Module 2

10.1 INTRODUCTION

10.2 BLOCK CODING

10.3 CYCLIC CODES

10.4 CHECKSUM

10.5 FORWARD ERROR CORRECTION

10-1 INTRODUCTION

- TCP/IP protocol does not define any protocol in DLL or PL.
- These two layers are territories of networks that when connected make up the Internet.
- These networks (wired/wireless), provide services to upper three layers of TCP/IP suite.
- Networks must be able to transfer data from one device to another with acceptable accuracy.
- For most of the applications, system must guarantee that data received are identical to the one transmitted .

INTRODUCTION

- In transmission, there are possibilities that any time message gets corrupted.
- Many factors alter one or more bits of a message.
- Some applications require a mechanism for detecting and correcting errors.
- Some applications can tolerate some errors.

Types of Errors

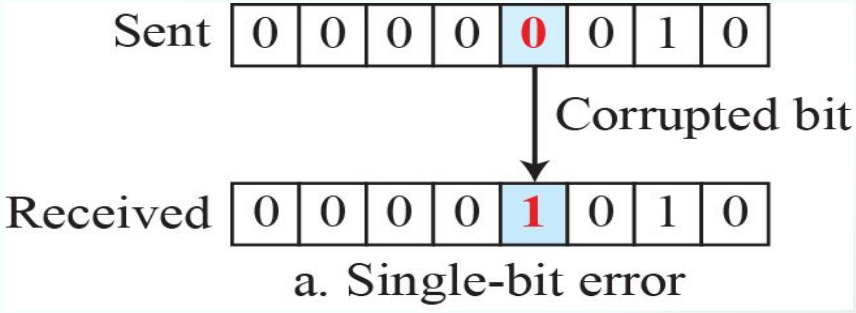
- Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference.

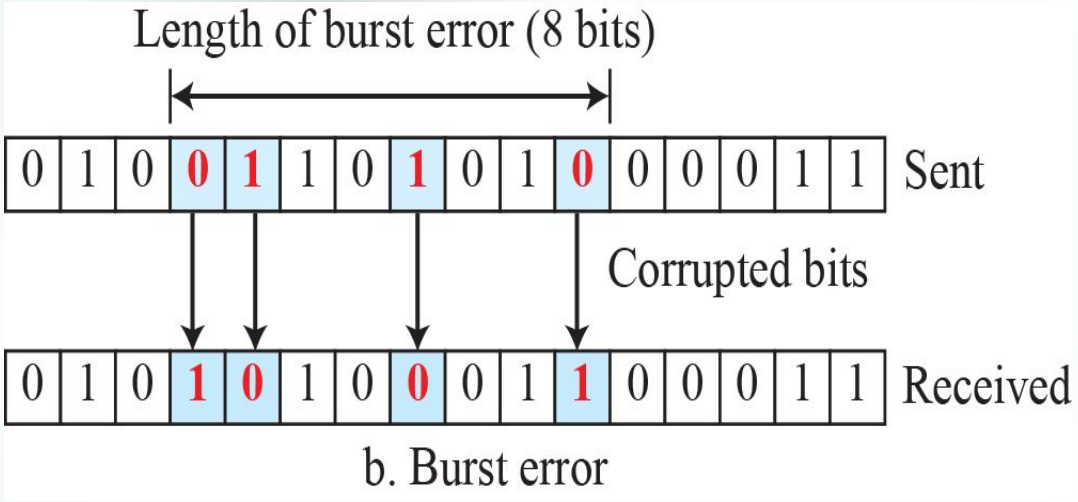
- This interference can change the shape of the signal.

- The term single-bit error means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.

- The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure below shows the effect of a single-bit and a burst error on a data unit.

Single-bit and burst error





Types of Errors

- Burst error is more likely to occur than single bit error.
- Reason is duration of noise signal is longer than the duration of a bit.
- When noise affects data, it affects set of bits.
- Number of bits affected depends on the duration of the noise and data rate.

Eg: If we send 1 kbps of data, a noise 1/100 s can affect 10 bits.
If 1 Mbps data rate, noise can affect 10,000 bits.

Redundancy

- The central concept in detecting or correcting errors is redundancy.
- To be able to detect or correct errors, need to send some extra bits with our data.
- These redundant bits are added by the sender and removed by the receiver.
- Presence of redundant bits allows the receiver to detect or correct corrupted bits.

Detection versus Correction

- The correction of errors is more difficult than the detection.

- In error detection, check if any error has occurred. The answer is a simple yes or no.Not interested in the number of corrupted bits.

- A single-bit error is the same for us as a burst error.

- In error correction,

  Need to know the exact number of bits that are corrupted and, more importantly, their location in the message.

Coding

- Redundancy is achieved through various coding schemes.
- The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits.
- The receiver checks the relationships between the two sets of bits to detect errors.
- The ratio of redundant bits to data bits and the robustness of the process are important factors in any coding scheme.

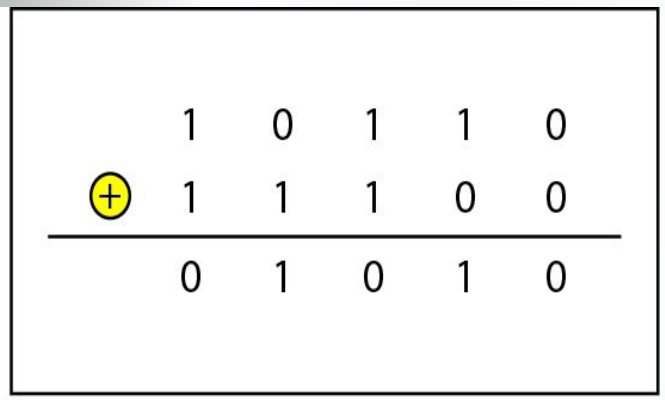
Coding

- Two coding schemes are

  i) Block coding ii) Convolution coding
- Concentrate on block codes.
- In modulo-N arithmetic, the integers in the range 0 to N −1 inclusive is used,

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

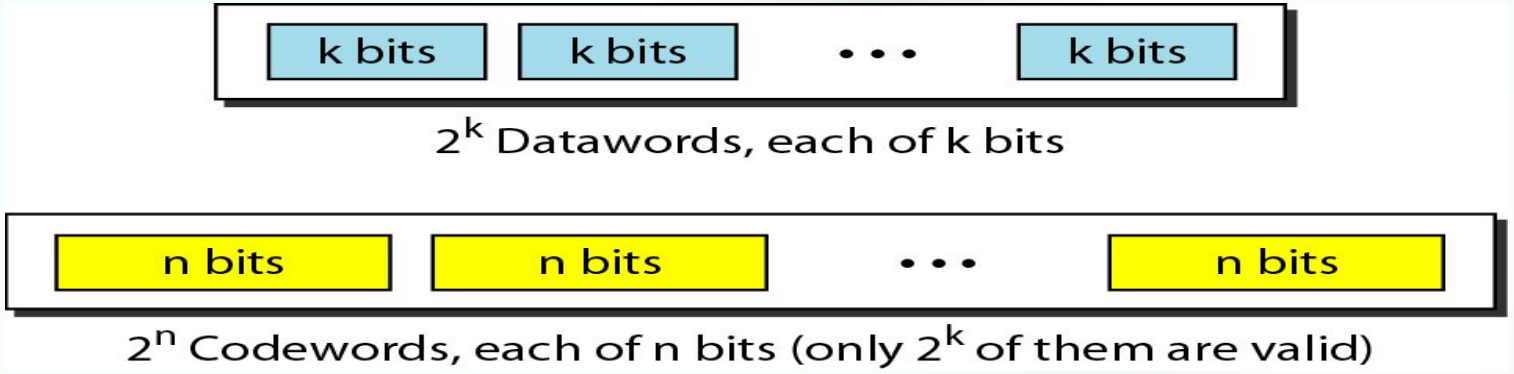
b. Two bits are different, the result is 1.



