Module 3
Data Link Layer
Logical Link Control

BECE401L

Outline

Error Detection Techniques

ARQ protocols

Framing

HDLC

Point to Point protocol

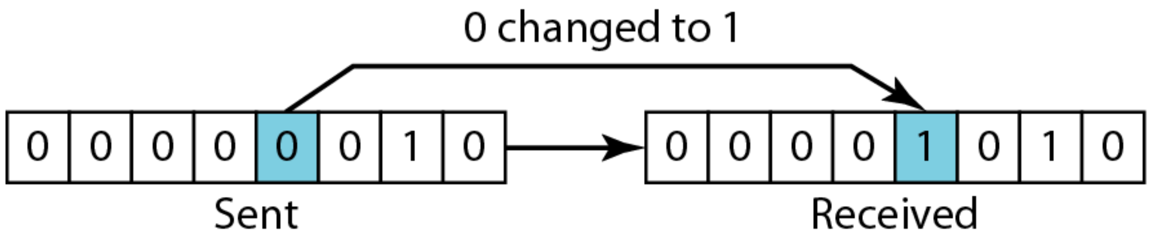
Error Detection & Correction

Data can be corrupted during transmission.

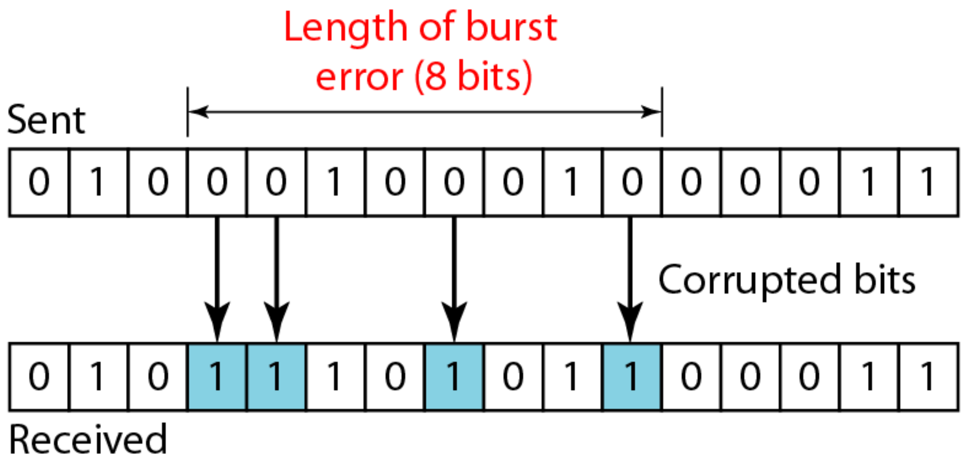
Some applications require errors be detected and corrected.

Type of Error:

- Single Bit Error
- Burst Error



Burst error of length 8

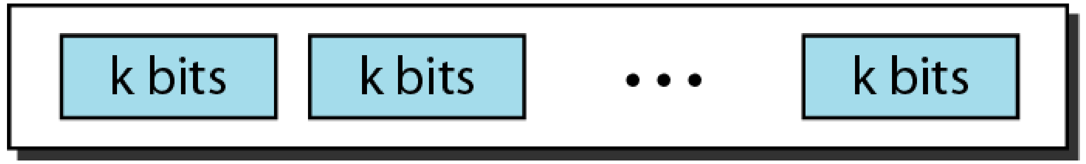


Block Coding

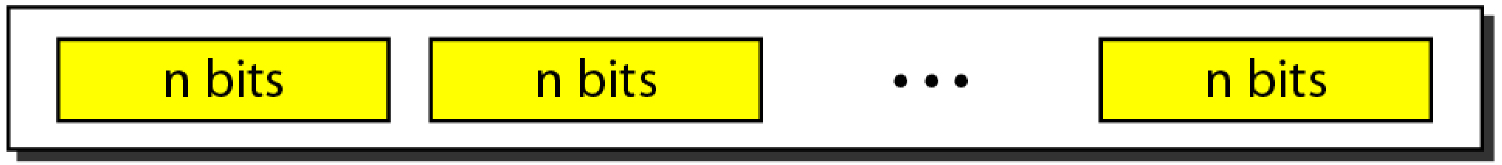
Divide our message into blocks, each of k bits, called datawords.

We add *r redundant bits* to each block to make the *length n = k + r*.

The resulting n-bit blocks are called codewords.



2^k Datawords, each of k bits



2^n Codewords, each of n bits (only 2^k of them are valid)

Block Coding

Example:

The 4B/5B block coding is a good example of this type of coding. In this coding scheme, $k = 4$ and $n = 5$.

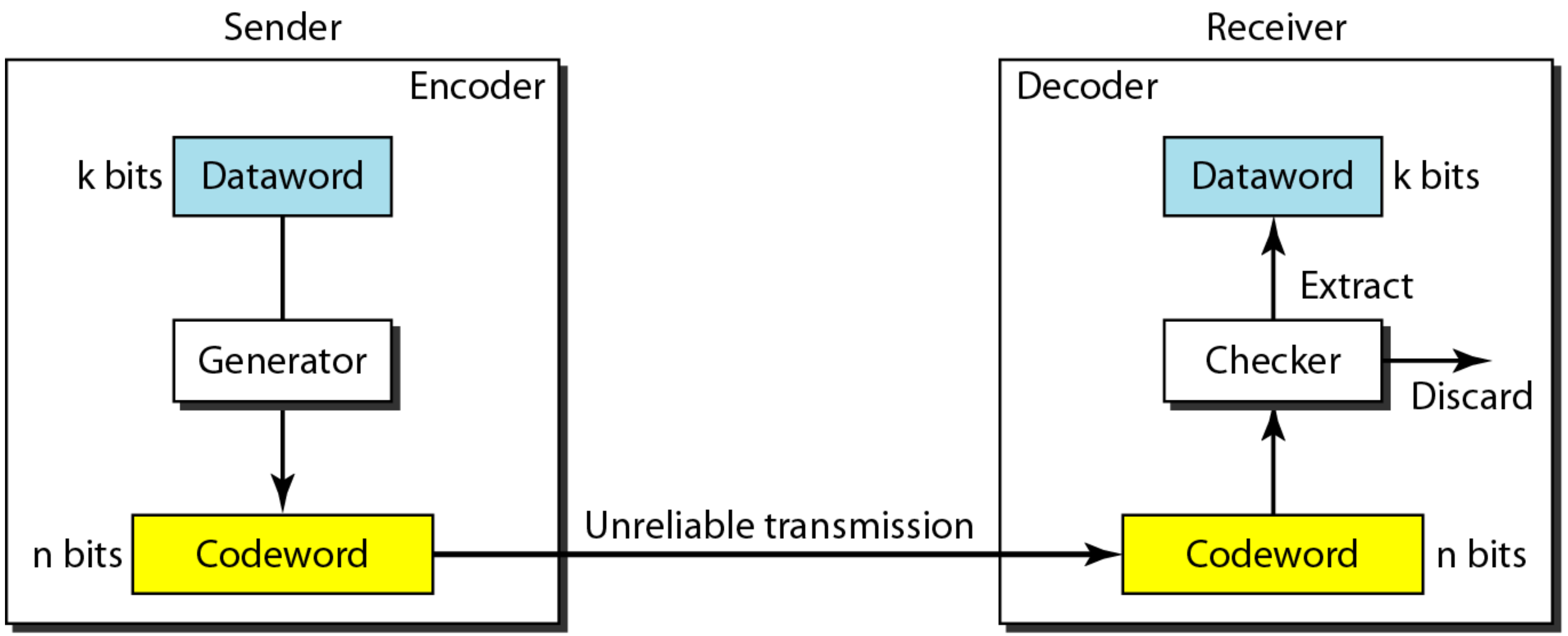
As we saw, we have $2^k = 16$ data words and $2^n = 32$ codewords.

