# MODULE 3: ERROR-DETECTION AND CORRECTION

## 3.1 INTRODUCTION

### 3.1.1 Types of Errors

- When bits flow from 1 point to another, they are subject to unpredictable-changes '.' of interference.
- The interference can change the shape of the signal.
- Two types of errors: 1) Single-bit error 2) Burst-error.

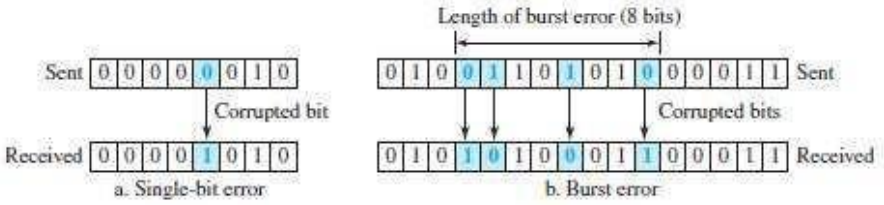



Figure 10.1 *Single-bit and burst error*

**1) Single-Bit Error**

 Only 1 bit of a given data is changed → from 1 to 0 or

→ from 0 to 1 (Figure 10.1a).

**2) Burst Error**

- Two or more bits in the data have changed → from 1 to 0 or

  → from 0 to 1 (Figure 10.1b).
- A burst-error occurs more than a single-bit error.

  This is because:

  Normally, the duration of noise is longer than the duration of 1-bit.  When noise affects data, the noise also affects the bits.
- The no. of corrupted-bits depends on → data-rate and → duration of noise.

### 3.1.2 Redundancy

- The central concept in detecting/correcting errors is *redundancy*.
- Some extra-bits along with the data have to be sent to detect/correct errors. These extra bits are called redundant-bits.
- The redundant-bits are

  → added by the sender and

  → removed by the receiver.
- The presence of redundant-bits allows the receiver to detect/correct errors.

### 3.1.3 Error Detection vs. Error Correction

- Error-correction is more difficult than error-detection.

**1) Error Detection**

- Here, we are checking whether any error has occurred or not.  The answer is a simple YES or NO.
- We are not interested in the number of corrupted-bits.

**2) Error Correction**

- Here, we need to know

→ exact number of corrupted-bits and
→ location of bits in the message.

- Two important factors to be considered: 1) Number of errors and 2) Message-size.

### 3.1.4 Coding

- Redundancy is achieved through various coding-schemes.
  1) Sender adds redundant-bits to the data-bits. This process creates a relationship between → redundant-bits and
     → data-bits.
  2) Receiver checks the relationship between redundant-bits & data-bits to detect/correct errors.
- Two important factors to be considered:
  1) Ratio of redundant-bits to the data-bits and 2) Robustness of the process.
- Two broad categories of coding schemes: 1) Block-coding and 2) Convolution coding.

### 3.2 Block Coding

- The message is divided into k-bit blocks. These blocks are called data-words.
- Here, r-redundant-bits are added to each block to make the length n=k+r.
- The resulting n-bit blocks are called code-words.
- Since n>k, the number of possible code-words is larger than the number of possible data-words.
- Block-coding process is 1-to-1; the same data-word is always encoded as the same code-word.
- Thus, we have $2^n$-$2^k$ code-words that are not used. These code-words are invalid or illegal.

### 3.2.1 Error Detection

- If the following 2 conditions are met, the receiver can detect a change in the original code-word:
  1) The receiver has a list of valid code-words.
  2) The original code-word has changed to an invalid code-words.

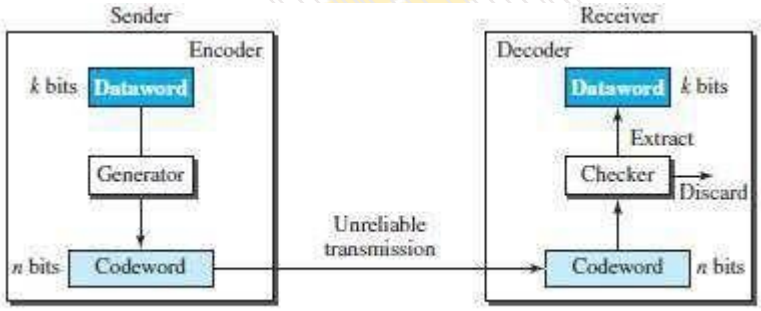



Figure 10.2 *Process of error detection in block coding*

- Here is how it works (Figure 10.2):

**1) At Sender**

i) The sender creates code-words out of data-words by using a generator. The generator applies the rules and procedures of encoding.

ii) During transmission, each code-word sent to the receiver may change.

