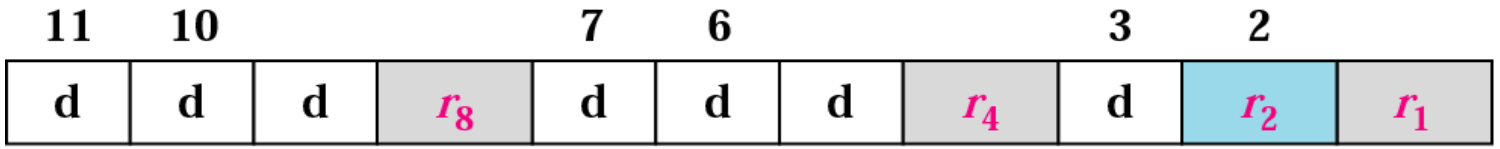


From previous edition of Forouzan

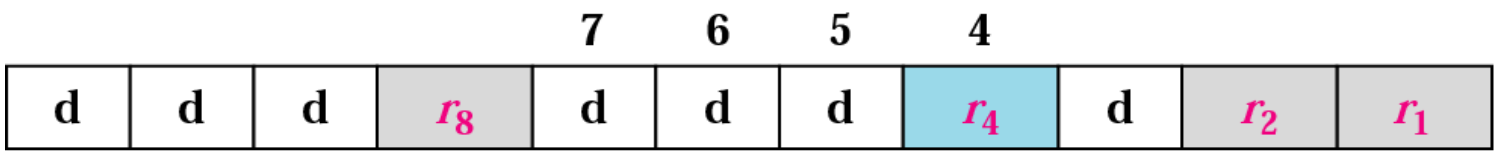
r_1 will take care of these bits.



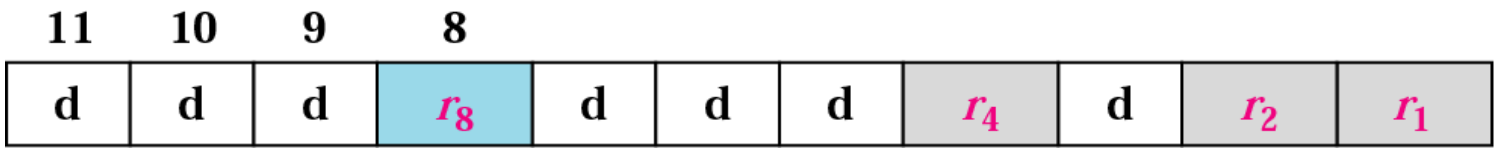
r_2 will take care of these bits.



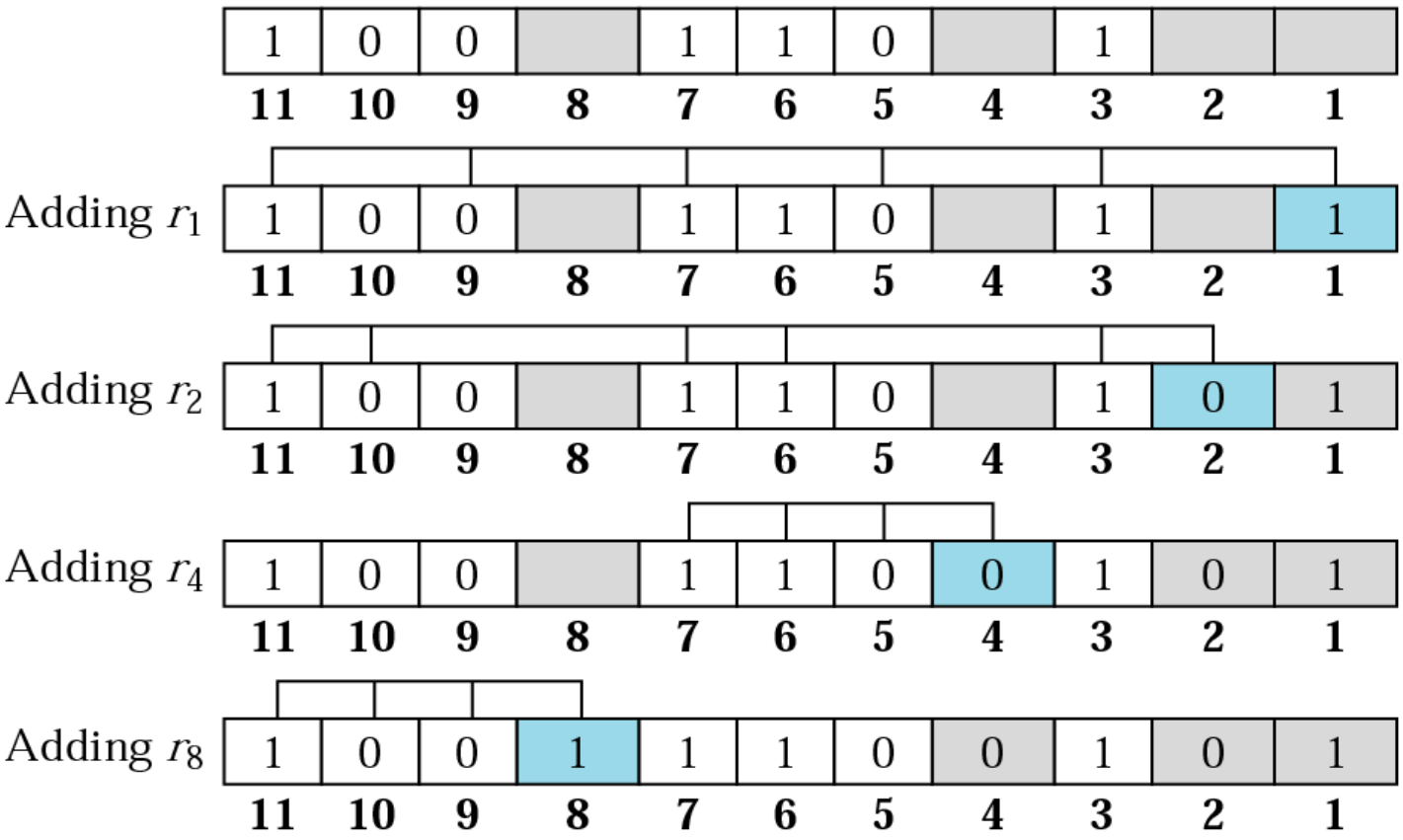
r_4 will take care of these bits.