We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.

Error Detection

Error Detection can be done:

- The receiver has (or can find) a list of valid codewords.
- The original codeword has changed to an invalid one.



Error Detection : Example

Let us assume that k = 2 and n = 3. **Table** shows the list of data words and codewords. Later, we will see how to derive a codeword from a data word.

Assume the sender encodes the data word 01 as 011 and sends it to the receiver.

Consider the following cases:

1. The receiver receives 011. It is a **valid codeword**. The receiver extracts the data word 01 from it.

2. The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is **discarded**.

3. The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the data word 00. Two corrupted bits have made the error undetectable.

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

Hamming Distance

Hamming distance between two words (same size)

- Is the number of differences between corresponding bits.
- Shown as $d(x, y)$
- Gives info of the number of bits that are corrupted

Example

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

$$000 \oplus 011 \text{ is } 011 \text{ (two 1s)}$$

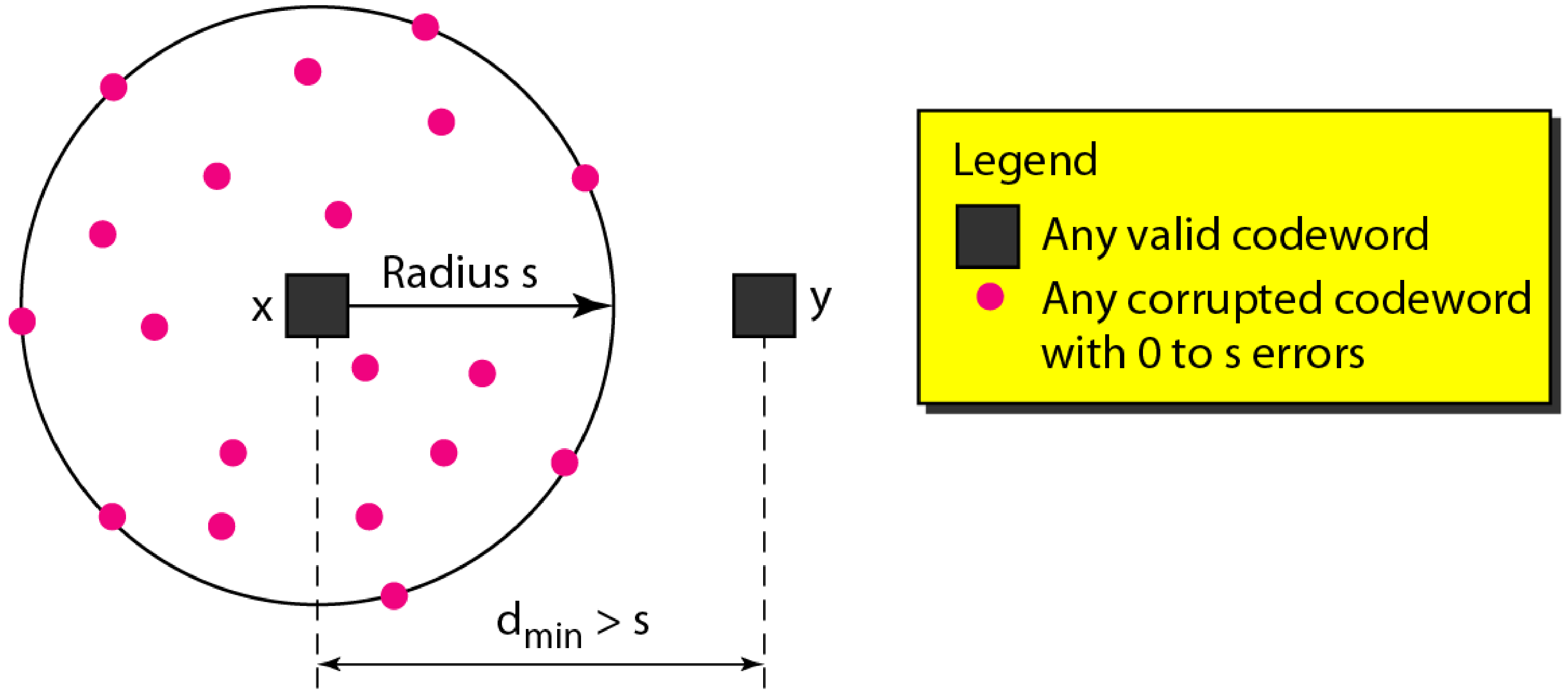
2. The Hamming distance d(10101, 11110) is 3 because

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

Minimum Hamming Distance

The smallest Hamming distance between all possible pairs of codewords.

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



Minimum Hamming Distance

Find the minimum Hamming distance of the coding scheme in

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

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

Minimum Hamming Distance

Find the minimum Hamming distance of the coding scheme in

d_{min} = 3.

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

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

Linear Block Codes

Almost all block codes used today belong to a subset called **linear block codes**.

A linear block code:

- In which the exclusive OR of two valid codewords creates another valid codeword.