c. Result of XORing two patterns

10-2 BLOCK CODING

- In block coding, message is divided into blocks, each of k bits, called data words.
- Add r redundant bits to each block to make the length $n = k + r$.
- The resulting n-bit blocks are called code words.



BLOCK CODING

- In block coding, we have set of data words, each of size k, code words, each of size n.
- Since n>k, the possible number of code words are larger than the data words.
- Block coding process is one-to-one, same data word is encoded as code word.
- $2^n - 2^K$ code words are not used.
- If receiver receives invalid codeword, it indicates data is corrupted during transmission.

Error Detection

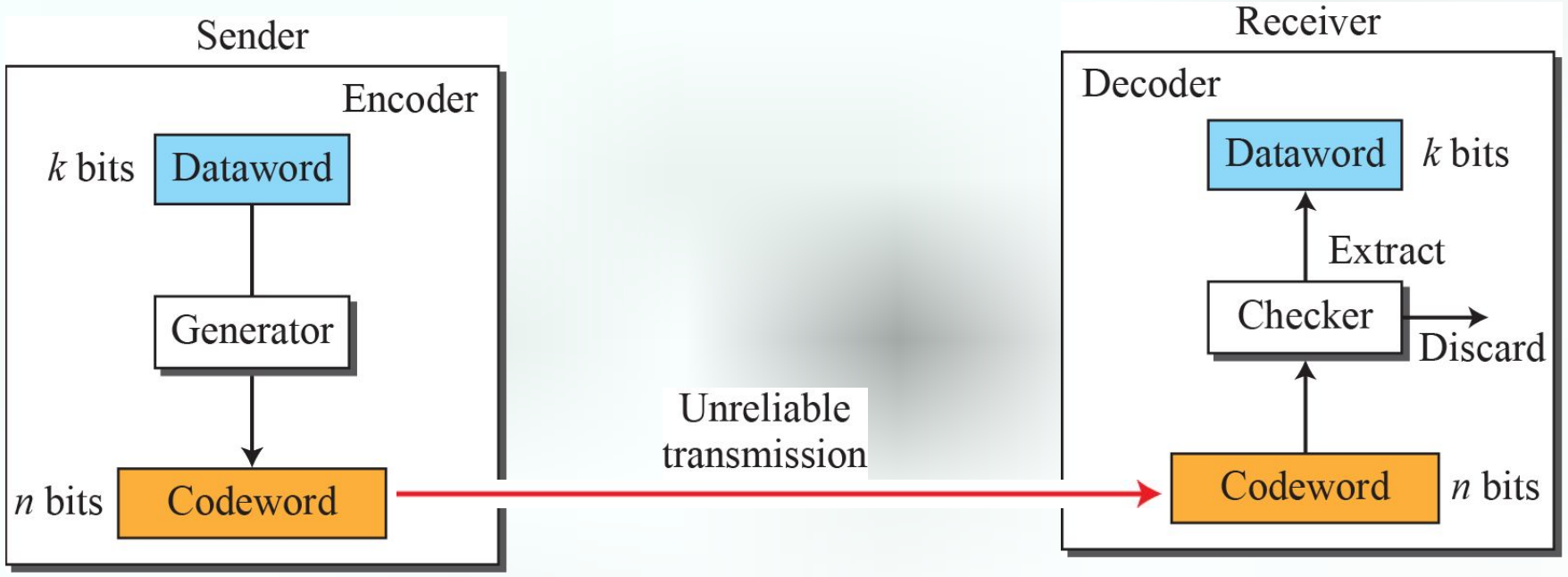
- How can errors be detected by using block coding?

- If the following two conditions are met, the receiver can detect a change in the original codeword.

a. The receiver has (or can find) a list of valid code words.

b. The original codeword has changed to an invalid one.

Process of error detection in block coding



Example 10.1

Let us assume that k = 2 and n = 3. Table below shows the list of data words and code words.

A code for error detection

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

Assume the sender encodes the dataword 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 dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Hamming distance

- The Hamming distance between two words(of same size) is the number of differences between corresponding bits. *(total number of 1's).*
- *Why HD in Error detection????*
- *HD between the sent code word & received codeword is the number of bits that got corrupted during transmission.*

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

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

Hamming Distance

- The Hamming distance d(000, 011) is 2.
- The Hamming distance d(10101, 11110) is 3 because

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

- *If the HD between sent & received code word is not zero, then code word has been corrupted during transmission.*
- *The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.*

Hamming distance

- Find the minimum Hamming distance of the coding scheme in

Datawords	Codewords
00	000
01	011
10	101
11	110

We first find all Hamming distances

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

The d_{min} in this case is 2

Example 10.2

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).

Example 10.3

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 code word (101) is sent and one error occurs, the received code word does not match any valid code word.

If two errors occur, however, the received code word may match a valid code word and the errors are not detected.

Example 10.6

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 $\boldsymbol{d}_{\text{min}} = 2$.

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 code words creates another valid codeword.

Table : 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

Example

- Let us see if the two codes defined in Tables above, belong to the class of linear block codes.

1. The scheme in the above Table is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword.

   So the minimum Hamming distance is $d_{min} = 2$.