**2) At Receiver**

i) a) If the received code-word is the same as one of the valid code-words, the code-word is accepted; the corresponding data-word is extracted for use.

b) If the received code-word is invalid, the code-word is discarded.

ii) However, if the code-word is corrupted but the received code-word still matches a valid codeword, the error remains undetected.

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

**Example 3.1**

Let us assume that $k = 2$ and $n = 3$. Table 10.1 shows the list of datawords and codewords.

Table 10.1 *A code for error detection*

| *Dataword* | *Codeword* | *Dataword* | *Codeword* |
|---|---|---|---|
| 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.

### 3.2.1.1 Hamming Distance

- The main concept for error-control: Hamming distance.
- The Hamming distance b/w 2 words is the number of differences between the corresponding bits.
- Let d(x,y) = Hamming distance b/w 2 words x and y.
- Hamming distance can be found by
  → applying the XOR operation on the 2 words and → counting the number of 1s in the result.
- For example:
  1) The Hamming distance d(000, 011) is 2 because 000⊕011= 011 (two 1s).
  2) The Hamming distance d(10101, 11110) is 3 because 10101⊕11110 = 01011 (three 1s). **Hamming Distance and Error**

- Hamming distance between the received word and the sent code-word is the number of bits that are corrupted during transmission.
- For example: Let Sent code-word = 00000
  Received word = 01101
  Hamming distance = d(00000, 01101) =3. Thus, 3 bits are in error.

#### 3.2.1.1.1 Minimum Hamming Distance for Error Detection

- Minimum Hamming distance is the smallest Hamming distance b/w all possible pairs of code-words.
- Let $d_{min}$ = minimum Hamming distance.
- To find $d_{min}$ value, we find the Hamming distances between all words and select the smallest one.

**Minimum-distance for Error-detection**

- If ‘s’ errors occur during transmission, the Hamming distance b/w the sent code-word and received code-word is ‘s’ (Figure 10.3).
- If code has to detect upto ‘s’ errors, the minimum-distance b/w the valid codes must be ‘s+1’ i.e. $d_{min}=s+1$.
- We use a geometric approach to define $d_{min}=s+1$.

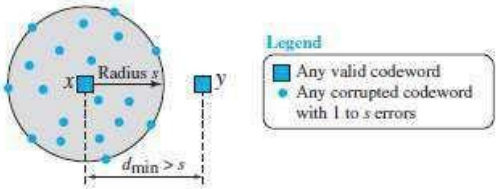



Figure 10.3 *Geometric concept explaining $d_{min}$ in error detection*

¤ Let us assume that the sent code-word x is at the center of a circle with radius s. ¤ All received code-words that are created by 0 to s errors are points inside the circle or on the perimeter of the circle.

¤ All other valid code-words must be outside the circle

• For example: A code scheme has a Hamming distance $d_{min} = 4$.

This code guarantees the detection of upto 3 errors (d = s + 1 or s = 3).

**3.2.1.2 Linear Block Codes**

- Almost all block codes belong to a subset of block codes called linear block codes.
- A linear block code is a code in which the XOR of 2 valid code-words creates another valid code-word. (XOR  Addition modulo-2)

Table 10.1 *A code for error detection*

| *Dataword* | *Codeword* | *Dataword* | *Codeword* |
|---|---|---|---|
| 00 | 000 | 10 | 101 |
| 01 | 011 | 11 | 110 |

- The code in Table 10.1 is a linear block code because the result of XORing any code-word with any other code-word is a valid code-word.

For example, the XORing of the 2nd and 3rd code-words creates the 4th one.

**3.2.1.2.1 Minimum Distance for Linear Block Codes**

- Minimum Hamming distance is no. of 1s in the nonzero valid code-word with the smallest no. of 1s.
- In Table 10.1,

The numbers of 1s in the nonzero code-words are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.

### 3.2.1.3 Parity Check Code

- This code is a linear block code. This code can detect an odd number of errors.
- A k-bit data-word is changed to an n-bit code-word where n=k+1.
- One extra bit is called the parity-bit.
- The parity-bit is selected to make the total number of 1s in the code-word even.
- Minimum hamming distance $d_{min}$ = 2. This means the code is a single-bit error-detecting code.

Table 10.2 *Simple parity-check code C(5, 4)*

| Dataword | Codeword | Dataword | Codeword |
|---|---|---|---|
| 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 |