Example:

Dataword	00	01	10	11
Codeword	000	011	101	110

Minimum Distance for Linear Block Codes

- The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Parity Check Code

A simple parity-check code is a single-bit error-detecting code in which

$$n = k + 1 \text{ with } d_{\min} = 2.$$

Even parity : Ensures that a codeword has an even number of 1's

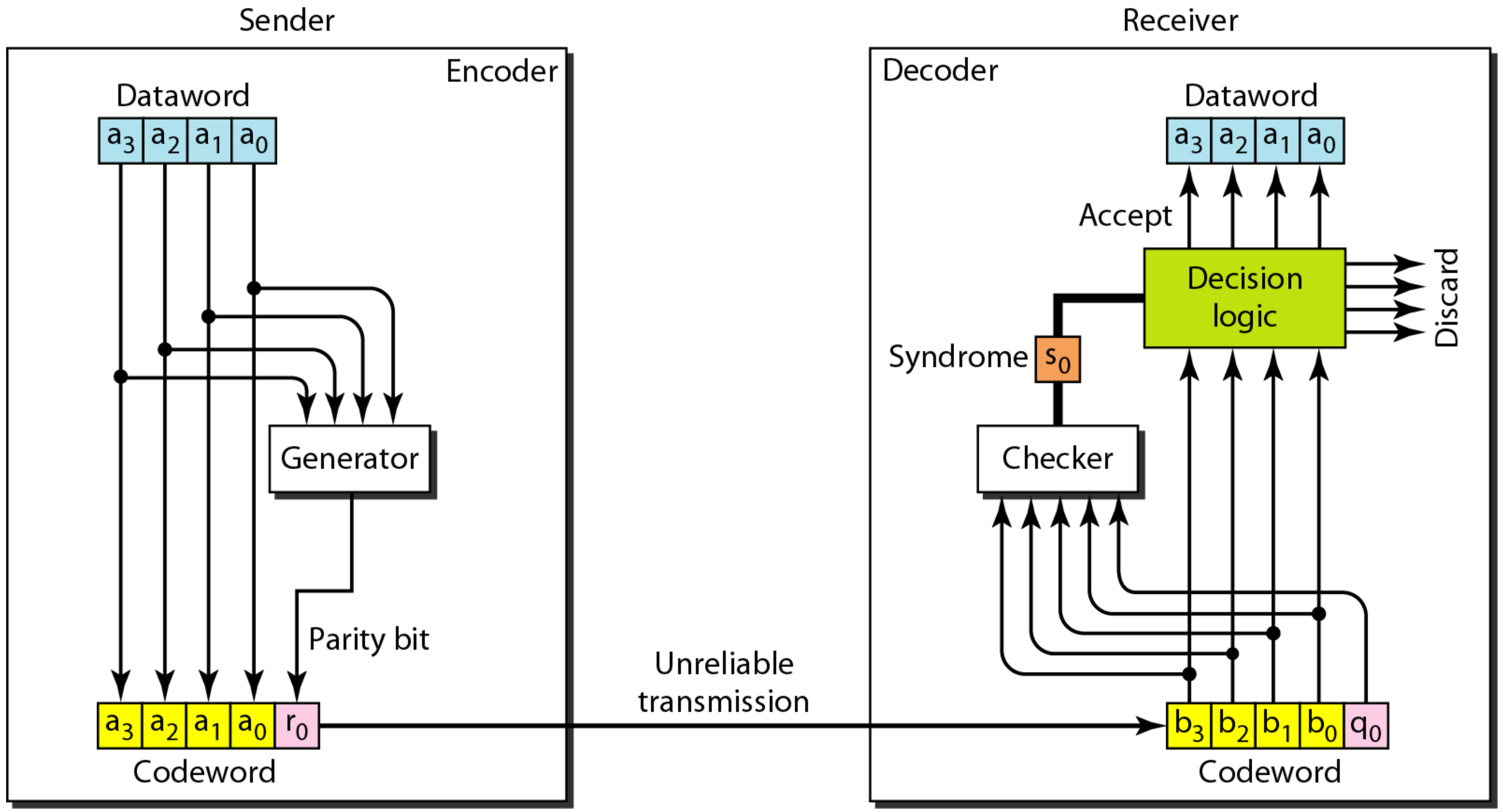
Odd parity : Ensures that there are an odd number of 1's in the codeword

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

Parity Check Code

Calculation is done in **modular arithmetic**



Encoder and decoder for simple parity-check code

The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

Syndrome is passed to the decision logic analyzer.
If the syndrome is 0, there is no detectable error in the received codeword;
the data portion of the received codeword is accepted as the dataword;

Cyclic Codes

Special linear block codes with one extra property.

- If a codeword is cyclically shifted (rotated), the **result is another codeword.**

For example,

- If 1011000 is a codeword
- We cyclically left-shift, then 0110001 is also a codeword.
- In this case, if we call the bits in the first word *a*0 to *a*6, and the bits in the second word *b*0 to *b*6, we can shift the bits by using the following:

$$b_1 = a_0;\ b_2 = a_1;\ b_3 = a_2;\ b_4 = a_3;\ b_5 = a_4;\ b_6 = a_5;\ b_0 = a_6$$

Cyclic Redundancy Check (CRC)

- CRC Code is a subset of cyclic codes, used in networks such as LANs and WANs.
- Have both the *linear and cyclic* properties.

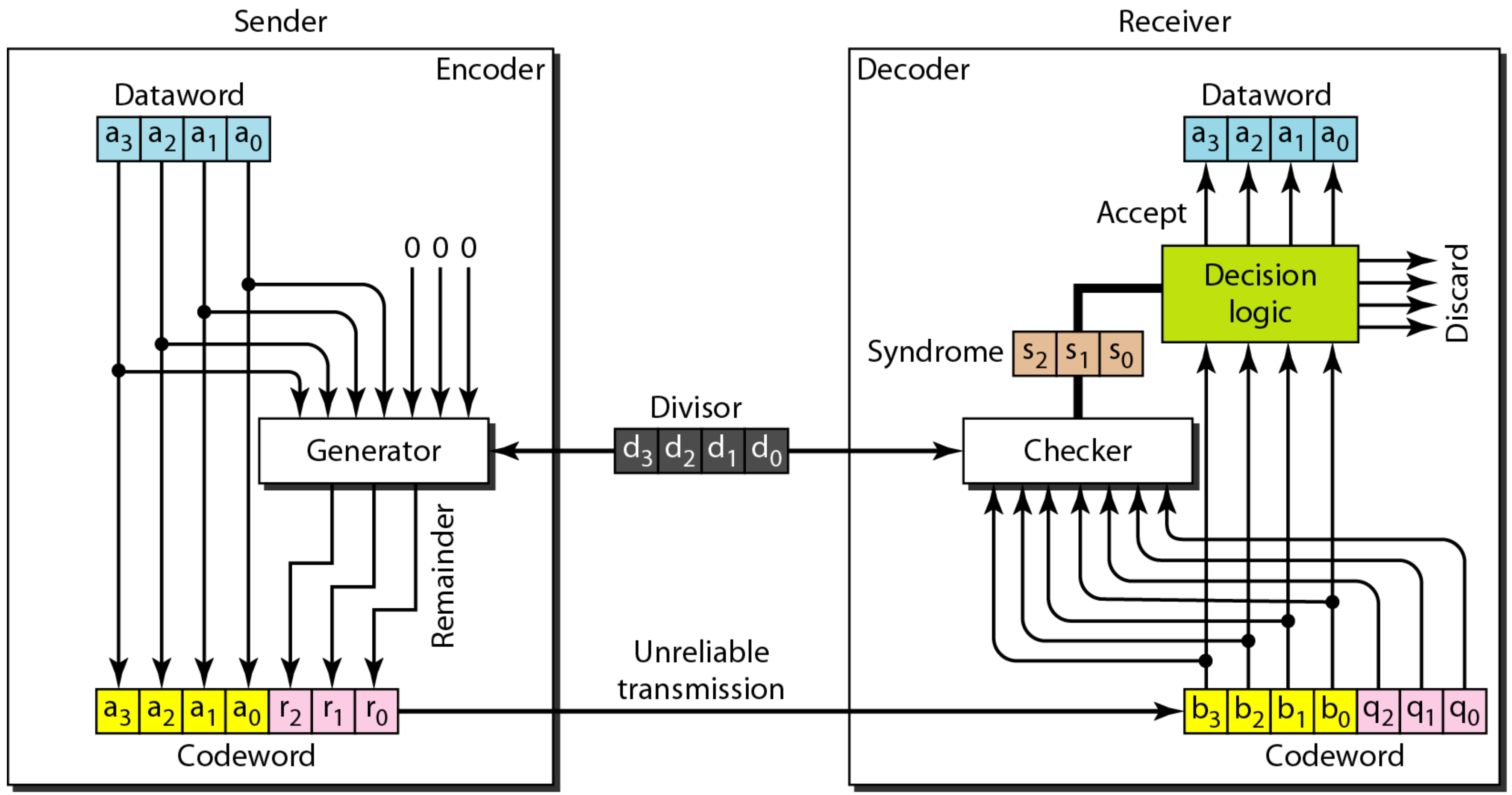
A CRC code with C(7, 4)