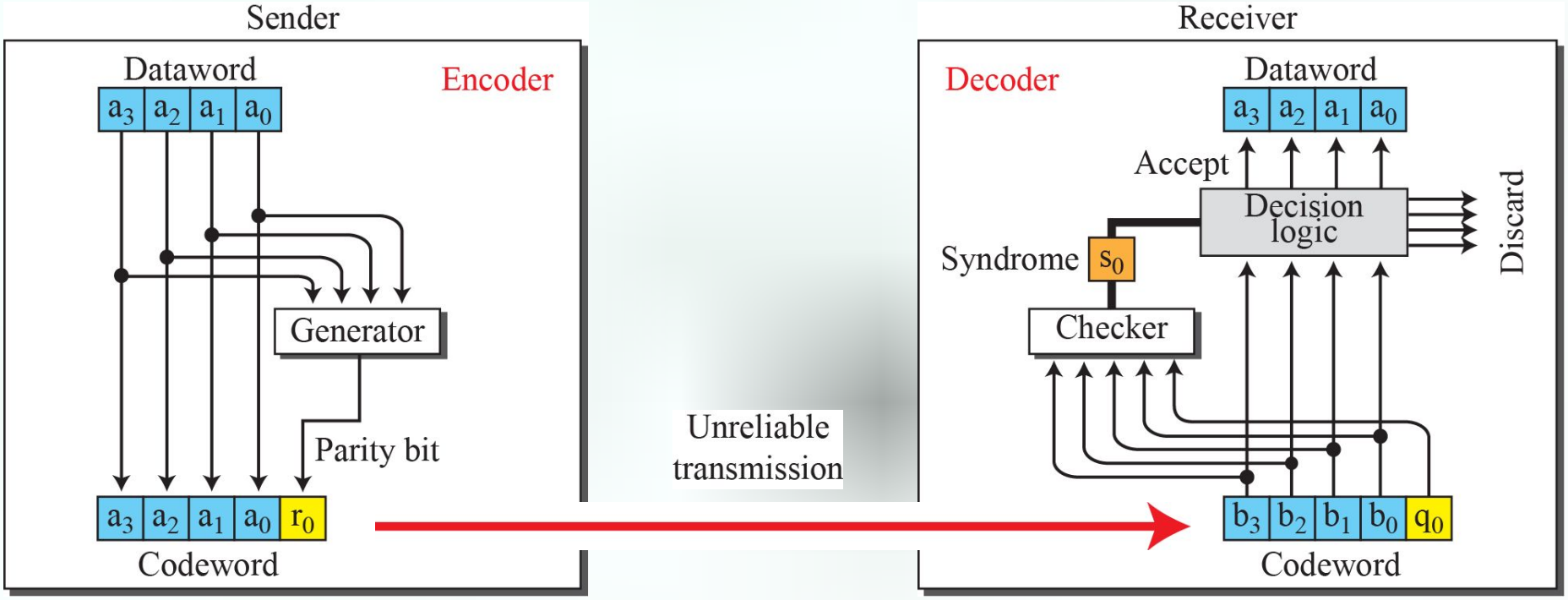
2. The scheme in Table second is also a linear block code. We can create all four codewords by XORing two other codewords.

- *So the minimum hamming distance is* $d_{min} = 3$.

Simple Parity check code

- A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.
- The extra bit, called parity bit , is selected to make number of 1's in the code word even.
- The code is single-bit error detecting code.
- It can not correct any error.
- In the fig. encoder takes a copy of 4 bit data word(a_0, a_1, a_2, a_3) and generates a parity bit r_0.
- Parity bit creates the 5-bit code word.

Encoder and decoder for simple parity-check code



Simple Parity check code

- The parity bit that is added makes the number of 1's in the codeword even. i.e

$$r_0 = a_0 + a_1 + a_2 + a_3 \quad \text{(modulo-2)}$$

- If number of 1s is even, the result is 0.
- If number of 1s is odd, the result is 1.
- The sender sends the code word which is corrupted during transmission.
- The receiver receives 5 bit code word.
- The checker performs modulo-2 of all the bits received. The result is called syndrome.

$$s_0 = b_0 + b_1 + b_2 + b_3 + q_0 \quad \text{(modulo-2)}$$

Continued..

- If number of 1s in the codeword is even, the syndrome is 0.
- If number of 1s in the codeword is odd, the syndrome is 1.
- The syndrome is passed to decision logic analyzer. If Syndrome is 0, no error in the code word received, data portion is extracted.
- If syndrome is 1, error in the received codeword and discarded.

Example 10.7

Let us look at some transmission scenarios. Assume the sender sends the data word 1011. The codeword created from this data word is 10111, which is sent to the receiver. Five cases can be examined:

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 10**0**11. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0. The received codeword is 1011**0**. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes r_0 and a second error changes a_3. The received codeword is **0**011**0**. 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. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is **010**11. 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.

10-3 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.

Eg:1011000 is a codeword and if it is cyclically left-shifted, then 0110001 is also a codeword.

We can shift the bits using

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

Cyclic Redundancy Check

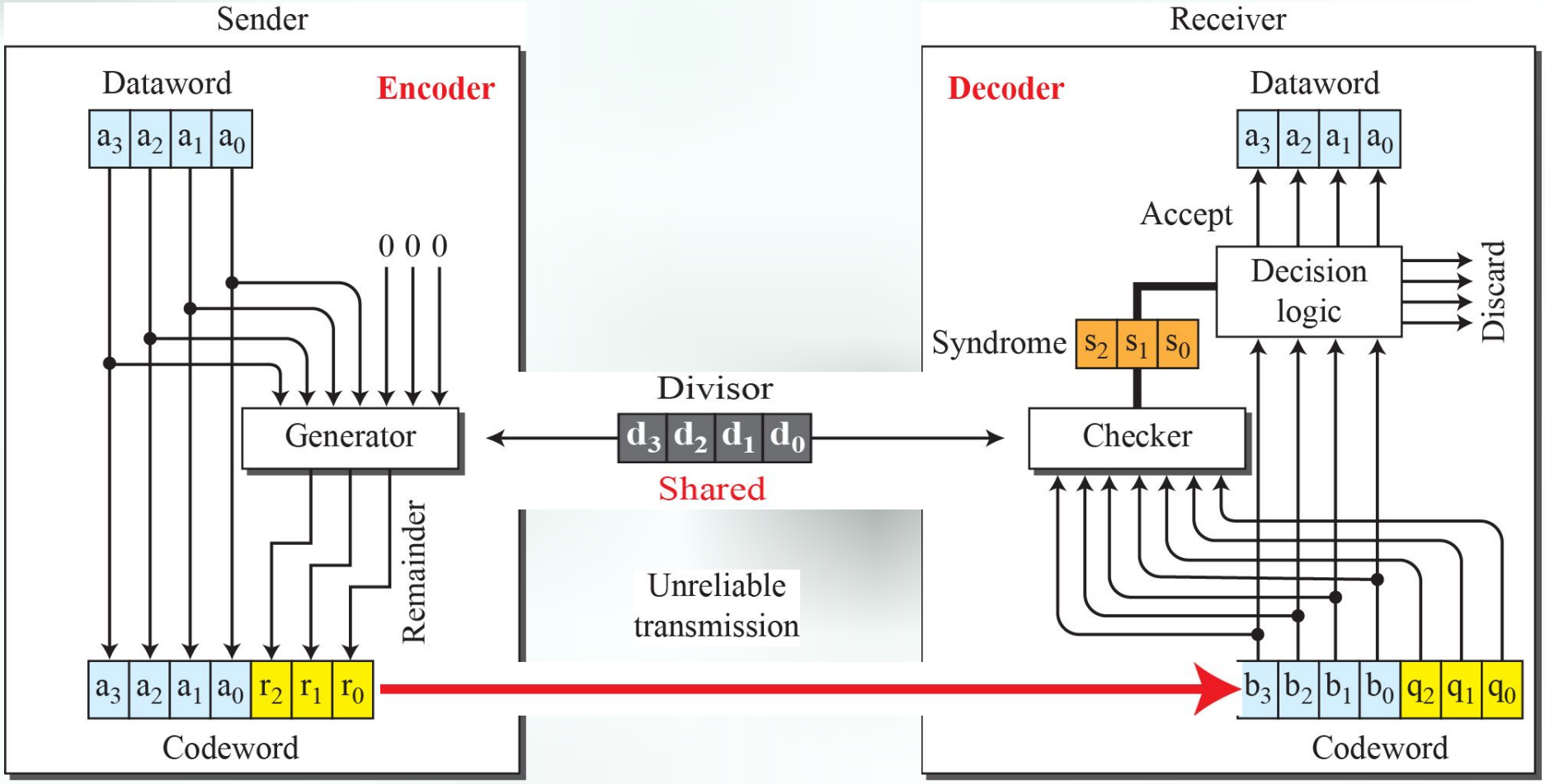
- Cyclic codes are created to correct errors.

