Chapter 8

Reliability and Channel Coding

This slide is adopted from Lami Kaya & Douglas Comer

Topics Covered

- 8.1 Introduction
- 8.2 The Three Main Sources of Transmission Errors
- 8.3 Effect of Transmission Errors on Data
- 8.4 Two Strategies for Handling Channel Errors
- 8.5 Block and Convolutional Error Codes
- 8.6 An Example Block Error Code: Single Parity Checking
- 8.7 The Mathematics of Block Error Codes and (n, k) Notation
- 8.8 Hamming Distance: A Measure of a Code's Strength
- 8.9 The Hamming Distance Among Strings in a Codebook
- 8.10 The Tradeoff Between Error Detection and Overhead
- 8.11 Error Correction with Row and Column (RAC) Parity
- 8.12 The 16-Bit Checksum Used in the Internet
- 8.13 Cyclic Redundancy Codes (CRCs)
- 8.14 An Efficient Hardware Implementation of CRC
- 8.15 Automatic Repeat reQuest (ARQ) Mechanisms

8.1 Introduction

- This chapter
  - continues the discussion by examining errors that can occur during transmission and techniques that can be used to control errors
  - The concepts presented here
    - are fundamental to computer networking
    - and are used in communication protocols at many layers of the stack

8.2 The Three Main Source of Transmission Errors

- **Interference**
  - electromagnetic radiation emitted from devices such as electric motors
  - and background cosmic radiation
  - can cause noise that can disturb radio transmissions and signals traveling across wires
- **Attenuation**
  - As a signal passes across a medium, the signal becomes weaker
    - signals on wires or optical fibers become weaker over long distances, just as a radio signal becomes weaker with distance

8.2 The Three Main Source of Transmission Errors

- **Distortion**
  - All physical systems distort signals
  - As a pulse travels along an optical fiber, the pulse disperses
  - Wires have properties of capacitance and inductance
    - that block signals at some frequencies while admitting signals at other frequencies
  - Placing a wire near a large metal object can change the set of frequencies that can pass through the wire
  - Metal objects can block some frequencies of radio waves, while passing others

8.2 The Three Main Source of Transmission Errors

- Shannon's Theorem suggests one way to reduce errors:
  - increase the signal-to-noise ratio (either by increasing the signal or lowering noise if possible)

$$C = B \log_2(1 + S/N)$$

where

- C is the effective limit on the channel capacity in bits per second
- B is the hardware bandwidth
- S/N is the signal-to-noise ratio, the ratio of the average signal power divided by the average noise power

  - Mechanisms like shielded wiring can help lower noise
  - But, Physical transmission system is always susceptible to errors
    - it may not be possible to change the signal-to-noise ratio

8.2 The Three Main Source of Transmission Errors

- Noise/interference cannot be eliminated completely
  - But many transmission errors can be detected
  - In some cases, errors can be corrected automatically
- We will see that error detection adds overhead
- Error handling is a tradeoff
  - in which a system designer must decide whether a given error is likely to occur, and if so, what the consequences will be

8.3 Effect of Transmission Errors on Data

- Below table lists the three principal ways transmission errors affect data

Type of Error	Description
Single Bit Error	A single bit in a block of bits is changed and all other bits in the block are unchanged (often results from very short-duration interference)
Burst Error	Multiple bits in a block of bits are changed (often results from longer-duration interference)
Erasure (Ambiguity)	The signal that arrives at a receiver is ambiguous (does not clearly correspond to either a logical 1 or a logical 0 – can result from distortion or interference)

8.3 Effect of Transmission Errors on Data

- For a burst error, the burst size, or length, is defined as the number of bits from the start of the corruption to the end of the corruption
- Figure 8.2 illustrates the definition