r_8 will take care of these bits.

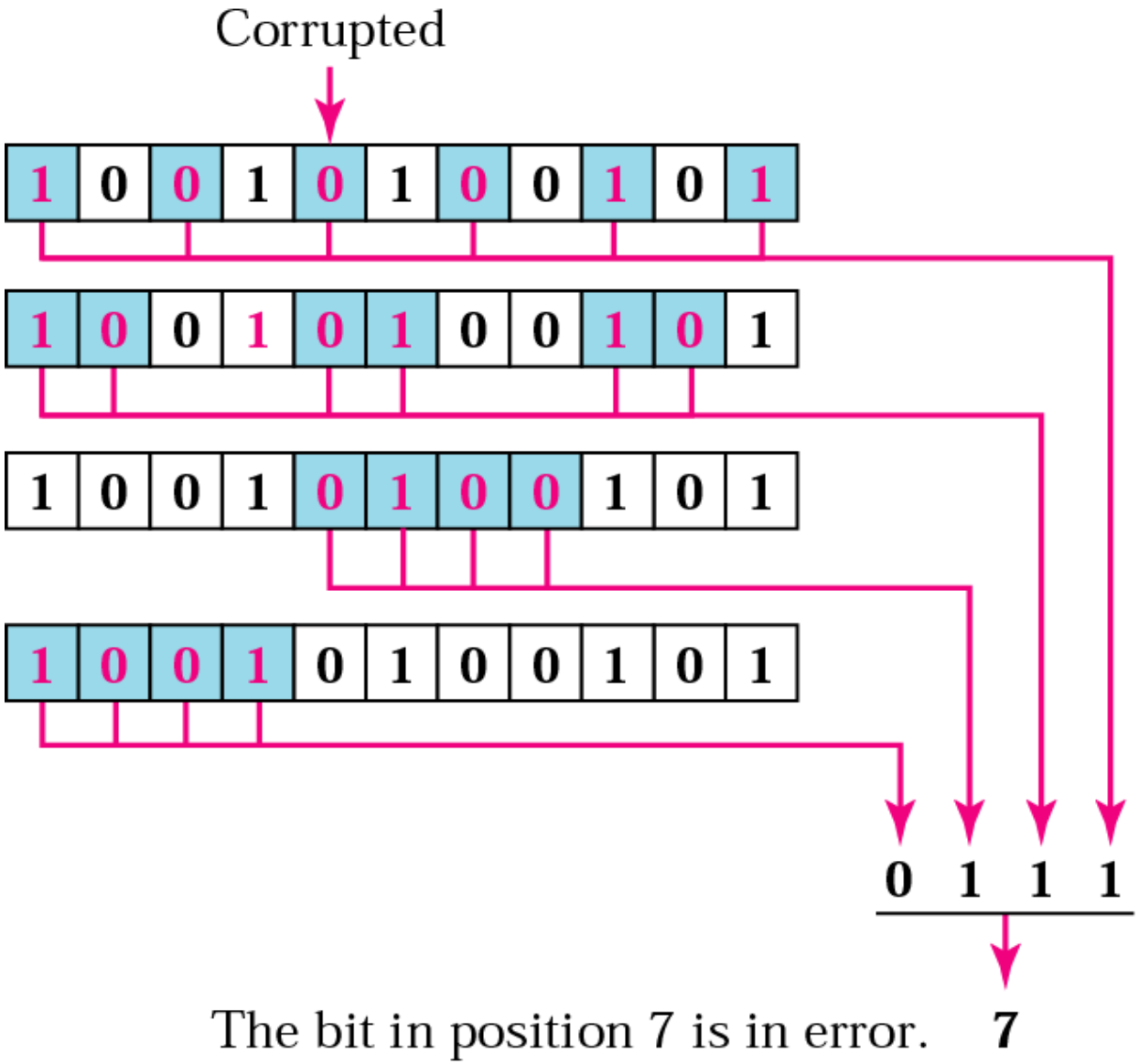


Example of redundancy bit calculation



Data:
1 0 0 1 1 0 1

Code:
1 0 0 1 1 1 0 0 1 0 1


Corrupted
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
0 1 1 1
The bit in position 7 is in error. 7

Review Questions

- What is the efficiency of simple parity?
- What is the efficiency of CRC?
- Be able to calculate a CRC using either shift register method or long-hand division method
- Be able to encode a character using a Hamming code, then decode, detecting and correcting an error