- Discuss a subset of cyclic codes called the cyclic redundancy check (CRC), which is used in networks such as LANs and WANs.

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 and decoder



CRC

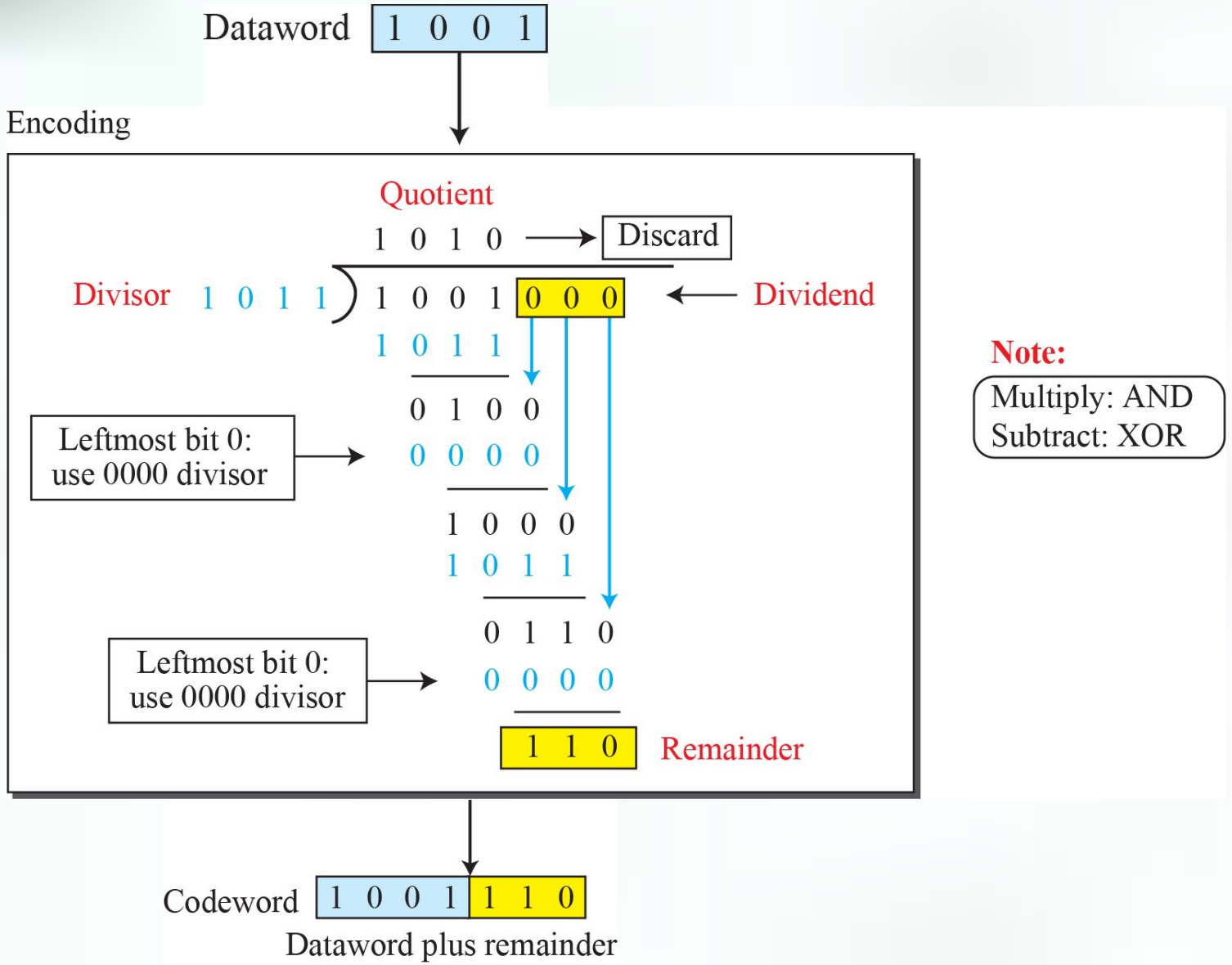
- In the encoder, the data word has k bits(4 bits), code word has n bits(7 here).
- The data word is augmented by adding n-k 0s to the right hand side of the word.
- The n-bit result is fed to the generator.
- The generator uses a divisor of size n-k+1which is predefined and agreed upon.
- The generator divides the augmented dataword

CRC

by divisor (modulo-2 division).

- The quotient of the division is discarded.
- The remainder $r_2 r_1 r_0$ is appended to the data word to create the codeword.
- The decoder receives the corrupted codeword.
- A copy of all n bits are fed to the checker.
- The remainder produced by checker is a syndrome of n-k bits, which is fed to the decision logic analyzer.
- If syndrome bits are 0, the 4 left most bits of the code word are accepted as a data word.