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

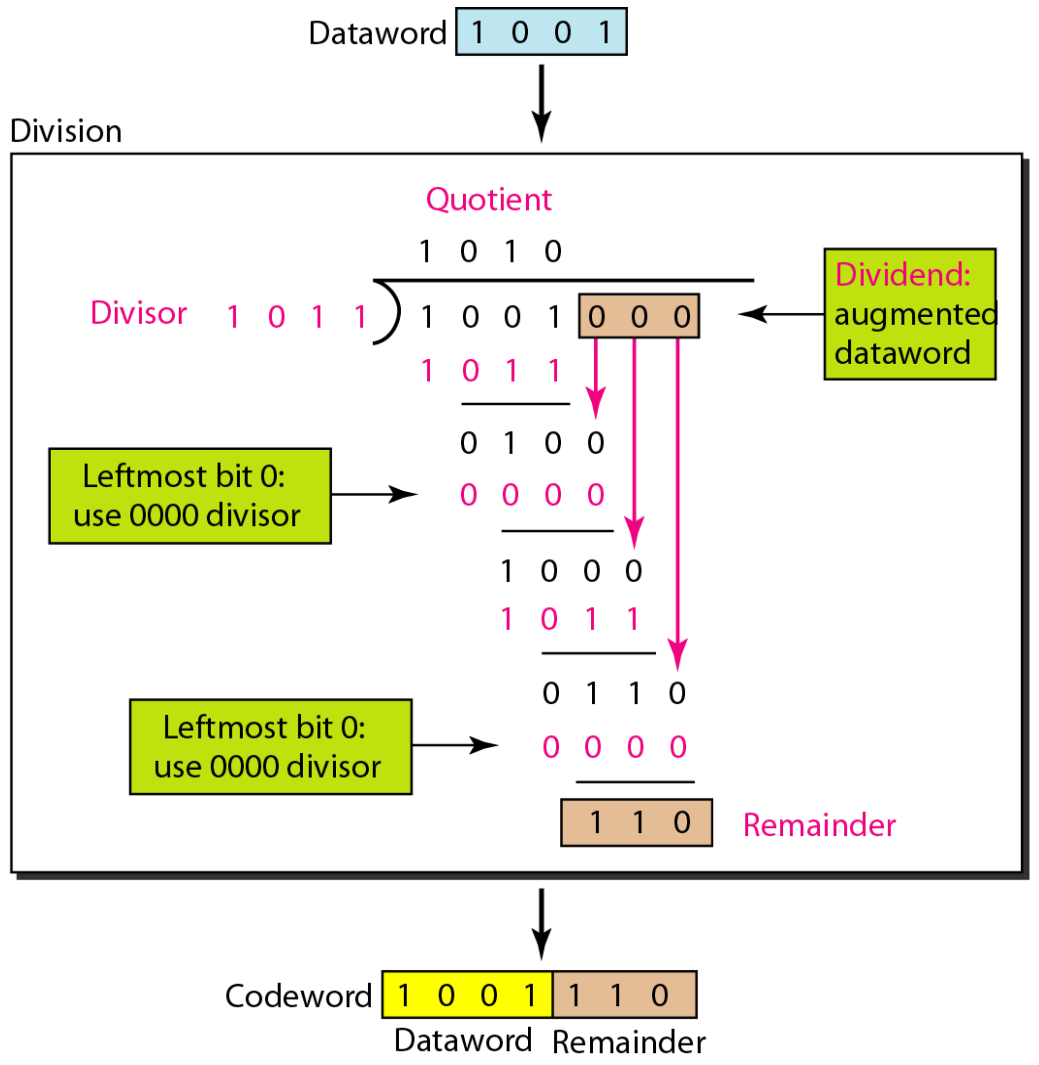
CRC Encoder & Decoder

•

Generator uses a divisor of size $n - k + 1$

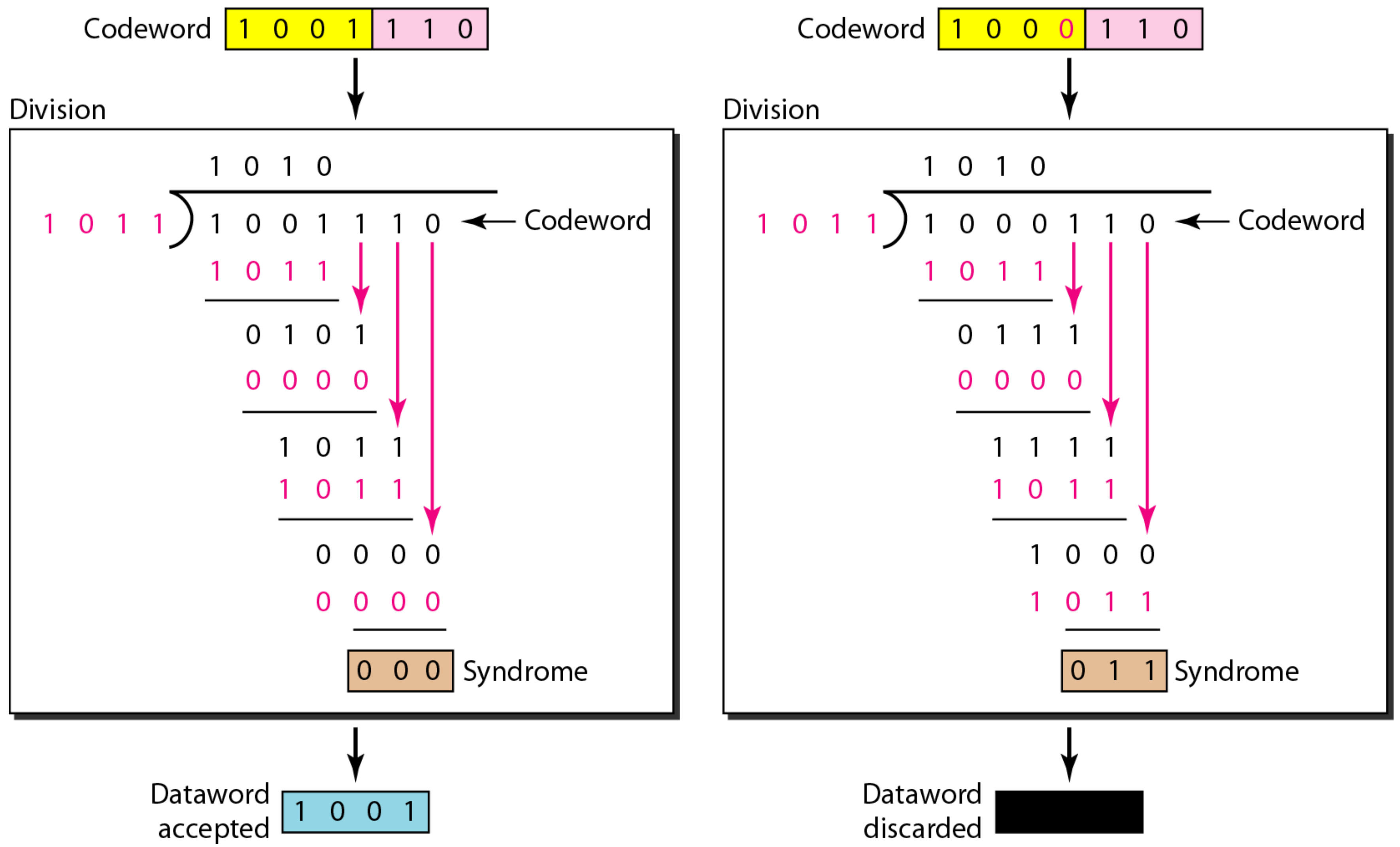


CRC Encoder



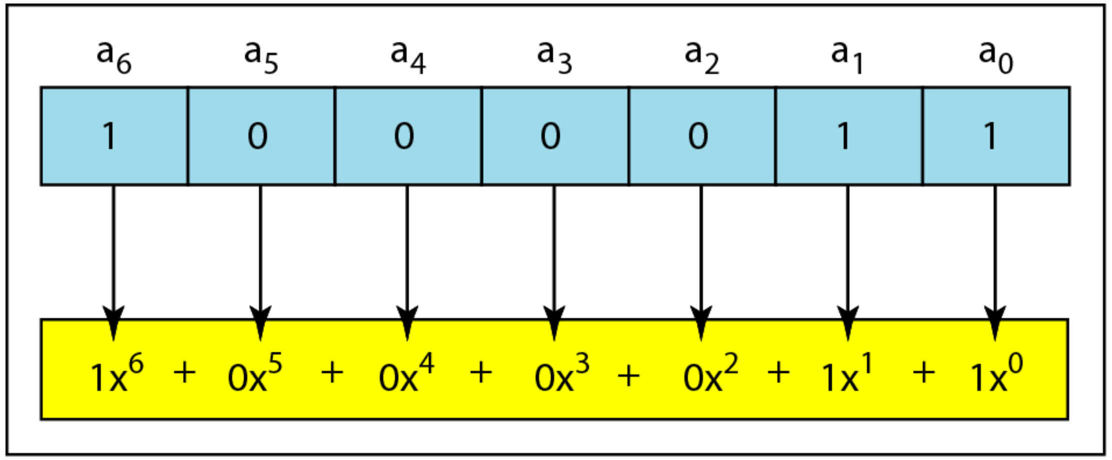
Dataword 1 0 0 1
Division
Quotient
1 0 1 0
Divisor 1 0 1 1
1 0 0 1 0 0 0
Dividend: augmented dataword
1 0 1 1
0 1 0 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 0 0 0
1 0 1 1
0 1 1 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 1 0
Remainder
Codeword 1 0 0 1 1 1 0