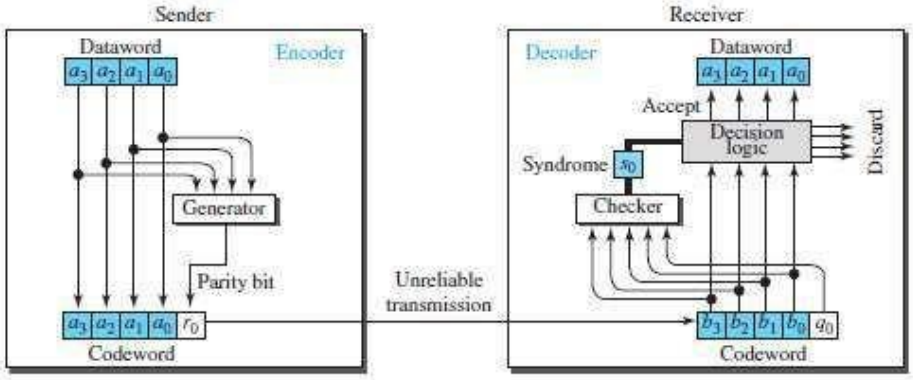



Figure 10.4 *Encoder and decoder for simple parity-check code*

- Here is how it works (Figure 10.4): **1) At Sender**
  - The encoder uses a generator that takes a copy of a 4-bit data-word ($a_0$, $a_1$, $a_2$, and $a_3$) and generates a parity-bit $r_0$.
  - The encoder
    → accepts a copy of a 4-bit data-word ($a_0$, $a_1$, $a_2$, and $a_3$) and → generates a parity-bit $r_0$ using a generator
    → generates a 5-bit code-word
  - The parity-bit & 4-bit data-word are added to make the number of 1s in the code-word even. The addition is done by using the following:

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

  - The result of addition is the parity-bit.
    1) If the no. of 1s in data-word = even, result = 0. ($r_0$=0)
    2) If the no. of 1s in data-word = odd, result = 1. ($r_0$=1)
    3) In both cases, the total number of 1s in the code-word is even.
  - The sender sends the code-word, which may be corrupted during transmission.

**2) At Receiver**

  - The receiver receives a 5-bit word.

- The checker performs the same operation as the generator with one exception: The addition is done over all 5 bits. The result is called the syndrome bit ($s_0$).

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

- Syndrome bit = 0 when the no. of 1s in the received code-word is even; otherwise, it is 1. The syndrome is passed to the decision logic analyzer.
  1) If $s_0$=0, there is no error in the received code-word. The data portion of the received code-word is accepted as the data-word.
  2) If $s_0$=1, there is error in the received code-word. The data portion of the received code-word is discarded. The data-word is not created.

**Example 3.2**

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. 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 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. 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 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.

### 3.3 Cyclic Codes

- Cyclic codes are special linear block codes with one extra property:

  If a code-word is cyclically shifted (rotated), the result is another code-word.

  For ex: if code-word = 1011000 and we cyclically left-shift, then another code-word = 0110001.

- Let First-word = $a_0$ to $a_6$ and Second-word = $b_0$ to $b_6$, we can shift the bits by using the following:

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

#### 3.3.1 Cyclic Redundancy Check (CRC)

- CRC is a cyclic code that is used in networks such as LANs and WANs.

Table 10.3 *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 |

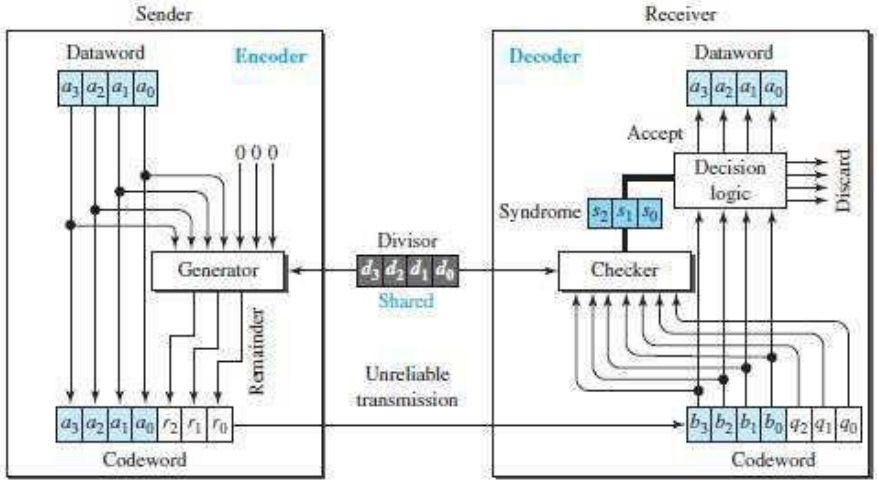



Figure 10.5 *CRC encoder and decoder*

- Let Size of data-word = k bits (here k=4).

  Size of code-word = n bits (here n=7).

  Size of divisor = n-k+1 bits (here n-k+1=4). (Augmented □ increased)

- Here is how it works (Figure 10.5):

  **1) At Sender**

  - n-k 0s is appended to the data-word to create augmented data-word. (here n-k=3). □ The augmented data-word is fed into the generator (Figure 10.6).
  - The generator divides the augmented data-word by the divisor.
  - The remainder is called check-bits ($r_2r_1r_0$).
  - The check-bits ($r_2r_1r_0$) are appended to the data-word to create the code-word.