Division in CRC encoder



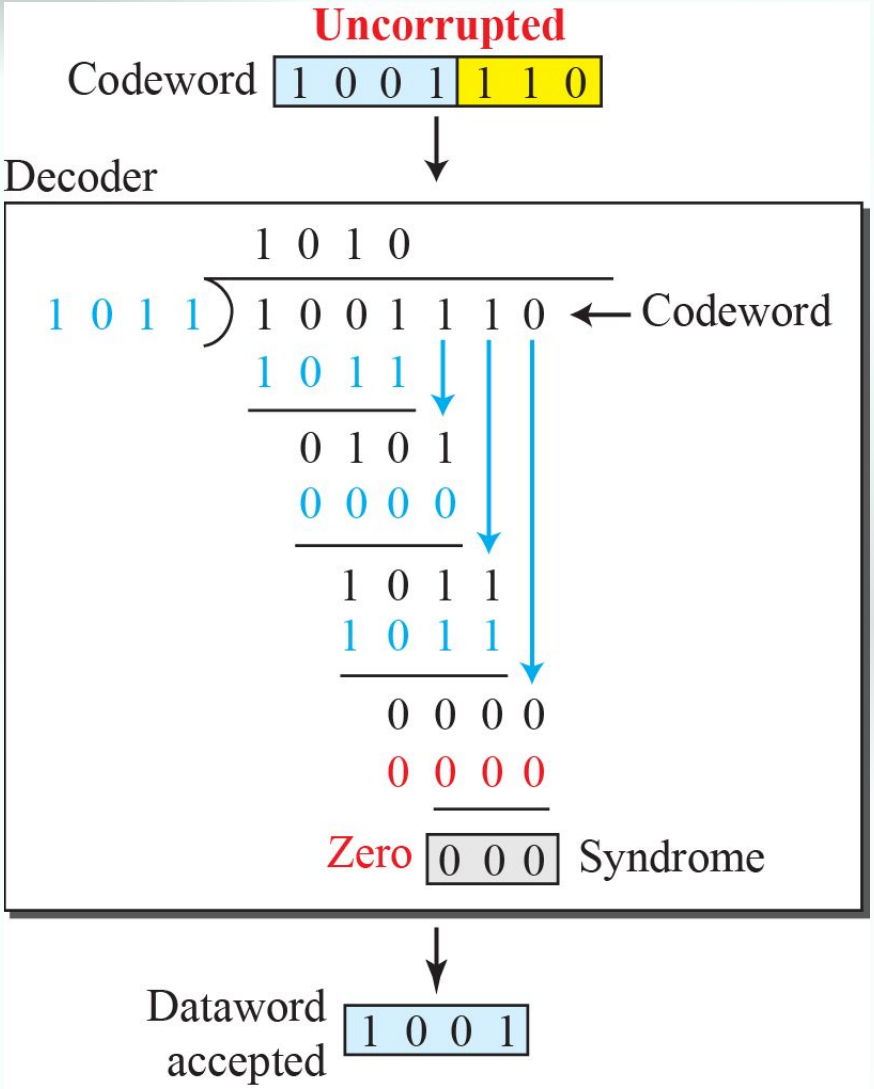
CRC-Encoder

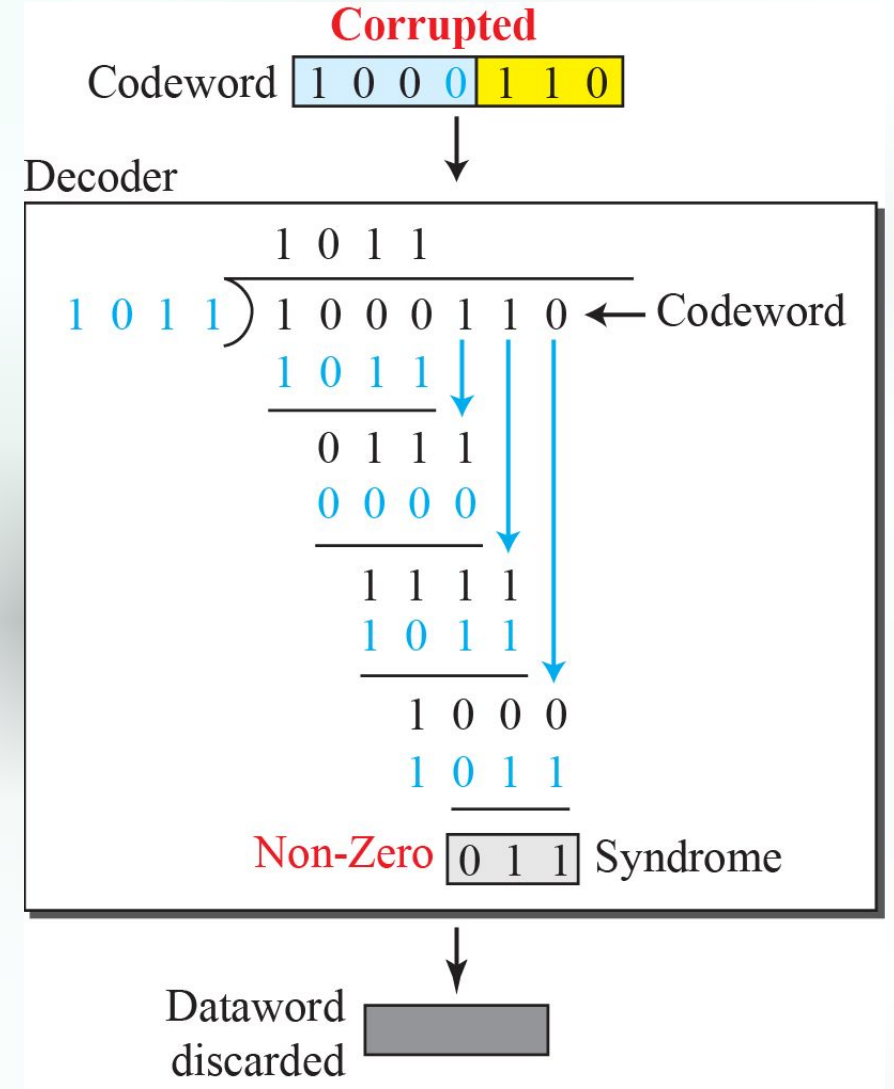
- Encoder takes data word and augments it with n-k number of 0s.
- It then divides the augmented data word by a divisor.
- Modulo-2 binary division is used. For Addition and subtraction, XOR operations is used.
- In each step, copy of the divisor is XOR ed with 4 bits of dividend. The result of XOR operation is 3 bits.

CRC-Decoder

- Code word can change during transmission.
- The decoder same thing as encoder and remainder of the division id the syndrome.
- If syndrome is all 0s, no error, data word is separated from the code word and accepted.
- Otherwise , everything is discarded.
- Question is how divisor 1011 is chosen??