Dataword
Remainder

CRC Decoder

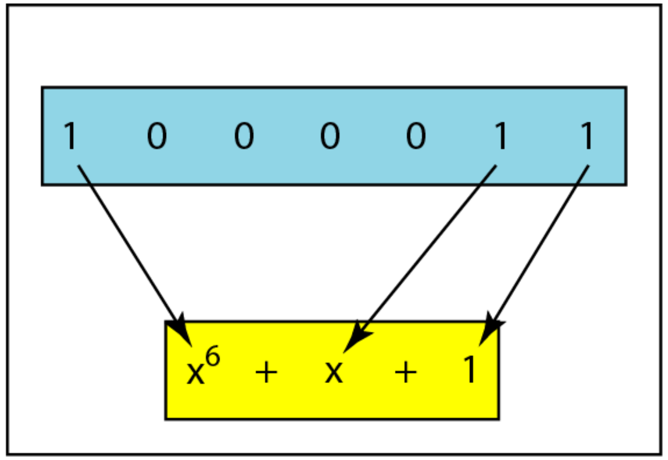

Codeword 1 0 0 1 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 1 1 1 0 ← Codeword
1 0 1 1
0 1 0 1
0 0 0 0
1 0 1 1
1 0 1 1
0 0 0 0
0 0 0 0
0 0 0 Syndrome
Dataword accepted 1 0 0 1
Codeword 1 0 0 0 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 0 1 1 0 ← Codeword
1 0 1 1
0 1 1 1
0 0 0 0
1 1 1 1
1 0 1 1
1 0 0 0
1 0 1 1
0 1 1 Syndrome
Dataword discarded

CRC Polynomial Representation

- 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.