  **2) At Receiver**

  - The possibly corrupted code-word is fed into the checker. □ The checker is a replica of the generator.
  - The checker divides the code-word by the divisor.
  - The remainder is called syndrome bits ($r_2r_1r_0$).
  - The syndrome bits are fed to the decision-logic-analyzer.

- The decision-logic-analyzer performs following functions:

**i) For No Error**

¤ If all syndrome-bits are 0s, the received code-word is accepted. ¤ Data-word is extracted from received code-word (Figure 10.7a).

**ii) For Error**

¤ If all syndrome-bits are not 0s, the received code-word is discarded (Figure 10.7b).

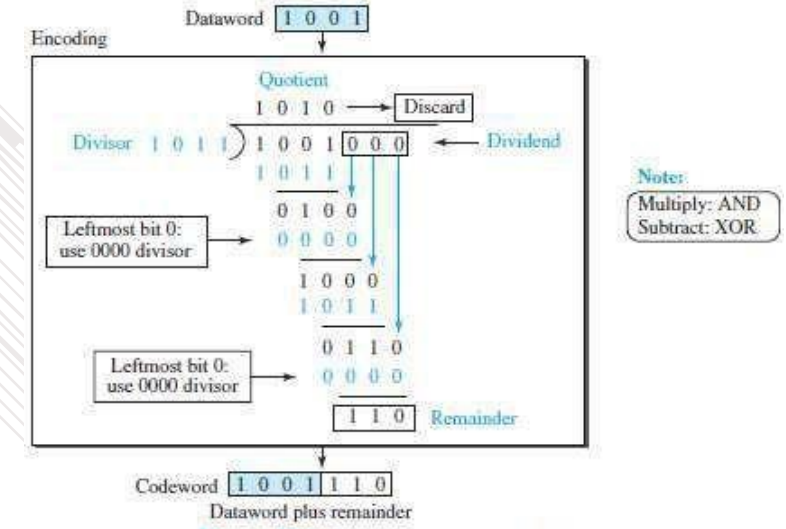



**Figure 10.6** *Division in CRC encoder*

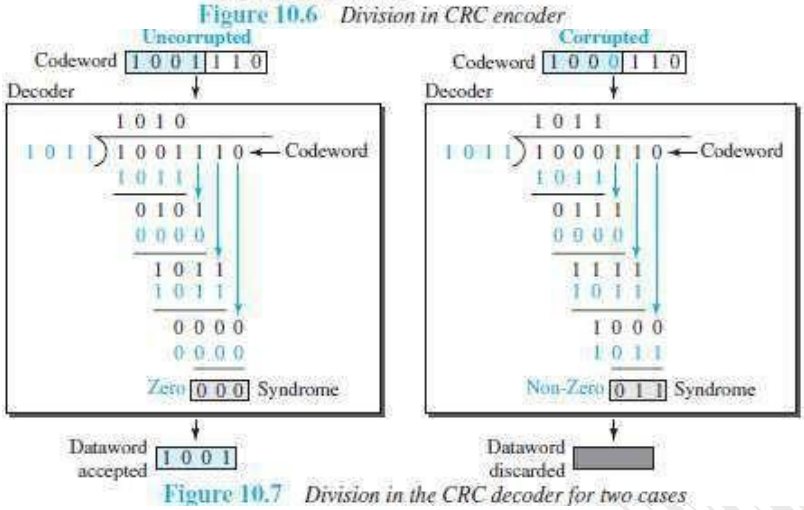
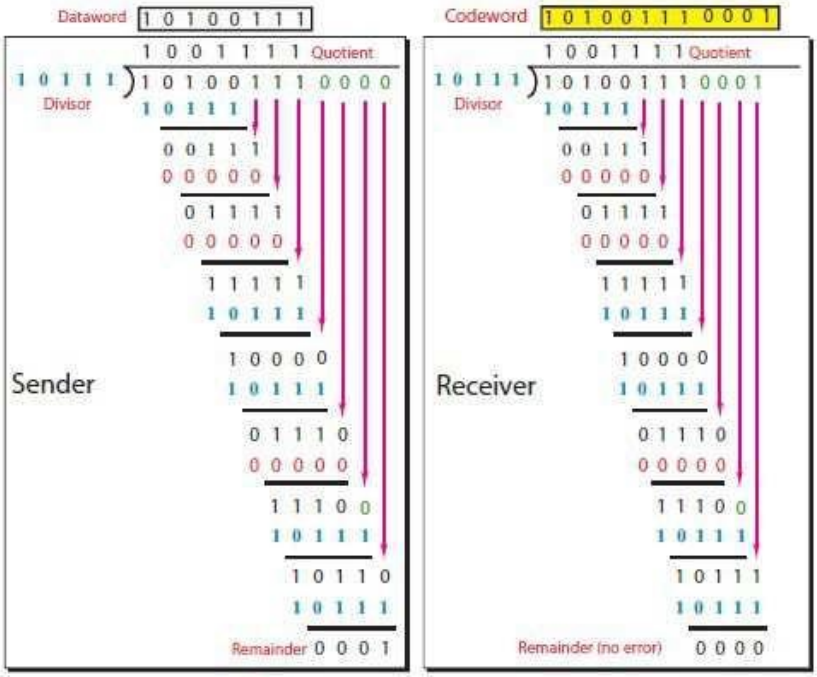