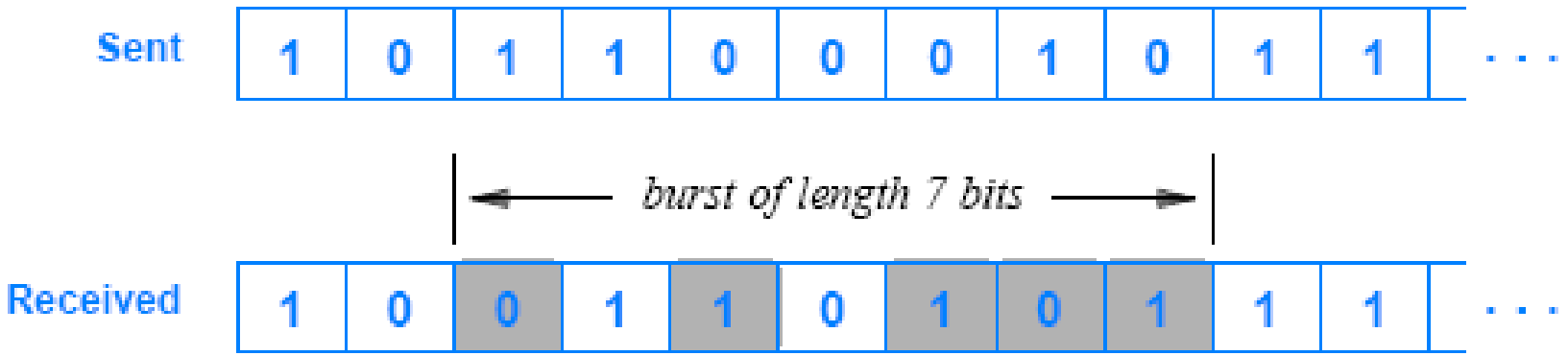


Figure 8.2 Illustration of a burst error with changed bits marked in gray.

8.4 Two Strategies for Handling Channel Errors

- A variety of mathematical techniques have been developed that overcome errors during transmission and increase reliability
  - Known collectively as channel coding
- The techniques can be divided into two broad categories:
  - Forward Error Correction (FEC) mechanisms
  - Automatic Repeat reQuest (ARQ) mechanisms

- The basic idea of FEC is straightforward:
  - add additional information to data that allows a receiver to verify that data arrives correctly and to correct errors (if possible)