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

a. Binary pattern and polynomial



b. Short form

CRC Polynomial Representation

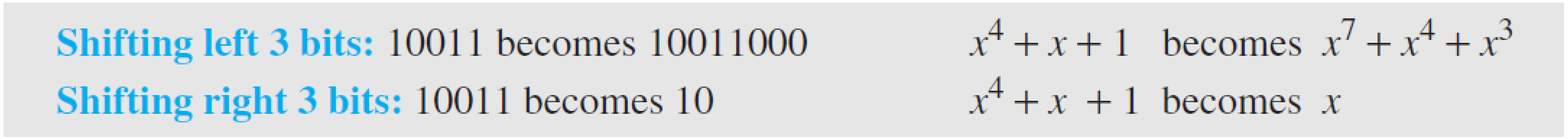
- **Degree of Polynomial**
  - Highest power in the polynomial.
  - For example, the degree of the polynomial $x^6 + x + 1$ is 6.
  - The bit pattern in this case has 7 bits.
- **Addition & Subtraction of Polynomials**
- **Multiplication & Division**
- **Multiplying Two Polynomials**

$$\begin{aligned}(x^5 + x^3 + x^2 + x)(x^2 + x + 1) &= x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x \\ &= x^7 + x^6 + x^3 + x\end{aligned}$$

- **Dividing Two Polynomials**

CRC Polynomial Representation

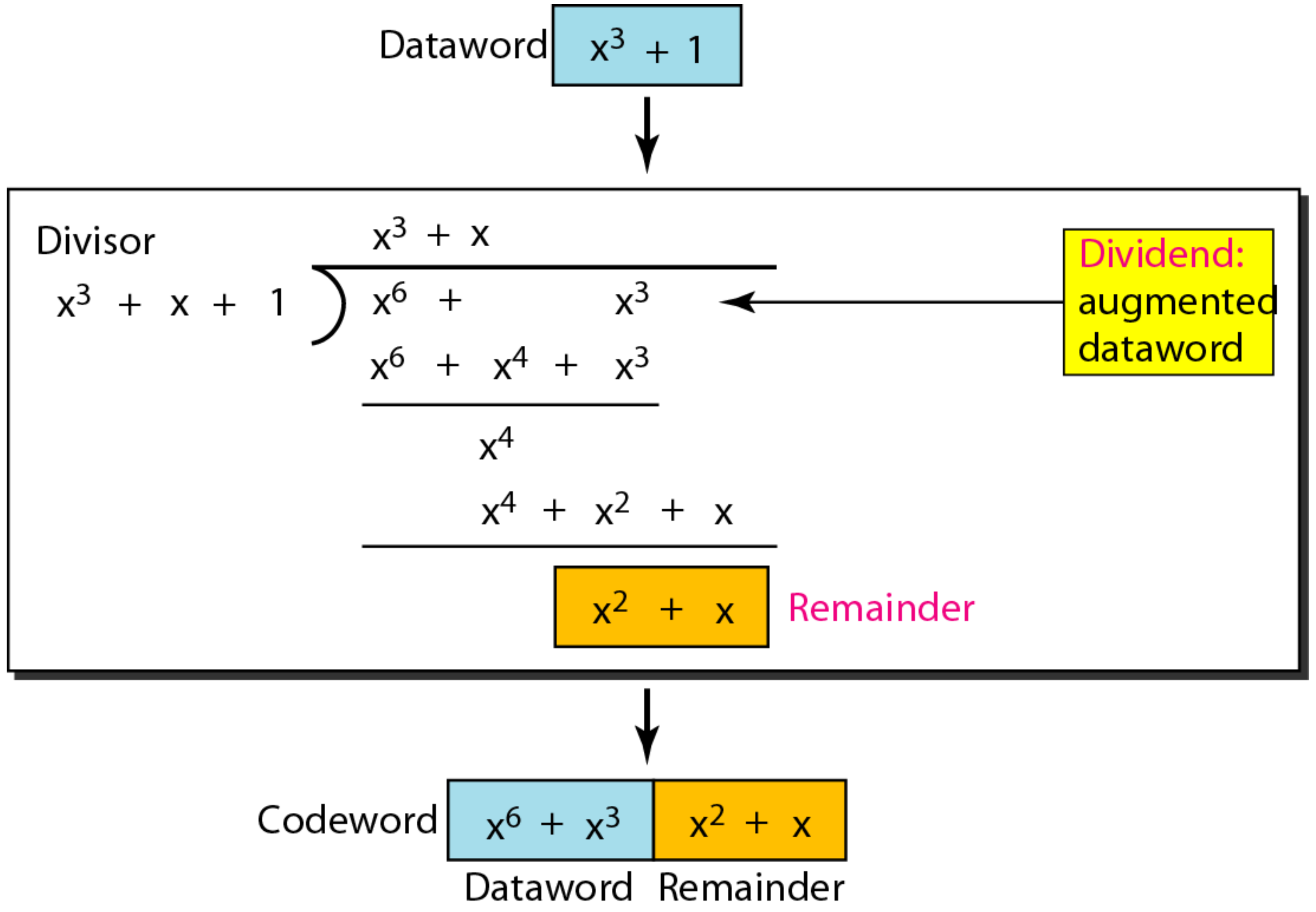
- Shifting
  - A binary pattern is often shifted a number of bits to the right or left.
  - Shifting to the left:
    - Adding extra 0s as rightmost bits;
    - Done by multiplying each term of the polynomial by x^m,
  - Shifting to the right:
    - Deleting some rightmost bits.
    - Done by dividing each term of the polynomial by x^m.



Shifting left 3 bits: 10011 becomes 10011000	$x^4 + x + 1$ becomes $x^7 + x^4 + x^3$
Shifting right 3 bits: 10011 becomes 10	$x^4 + x + 1$ becomes x

Cyclic Code Encoder using Polynomials

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



Cyclic Code Analysis

Polynomial with Binary Coefficients : f(x)

Dataword : d(x); Codeword : c(x); Generator : g(x); Syndrome : s(x); Error : e(x)

In a cyclic code,

- **If $s(x) \neq 0$, one or more bits is corrupted.**
- **If $s(x) = 0$, either**
  - No bit is corrupted. or
  - Some bits are corrupted, but the decoder failed to detect them.

In a cyclic code, those *e*(*x*) errors that are divisible by *g*(*x*) are not caught.

$$Received\ Code\ Word : c(x) + e(x)$$

$$\frac{Received\ Code\ Word}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

Cyclic Code Analysis

Single Bit Error

- A single-bit error is $e(x) = x^i$
  - where i is the position of the bit.
- If a single-bit error is caught, then x^i is not divisible by $g(x)$.
- If $g(x)$ has at least two terms and the coefficient of x^0 is not zero (the rightmost bit is 1), then $e(x)$ cannot be divided by $g(x)$.

Cyclic Code Analysis

Single Bit Error

Example:

Which of the following g(x) values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$ b. x^3 c. 1