**Figure 10.7** *Division in the CRC decoder for two cases*

**Example 3.3**

Given the dataword 10100111 and the divisor 10111, show the generation of the CRC codeword at the sender site (using binary division).




### 3.3.2 Polynomials

- A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1 (Figure 10.8).
- The power of each term shows the position of the bit; the coefficient shows the value of the bit.

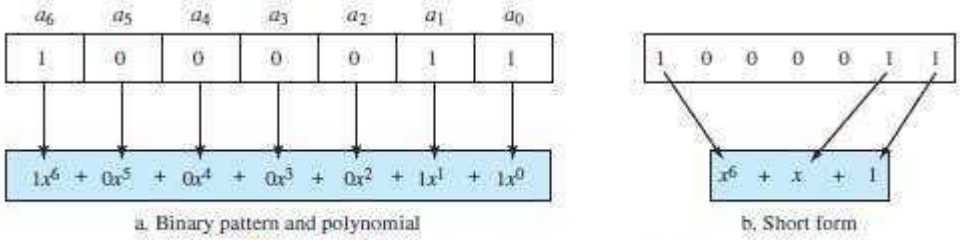



Figure 10.8 *A polynomial to represent a binary word*

### 3.3.3 Cyclic Code Encoder Using Polynomials

- Let Data-word = 1001 = $x^3+1$. Divisor = 1011 = $x^3+x+1$.
- In polynomial representation, the divisor is referred to as generator polynomial t(x) (Figure 10.9).

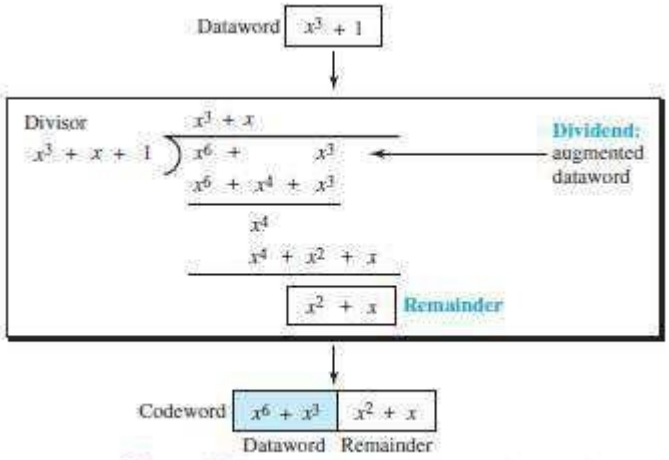



**Figure 10.9** *CRC division using polynomials*

### 3.3.4 Cyclic Code Analysis

• We 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: $e(x)$**

**In a cyclic code,**

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

**Single Bit Error**

• If the generator has more than one term and the coefficient of $x^0$ is 1, all single-bit errors can be caught.

**Two Isolated Single-Bit Errors**

• If a generator cannot divide $x^i+1$ (t between 0 & n-1), then all isolated double errors can be detected (Figure 10.10).

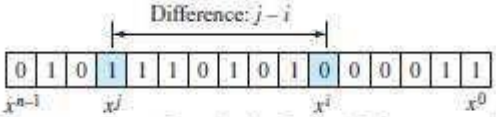



**Figure 10.10** *Representation of two isolated single-bit errors using polynomials*

**Odd Numbers of Errors**

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

**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 $x + 1$.**

**Standard Polynomials**

Table 10.4 *Standard polynomials*

| *Name* | *Polynomial* | *Used in* |
|---|---|---|
| CRC-8 | $x^8+x^2+x+1$<br>**100000111** | ATM header |
| CRC-10 | $x^{10}+x^9+x^5+x^4+x^2+1$<br>**11000110101** | ATM AAL |
| CRC-16 | $x^{16}+x^{12}+x^5+1$<br>**10001000000100001** | HDLC |
| CRC-32 | $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$<br>**10000010011000001000111010110111** | LANs |

### 3.3.5 Advantages of Cyclic Codes

- The cyclic codes have a very good performance in detecting
  - → single-bit errors
  - → double errors
  - → odd number of errors and
  - → burst-errors.
- They can easily be implemented in hardware and software. They are fast when implemented in hardware.