Division in the CRC decoder for two cases

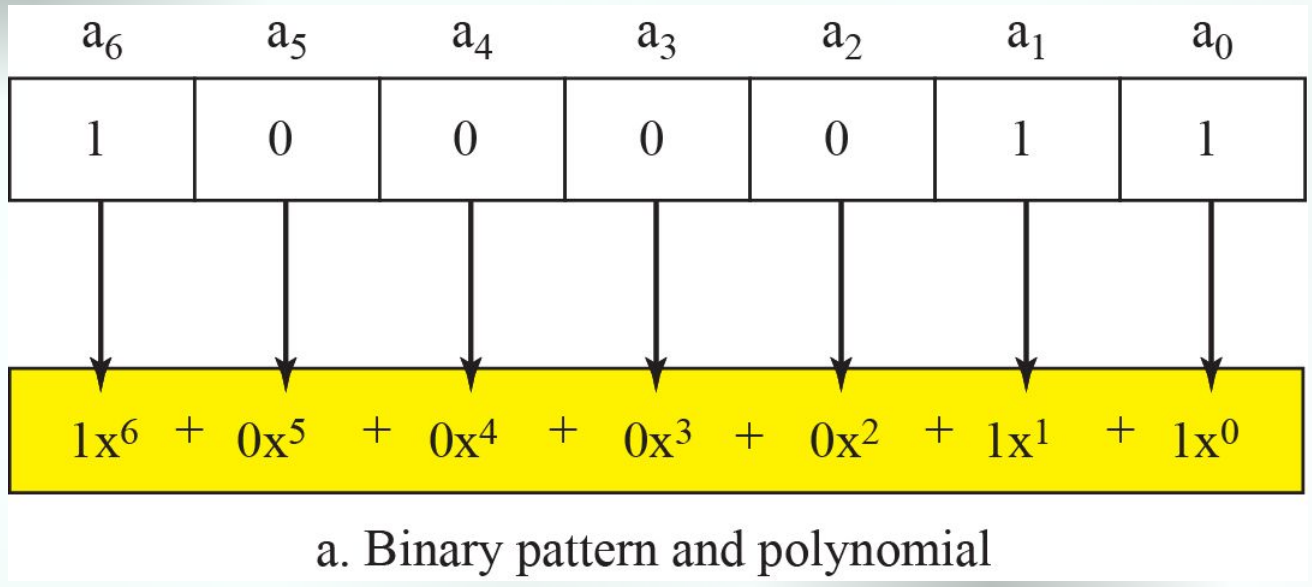




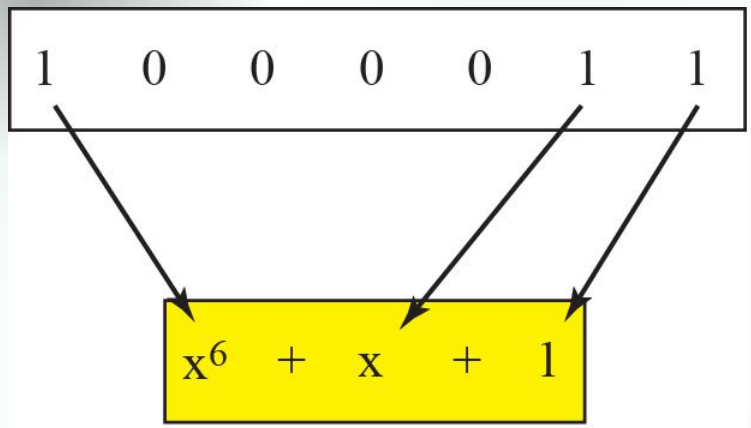
Polynomials

- A better way to understand cyclic codes, and how they can be analyzed is to represent them as polynomials.
- A pattern of 0s and 1s can be represented as polynomial with coefficient of 0 and 1.
- The power of each term shows the position of the bit.
- The coefficient shows the value of the bit.

A polynomial to represent a binary word



a. Binary pattern and polynomial



b. Short form

Degree of Polynomials

- Is the highest power of the polynomial.

Eg: Degree of $x^6 + x+1$ is 6.

- Degree of polynomial is one less than that the number of bits in the pattern.
- Addition and subtraction of polynomials is done by adding and subtracting the coefficient terms with the same power.
- Coefficients are only 0 and 1, and adding is in modulo-2

Adding and subtracting of polynomials

- Addition and subtraction is done by combining terms and deleting pair of identical terms.
- Eg: $x^5 + x^4 + x^2$

$$x^6 + x^4 + x^2$$

$$x^6 + x^5$$

Multiplication and Division of polynomials

- Multiplication is adding powers.

Eg: $x^4 * x^2 = x^6$

- Division, subtract the power of second term from the first term.

Eg: $x^5 / x^2 = x^3$

- Multiplication of two polynomials

$(x^5 + x^3 + x^2)(x^2 + x + 1) = x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2$

$\Rightarrow x^7 + x^6 + x^2$

Shifting of polynomials

- A binary pattern is shifted left or right by number of bits.
- Shifting to the left means adding extra 0s as right most bits.
- Shifting to the right means deleting some right most bits.
- Shifting to the left is accomplished by multiplying each term of the polynomial by x^m
  where m is the number of shifted bits

Continued..

- Shifting to the right is accomplished by dividing each term of the polynomial by x^m where m is the number of shifted bits.

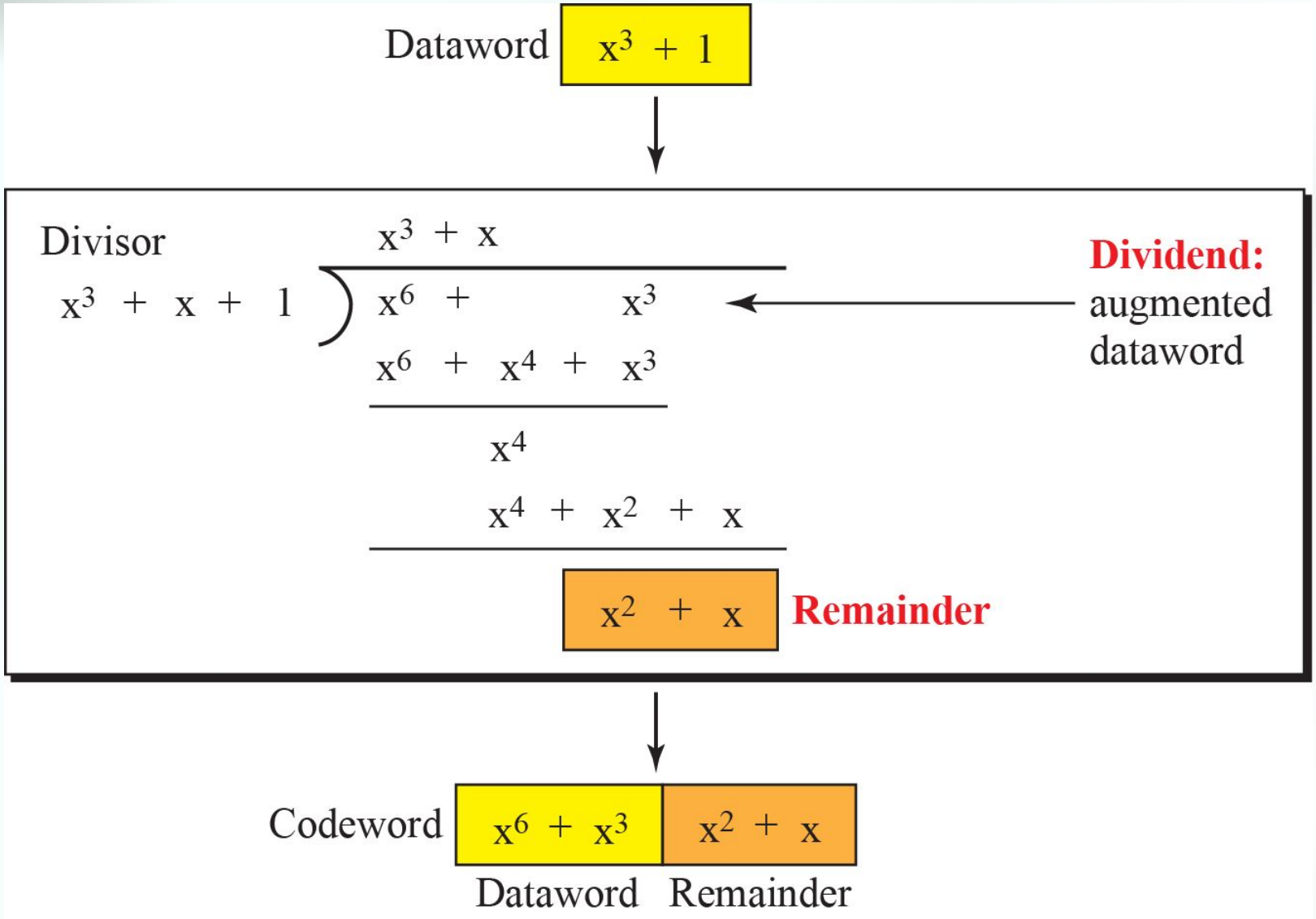
- Shift left by 3 bits :10011 becomes 10011000