8.4 Two Strategies for Handling Channel Errors

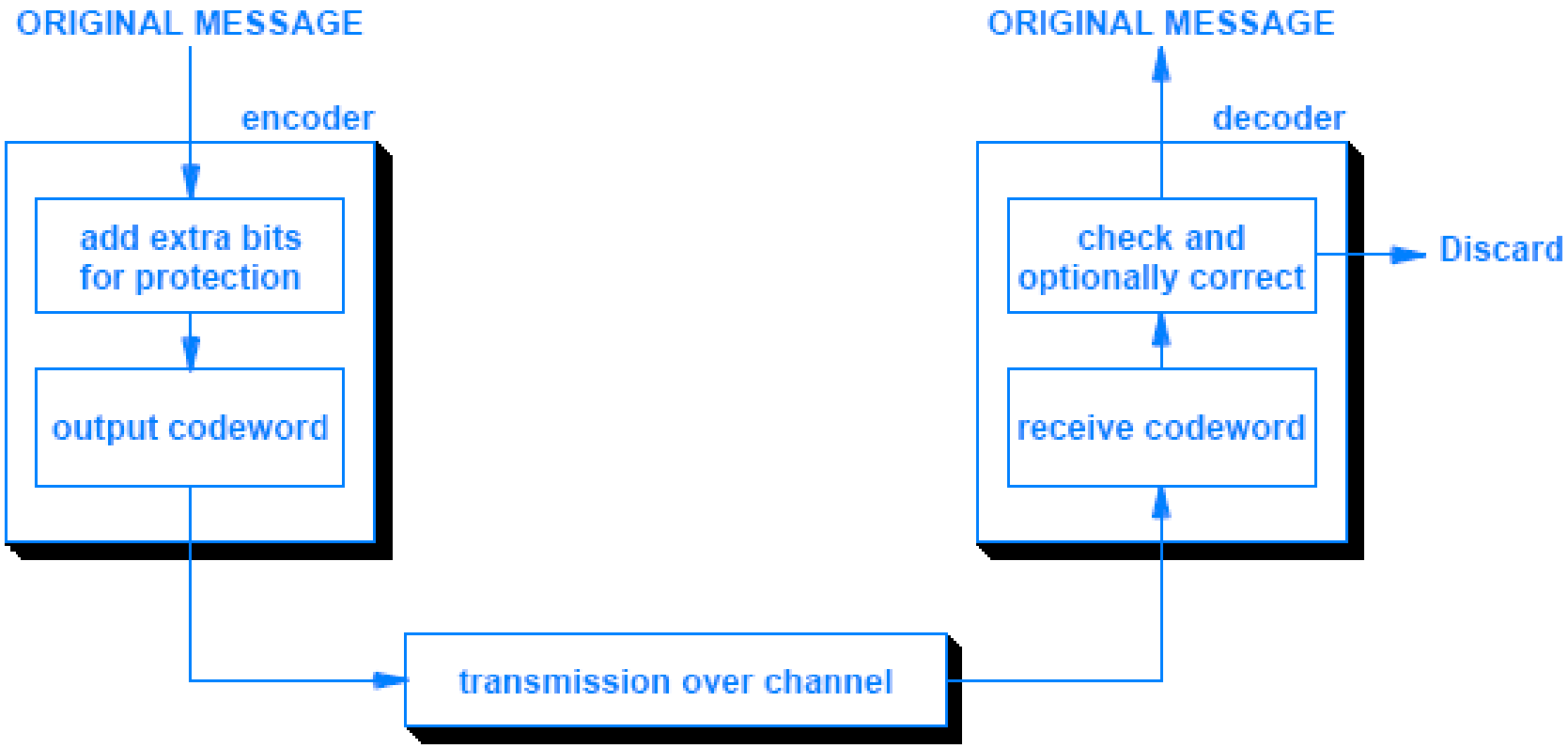
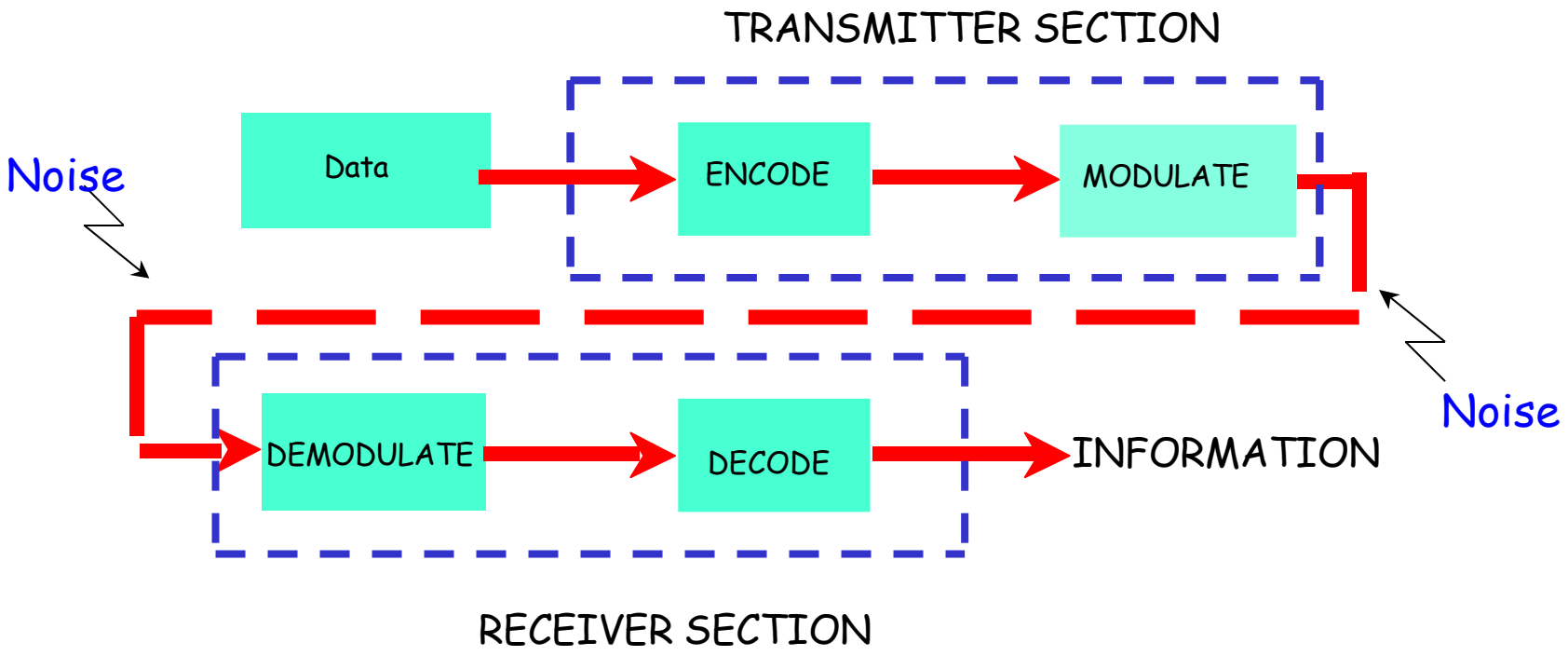