### 3.4 Checksum

- Checksum is an error-detecting technique.
- In the Internet,
  - → The checksum is mostly used at the network and transport layer. → The checksum is not used in the data link layer.
- Like linear and cyclic codes, the checksum is based on the concept of redundancy.
- Here is how it works (Figure 10.15):

**1) At Source**

- Firstly the message is divided into m-bit units.
- Then, the generator creates an extra m-bit unit called the checksum. The checksum is sent with the message.

**2) At Destination**

The checker creates a new checksum from the combination of the message and sentchecksum.

i) If the new checksum is all 0s, the message is accepted.

ii) If the new checksum is not all 0s, the message is discarded.

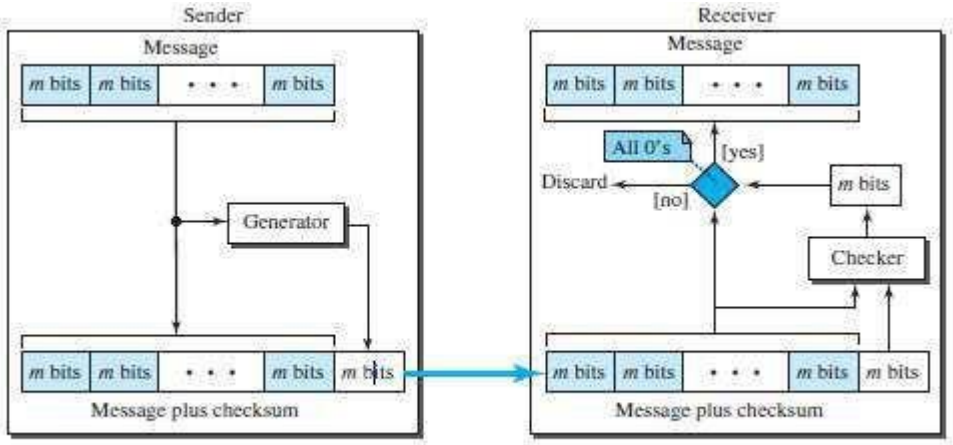



Figure 10.15 *Checksum*

### 3.4.1 Concept of Checksum

Consider the following example: ***Example 3.4***

- 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:

  Let set of numbers = (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 result & the sum 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 3.5***

- To make the job of the receiver easy if we send the negative (complement) of the sum, called thechecksum.
- 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.

#### 3.4.1.1 One's Complement

- The previous example has one major drawback.

  All of our data can be written as a 4-bit word (they are less than 15) except for the checksum.
- Solution: Use one's complement arithmetic.
  - We can represent unsigned numbers between 0 and $2^n$-1 using only n bits.
  - If the number has more than n bits, the extra leftmost bits need to be added to the n rightmost bits (wrapping).
  - A negative number can be represented by inverting all bits (changing 0 to 1 and 1 to 0). □ This is the same as subtracting the number from $2^n$-1.

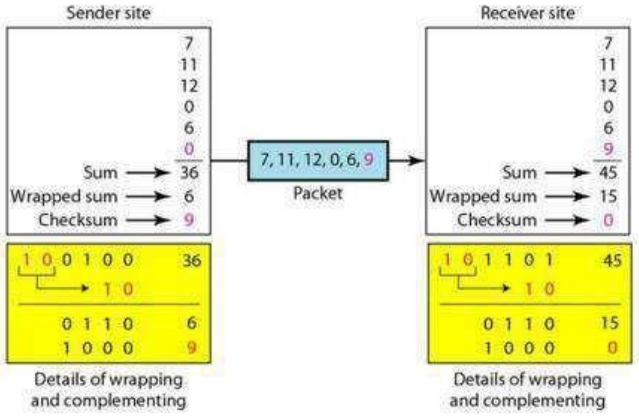



Figure 10.16

- Here is how it works (Figure 10.16):

**1) At Sender**

- The sender initializes the checksum to 0 and adds all data items and the checksum.  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.  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.

**2) At Receiver**

- 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.

### 3.4.1.2 Internet Checksum

- Traditionally, the Internet has been using a 16-bit checksum.
- The sender or the receiver uses five steps.