x^4+x+1 becomes $x^7 + x^4+x^3$

- Shift right by 3 bits: 10011 becomes 10

x^4+x+1 becomes x

CRC division using polynomials



Cyclic code analysis using polynomials

- In a polynomial representation, divisor is normally referred to as generator polynomial t(x).
- f(x) is the polynomial with binary coefficients data word d(x), code word c(x), Generator g(x) syndrome s(x) and error e(x).
  - Is s(x) is not zero, one or more bits are corrupted. If s(x) is zero, either no error or decoder failed to detect any error.

Cyclic Code Analysis

- Analyze a cyclic code to find its capabilities by using polynomials.

- Define the following, where f(x) is a polynomial with binary coefficients.

Dataword: $d(x)$ **Codeword:** $c(x)$

Generator: $g(x)$ **Syndrome:** $s(x)$ **Error:** $e(x)$

Example 10.8

Which of the following g(x) values guarantees that a single-bit error is caught? $x + 1$, x^3 and 1

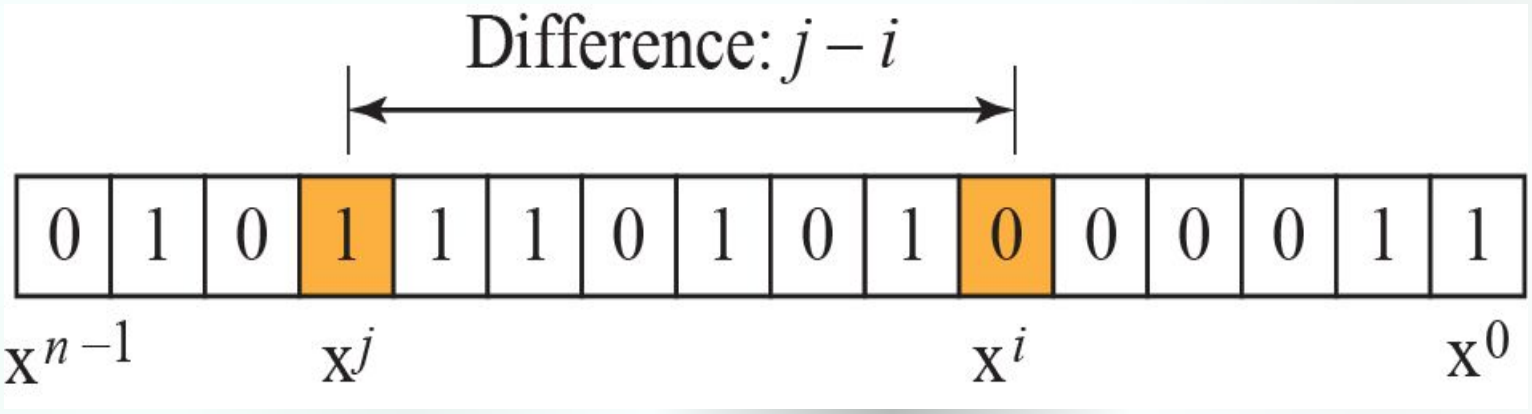
Solution

a. No x^i can be divisible by $x + 1$. In other words, $x^i/(x + 1)$ always has a remainder. So the syndrome is nonzero. Any single-bit error can be caught.

b. If i is equal to or greater than 3, x^i is divisible by $g(x)$. The remainder of x^i/x^3 is zero, and the receiver is fooled into believing that there is no error, although there might be one. Note that in this case, the corrupted bit must be in position 4 or above. 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. In addition, this $g(x)$ is useless because it means the codeword is just the dataword augmented with $n - k$ zeros.

Representation of isolated single-bit errors



Example 10.9

Find the suitability of the following generators in relation to burst errors of different lengths: $x^6 + 1$, $x^{18} + x^7 + x + 1$, and $x^{32} + x^{23} + x^7 + 10$.

Solution

a. This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.

b. This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of 1 million burst errors of length 20 or more will slip by.

c. This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.

Example 10.10

Find the status of the following generators related to two isolated, single-bit errors: $x + 1$, $x^4 + 1$, $x^7 + x^6 + 1$, and $x^{15} + x^{14} + 1$

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. The two errors can be anywhere, but if their distance is 4, they remain undetected.

c. This is a good choice for this purpose.

d. This polynomial cannot divide any error of type $x^t + 1$ if t is less than 32,768. This means that a codeword with two isolated errors that are next to each other or up to 32,768 bits apart can be detected by this generator.

Summary

- 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 $x + 1$.

Table :Standard polynomials

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **1000100000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **10000010011000001000111011011 0111**	LANs

Advantages of Cyclic Codes

- Cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors.

- They can easily be implemented in hardware and software.

- They are especially fast when implemented in hardware which has made cyclic codes a good candidate for many networks.

Other Cyclic Codes

- The cyclic codes we have discussed are very simple.
- The check bits and syndromes can be calculated by simple algebra.
- There are, however, more powerful polynomials that are based on abstract algebra involving Galois fields.
- One of the most interesting of these codes is the Reed-Solomon code used today for both detection and correction.

10-4 CHECKSUM

- Checksum is an error-detecting technique that can be applied to a message of any length.
- In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.
- At source,

 Message is divided into m-bit units.