Solution

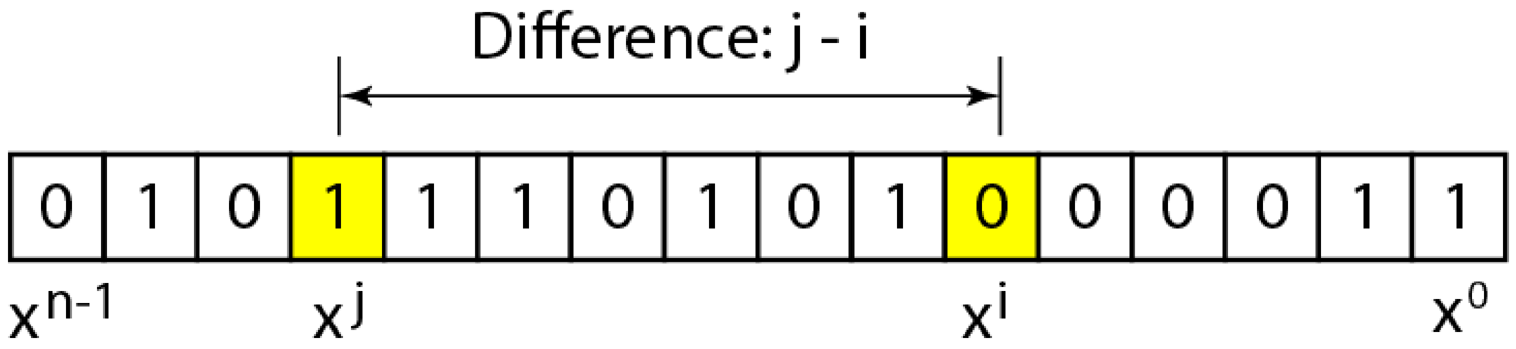
a. ***No x^i can be divisible by $x + 1$.*** *Any single-bit error can be caught.*

b. ***If i is equal to or greater than 3, x^i is divisible by g(x).*** *All single-bit errors in positions 1 to 3 are caught.*

c. ***All values of i make x^i divisible by g(x).*** *No single-bit error can be caught. This g(x) is useless.*

Cyclic Code Analysis

Two isolated single-bit errors



If a generator cannot divide $x^t + 1$ (t between 0 and n – 1), then all isolated double errors can be detected.

Cyclic Code Analysis

Two isolated single-bit errors

Find the status of the following generators related to two isolated, single-bit errors.
a. $x + 1$ *b.* $x^4 + 1$ *c.* $x^7 + x^6 + 1$ *d.* $x^{15} + x^{14} + 1$

Solution

a. This is a very poor choice for a generator. Any two errors next to each other cannot be detected.

b. This generator cannot detect two errors that are four positions apart.

c. This is a good choice for this purpose.

d. This polynomial cannot divide $x^t + 1$ *if t is less than 32,768. A codeword with two isolated errors up to 32,768 bits apart can be detected by this generator.*

Cyclic Code Analysis

Odd Number of Errors

A generator that contains a factor of
***x* + 1 can detect all odd-numbered errors.**

***For example*,**
$x^4 + x^2 + x + 1$ can catch all odd-numbered errors
since it can be written as a product of the two polynomials *x* + 1 and $x^3 + x^2 + 1$.

Burst Errors

- **All burst errors with $L \leq r + 1$ will be detected.**
- **All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$.**
- **All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^{r}$.**

Cyclic Code Analysis

Burst Errors

- A burst error is of the form $e(x) = (x^j + \ldots + x^i)$.
- **Difference between a burst error and two isolated single-bit errors.**
  - The first can have two terms or more;
  - The second can only have two terms.
- We can **factor out x^i** and write the **error as $(x^{j-i} + . . . + 1)$.**
- If our ***generator can detect a single error*** (minimum condition for a generator), then *it cannot divide x^i*.
- What we should worry *about are those generators that divide x^{j-i} + . . . + 1.*
- In other words, the remainder of $(x^{j-i} + . . . + 1)/(x^r + . . . + 1)$ must not be zero.
  - Note that the denominator is the generator polynomial.

Cyclic Code Analysis

Burst Errors

- Case 1: If $j - i < r$,
  - the remainder can never be zero.
  - We can write $j - i = L - 1$,
    - where L is the length of the error.
  - So $L - 1 < r$ or $L < r + 1$.
    - This means all burst errors with length smaller than or equal to the number of check bits r will be detected.

- **All burst errors with $L \leq r + 1$ will be detected.**

Cyclic Code Analysis

Burst Errors

- Case 2: *if j – i = r, or L = r + 1*,
  - the syndrome is 0 and the error is undetected.
  - It can be proved that in these cases, the probability of undetected burst error of length r + 1 is $(1/2)^{r-1}$.
  - For example, if our generator is **$x^{14} + x^{3} + 1$,** in which r = 14,
  - a burst error of length **L = 15 can slip by undetected with the probability of $(1/2)^{14-1}$ or almost 1 in 10,000.**

- **All burst errors with *L = r + 1* will be detected with probability 1 - $(1/2)^{r-1}$.**

Cyclic Code Analysis

Burst Errors

- Case 3: if $j - i > r$, or $L > r + 1$,
  - the syndrome is 0 and **the error is undetected**.
  - It can be proved that in these cases, the probability of undetected burst error of length greater than r + 1 is $(1/2)^r$.
  - For example, if our generator is $x^{14} + x^3 + 1$, in which r = 14,
  - a **burst error of length greater than 15 can slip by undetected** with the probability of $(1/2)^{14}$ or almost 1 in 16,000 cases.