Table 10.5 *Procedure to calculate the traditional checksum*

| *Sender* | *Receiver* |
|---|---|
| 1. The message is divided into 16-bit words. | 1. The message and the checksum are 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. |

### 3.4.1.3 Algorithm

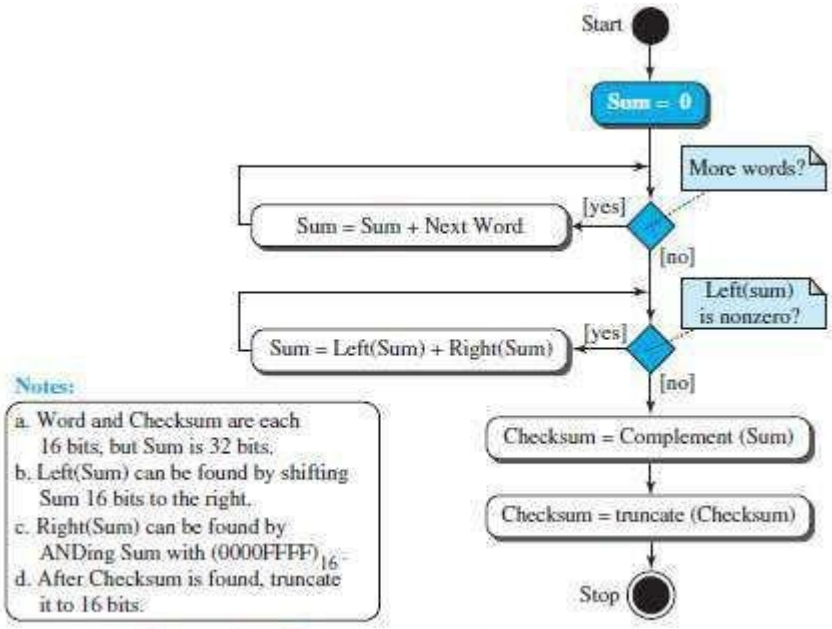



Figure 10.17 *Algorithm to calculate a traditional checksum*

### 3.4.2 Other Approaches to the Checksum

- If two 16-bit items are transposed in transmission, the checksum cannot catch this error.
- The reason is that the traditional checksum is not weighted: it treats each data item equally.
- In other words, the order of data items is immaterial to the calculation.
- Two approaches have been used to prevent this problem: 1)Fletcher and 2)Adler

### 3.4.2.1 Fletcher Checksum

- The Fletcher checksum was devised to weight each data item according to its position.
- Fletcher has proposed two algorithms: 8-bit and 16-bit (Figure 10.18).
- The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum.

  The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.
- The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum.
- The calculation is done modulo 256 ($2^8$), which means the intermediate results are divided by 256 and the remainder is kept.
- The algorithm uses two accumulators, L and R.
- The first simply adds data items together; The second adds a weight to the calculation.

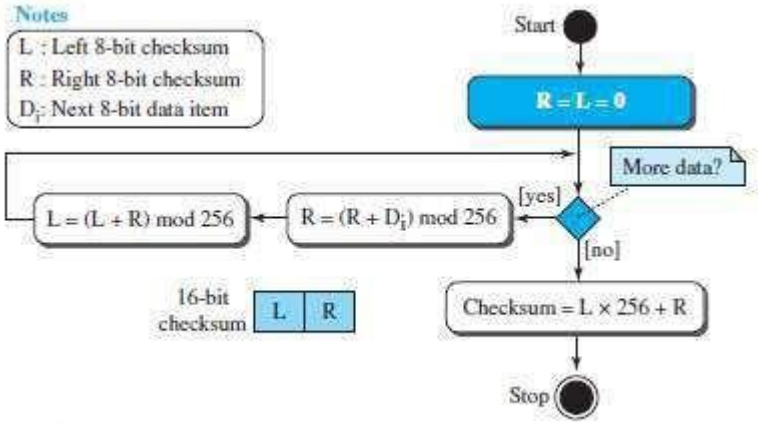



**Figure 10.18** *Algorithm to calculate an 8-bit Fletcher checksum*

### 3.4.2.2 Adler Checksum

- The Adler checksum is a 32-bit checksum.
- It is similar to the 16-bit Fletcher with three differences (Figure 10.19).
    1) Calculation is done on single bytes instead of 2 bytes at a time.
    2) The modulus is a prime number (65,521) instead of 65,536. 3) L is initialized to 1 instead of 0.
- A prime modulo has a better detecting capability in some combinations of data.

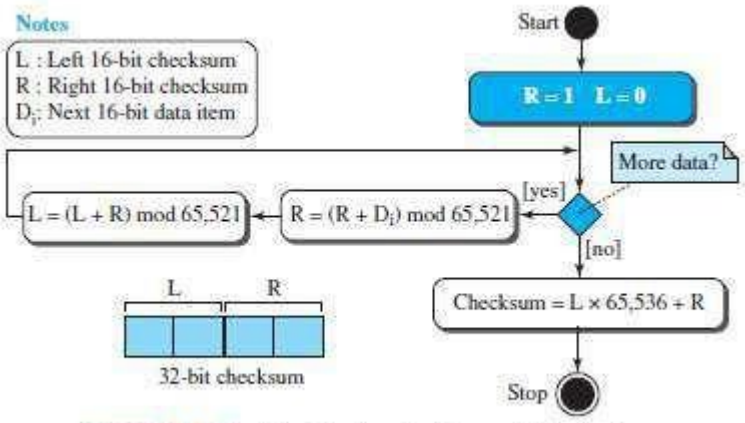



**Figure 10.19** *Algorithm to calculate an Adler checksum*

## 3.5 FORWARD ERROR CORRECTION

- Retransmission of corrupted and lost packets is not useful for real-time multimedia transmission because it creates an unacceptable delay in reproducing: we need to wait until the lost or corrupted packet is resent.
- We need to correct the error or reproduce the packet immediately.
- Several schemes have been designed and used that are collectively referred to as forward error correction (FEC) techniques.