Figure 8.3 The conceptual organization of a forward error correction mechanism.

8.4 Two Strategies for Handling Channel Errors

- Basic error detection mechanisms allow a receiver to detect when an error has occurred;
- FEC (Forward Error Correction) mechanisms allow a receiver to determine
  - exactly which bits have been changed and to compute correct values
- The second approach to channel coding, known as an ARQ (Automatic Repeat reQuest),
  - requires the cooperation of a sender; a sender and receiver exchange messages to insure that all data arrives correctly

A Typical Error Correcting Communication System.



An FEC communication system adds information at the transmitter for error correction of corrupted data at the receiver.

Forward Error Correction (FEC) Applications.

- As previously discussed, forward error correction is used at the receiving end to detect errors and correct for them.
- Some applications include CD and DVD players, HDTV, Data storage systems, Wireless communications, Satellite communications, and Modem technologies.
- Designers have implemented FEC circuits in Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), and Field Programmable Gate Arrays (FPGAs).
- FEC techniques can be realized for both Block codes and for Convolutional codes

8.5 Block and Convolutional Error Codes

- The two types of FEC techniques satisfy separate needs:
  - Block Error Codes
    - It divides the data to be sent into a set of blocks
    - It attaches extra information known as redundancy to each block
    - The encoding for a given block of bits depends only on the bits themselves, not on bits that were sent earlier
    - They are memory-less in the sense that the encoding mechanism does not carry state information from one block of data to the next
  - Convolutional Error Codes
    - It treats data as a series of bits, and computes a code over a continuous series
    - Thus, the code computed for a set of bits depends on the current input and some of the previous bits in the stream
    - Convolutional codes are said to be codes with memory

8.6 An Example Block Error Code: Single Parity Checking

- How can additional information be used to detect errors?
  - consider a single parity checking (SPC) mechanism
  - One form of SPC defines a block to be an 8-bit unit of data (i.e., a single byte)
    - On the sending side, an encoder adds an extra bit, called a parity bit, to each byte before transmission
    - A receiver uses parity bit to check whether bits in the byte are correct
    - Before parity can be used, the sender and receiver must be configured for either even parity or odd parity
- Figure 8.4 lists examples of data bytes and the value of the parity bit that is sent when using even or odd parity

8.6 An Example Block Error Code: Single Parity Checking

Original Data	Even Parity	Odd Parity
0 0 0 0 0 0 0 0	0	1
0 1 0 1 1 0 1 1	1	0
0 1 0 1 0 1 0 1	0	1
1 1 1 1 1 1 1 1	0	1
1 0 0 0 0 0 0 0	1	0
0 1 0 0 1 0 0 1	1	0

Figure 8.4 Data bytes and the corresponding value of a single parity bit when using even parity or odd parity.