- **All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^r$.**

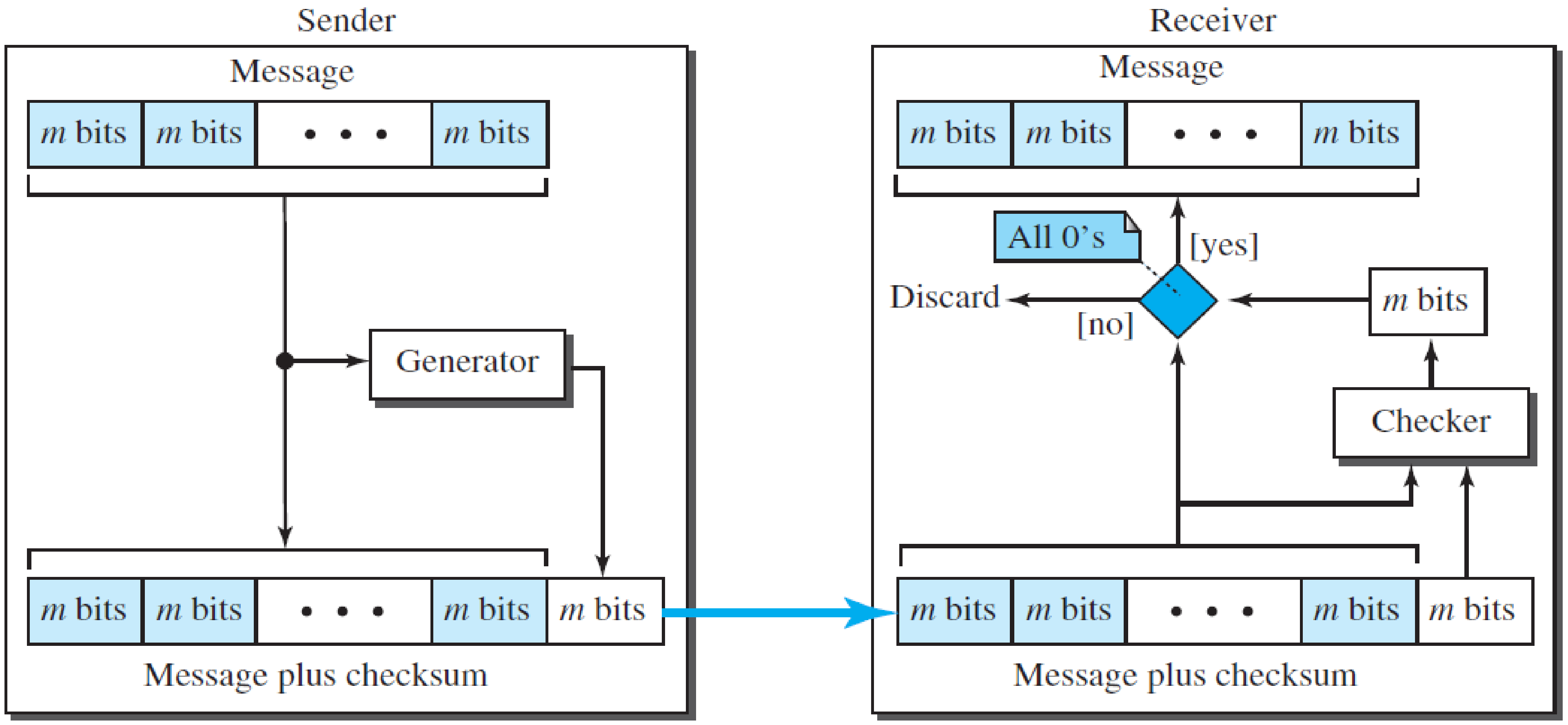
Summary : Good Polynomial Generator

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

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

Checksum


Sender
Message
m bits
m bits
• • •
m bits
Generator
m bits
m bits
• • •
m bits
m bits
Message plus checksum
Receiver
Message
m bits
m bits
• • •
m bits
All 0's
[yes]
Discard
[no]
m bits
Checker
m bits
m bits
• • •
m bits
m bits
Message plus checksum

Concept of Checksum

- 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.
    - 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.

Concept of Checksum … (contd.)

- The previous example has one major drawback.
  - Each number can be written as a **4-bit word** (each is less than 15) except for the sum.
  - One solution is to use **one's complement** arithmetic.
  - In this arithmetic,
    - We can represent unsigned numbers between 0 and $2^m - 1$ using only *m* bits.
    - If the number has more than *m* bits, the extra leftmost bits need to be added to the *m* rightmost bits (wrapping).

Example: The decimal number 36 in binary is $(100100)_2$. To change it to a 4-bit number we add the extra leftmost bit to the right four bits as:

$$(10)_2 + (0100)_2 = (0110)_2 = (6)_{10}$$

Concept of Checksum … (contd.)

- 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.

Concept of Checksum … (contd.)

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

Solution

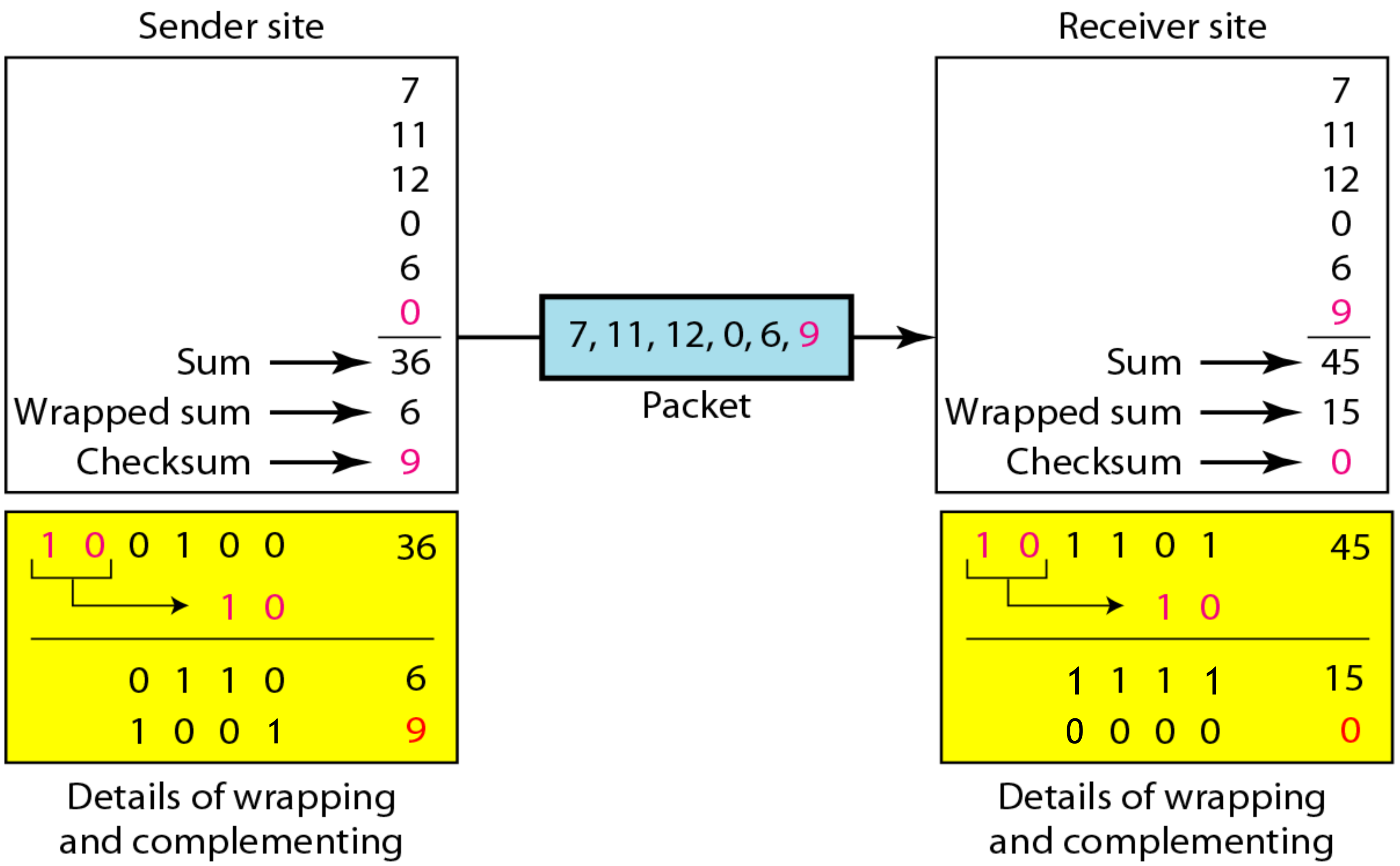
In one’s complement arithmetic, the negative or complement of a number is found by inverting all bits.

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).

Checksum : Example



Internet Checksum

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.

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.

Reference

Forouzan, A. Behrouz. *Data Communications & Networking*. 5th Edition. Tata McGraw-Hill Education.

Chapter 10 Error Detection and Correction

Topic: 10.1, 10.2, 10.3, 10.4