### 3.5.1 Using Hamming Distance

- To detect $t$ errors, we need to have $d_{min} = 2t + 1$ (Figure 10.20).
- In other words, if we want to correct 10 bits in a packet, we need to make the minimum hamming distance 21 bits, which means a lot of redundant bits need to be sent with the data.

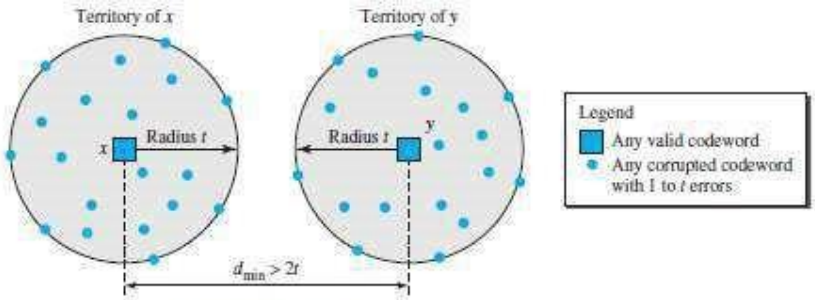



Figure 10.20 *Hamming distance for error correction*

### 3.5.2 Using XOR

- Use the property of the exclusive OR operation as shown below.

$$R = P_1 \oplus P_2 \oplus \ldots \oplus P_i \oplus \ldots \oplus P_N \quad \rightarrow \quad P_i = P_1 \oplus P_2 \oplus \ldots \oplus R \oplus \ldots \oplus P_N$$

- We divide a packet into *N* chunks, create the exclusive OR of all the chunks and send *N* + 1 chunks.
- If any chunk is lost or corrupted, it can be created at the receiver site.
- If N = 4, it means that we need to send 25 percent extra data and be able to correct the data if only one out of four chunks is lost.

### 3.5.3 Chunk Interleaving

- Another way to achieve FEC in multimedia is to allow some small chunks to be missing at thereceiver.
- We cannot afford to let all the chunks belonging to the same packet be missing. However, we can afford to let one chunk be missing in each packet.

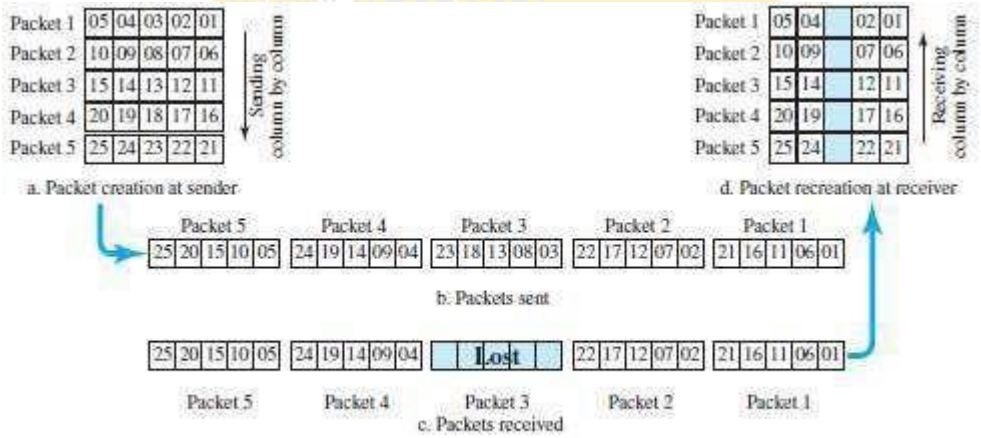



Figure 10.21 *Interleaving*

- In Figure 10.21, each packet is divided into 5 chunks (normally the number is much larger).
- Then, we can create data chunk-by-chunk (horizontally), but combine the chunks into packets vertically.
- In this case, each packet sent carries a chunk from several original packets.
- If the packet is lost, we miss only one chunk in each packet.
- Normally, missing of a chunk is acceptable in multimedia communication.

### 3.5.4 Combining Hamming Distance and Interleaving •

Hamming distance and interleaving can be combined.

- Firstly, we can create n-bit packets that can correct t-bit errors.
- Then, we interleave m rows and send the bits column-by-column.
- In this way, we can automatically correct burst-errors up to m×t bit errors.