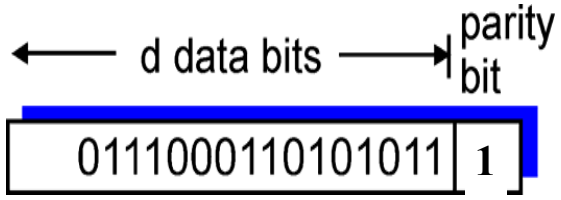
8.6 An Example Block Error Code: Single Parity Checking

- SPC is a weak form of channel coding that can detect errors
  - but cannot correct errors
  - An even parity mechanism can only handle errors where an odd number of bits are changed
    - If one of the nine bits (including the parity bit) is changed during transmission, the receiver will declare that the incoming byte is invalid
    - However, if a burst error occurs in which two, four, six, or eight bits change value, the receiver will incorrectly classify the incoming byte as valid

Example of Block Error Code

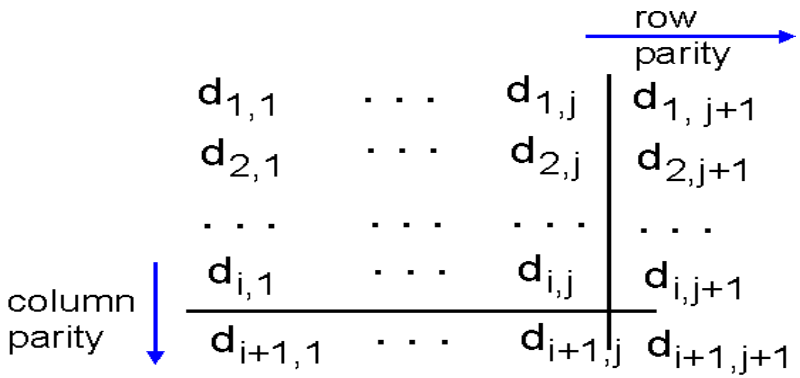
Single Bit Parity:

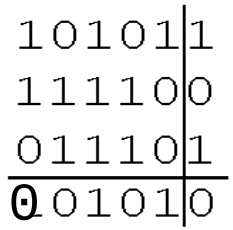
Detect single bit errors



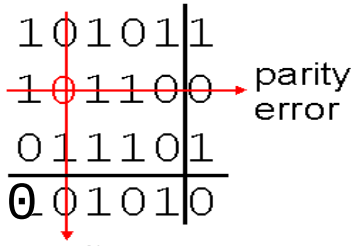
Two Dimensional Bit Parity (RAC):

Detect *and correct* single bit errors





no errors



correctable single bit error

8.7 The Mathematics of Block Error Codes and Notation

- Mathematically
  - we define the set of all possible messages to be a set of datawords
  - and define the set of all possible encoded versions to be a set of codewords
- If a dataword contains k bits and r additional bits are added to form a codeword, we say that the result is an

  (n, k) encoding scheme
  - where $n = k + r$
- The key to successful error detection lies in choosing a subset of the 2^n possible combinations that are valid codewords
  - The valid subset is known as a codebook

8.7 The Mathematics of Block Error Codes and Notation

- Error Detection
  - Example
    - Let us assume that k = 2 and n = 3. The below table shows the list of data-words and code-words. Later, we will see how to derive a code-word from a data-word.

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

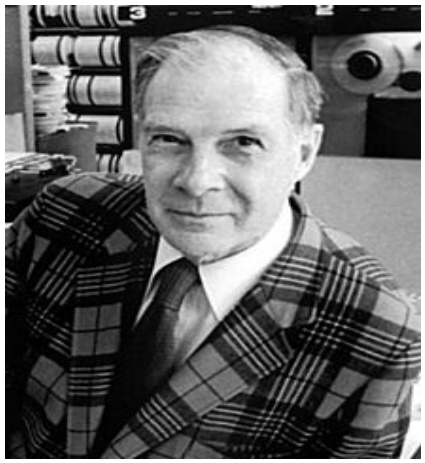
Assume the sender encodes the dataword "01" as "011" and send it to the receiver

1) If the receiver receives 011 ?
   - Valid codeword and extract 01
2) If the receiver receives 111 (the left most bit is corrupted)?
   - Invalid codeword and discard
3) If the receiver receives 000 ? (the right two bits are corrupted)
   - Valid codeword and extract 00 (detected ?)