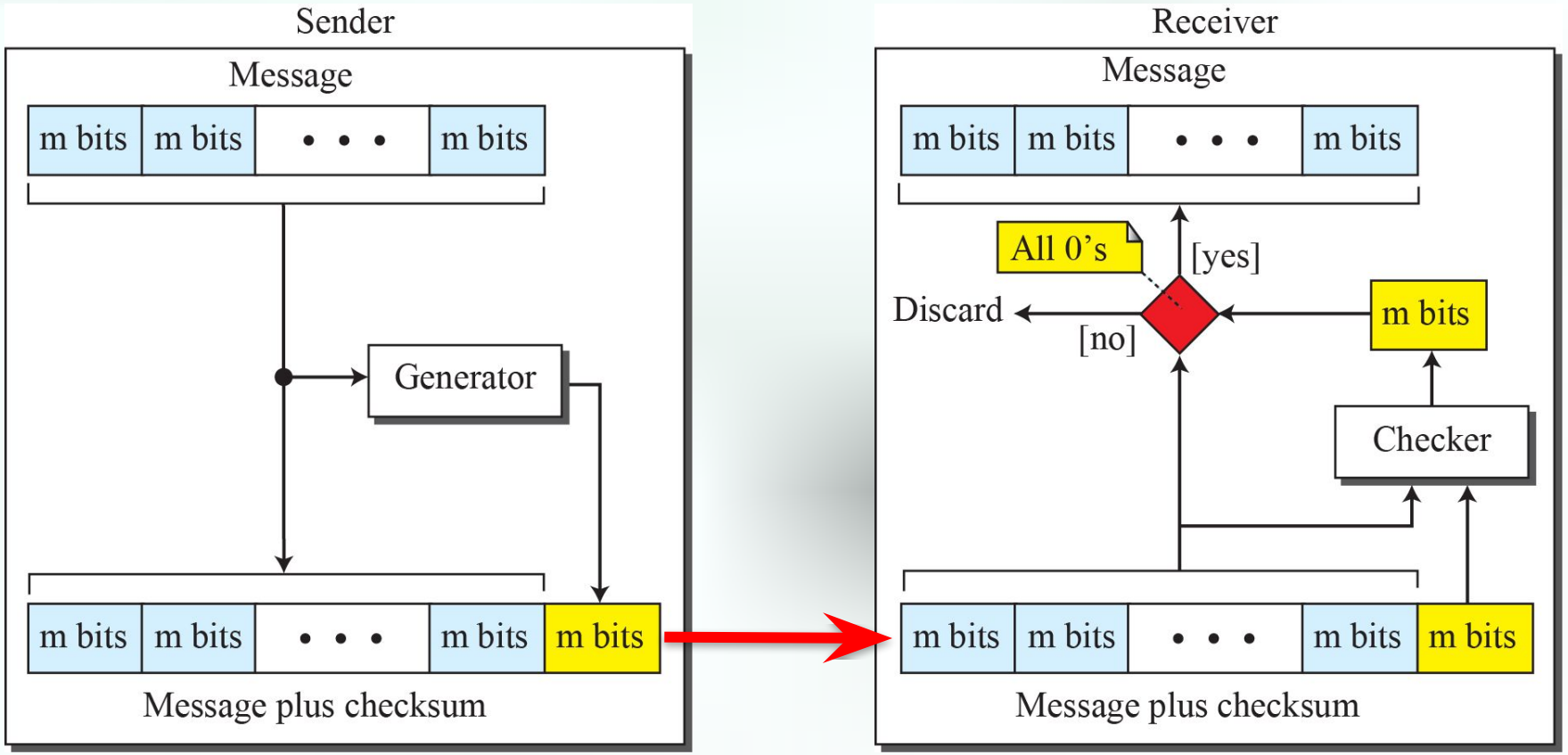
Generator then creates an extra m-bit unit called checksum, which is sent with message.

CHECKSUM

- At Destination,

- Checker creates a new checksum from combination of the message and the sent checksum.
- If new checksum is all 0's, the message is accepted else discarded.

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

- The idea of the traditional checksum is simple.

Show this using a simple example.

Suppose the message 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.

Eg: 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 message not accepted.

Ones compliment Addition

- Each number can be written as a 4 bit word except sum.
- This drawback we can over come using ones compliment arithmetic.
- 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.

Example 10.12

In the previous 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 shown below.

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

Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, 6).

The receiver can add the first five numbers in one's complement arithmetic.

If the result is 6, the numbers are accepted; otherwise, they are rejected.

Example 10.13

Let us use the idea of the checksum in Example 10.12. The sender adds all five numbers in one's complement to get the sum = 6.

The sender then complements the result to get the checksum = **9**, which is 15 − 6.

- Note that 6 = (0110)2 and **9** = (1001)2;they are complements of each other.
- The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**).
- If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one's complement to get 15.

Example 10.13

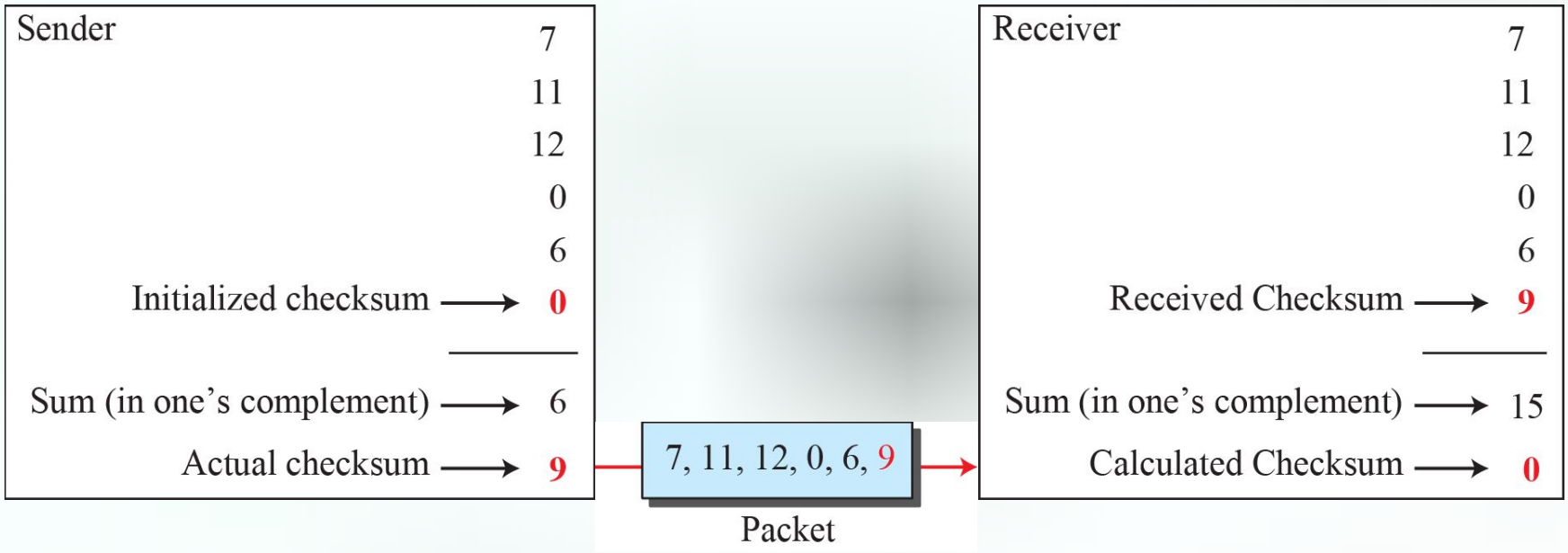


Table : Procedure to calculate the traditional checksum

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.