8.7 The Mathematics of Block Error Codes and Notation

- As an example, consider SPC(Single Parity Check)
  - The set of datawords consists of any possible combination of 8-bits
    - Thus, k = 8 and there are 2^8 or 256 possible data words
    - The set of n = 9 bits, so there are 2^9 or 512 possibilities
      - However, only half of the 512 values form valid codewords
- Think of the set of all possible n-bit values and the valid subset that forms the codebook
  - If an error occurs during transmission
    - one or more of the bits in a codeword will be changed, which will either produce another valid codeword or an invalid combination
  - For the example discussed above
    - a change to a single bit of a valid codeword produces an invalid combination, but changing two bits produces another valid codeword

8.8 Hamming Distance: A Measure of a Code's Strength



- No channel coding scheme is ideal!
  - changing enough bits will always transform to a valid codeword
- What is the minimum number of bits of a valid codeword that must be changed to produce another valid codeword?
  - To answer the question, engineers use a measure known as the Hamming distance, named after a theorist at Bell Laboratories who was a pioneer in the field of information theory and channel coding
  - Given two strings of n bits each, the Hamming distance is defined as the number of differences
  - Figure 8.5 illustrates the definition
  - ECC memory uses Hamming Codes

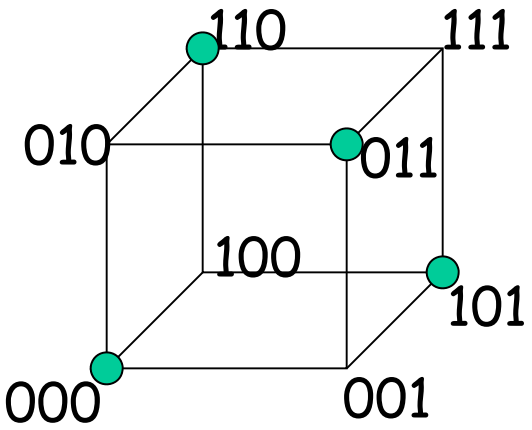
8.8 Hamming Distance: A Measure of a Code's Strength

d(000,001) = 1	d(000,101) = 2
d(101,100) = 1	d(001,010) = 2
d(110,001) = 3	d(111,000) = 3

Figure 8.5 Examples of Hamming distance for various pairs of 3-bit strings.

8.8 Hamming Distance

- *Hamming distance* (d) of two bit strings = number of bit positions in which they differ.
  - e.g.: d (000,011) = 2, (000 ⊕ 011) = 011 (two 1s)
  - d (10101,11110) = 3, (10101 ⊕ 11110) = 01011 (three 1s)



- codeword

d_{min} = 2 = min distance
n = 3 = dimensionality
2^n = 8 = number of nodes

8.8 Hamming Distance: A Measure of a Code's Strength

- One way to compute the Hamming distance consists of
  - taking the exclusive or (xor) between two strings
  - and counting the number of 1 bits in the answer
  - For example, consider the Hamming distance between strings 110 and 011

    - The xor of the two strings is:

      $$110 \oplus 011 = 101$$

      which contains two 1 bits
    - Therefore, the Hamming distance between 011 and 101 is 2

8.9 The Hamming Distance Among Strings in a Codebook

- Errors can transform a valid codeword into another valid codeword
  - To measure such transformations, we compute the Hamming distance between all pairs of codewords in a given codebook
- Consider odd parity applied to 2-bit datawords
- Figure 8.6 lists the four (4) possible datawords
  - the 4 possible codewords that result from appending a parity bit
  - and the Hamming distances for pairs of codewords
- We use to denote the minimum Hamming distance, d_{min} among pairs in a codebook
  - The concept gives an answer to the question above
  - How many bit errors can cause a transformation from one valid codeword into another valid codeword?

8.9 The Hamming Distance Among Strings in a Codebook

Dataword	Codeword
0 0	0 0 1
0 1	0 1 0
1 0	1 0 0
1 1	1 1 1

(a)

d(001,010) = 2	d(010,100) = 2
d(001,100) = 2	d(010,111) = 2
d(001,111) = 2	d(100,111) = 2

(b)

Figure 8.6 (a) The datawords and codewords for a single parity encoding of 2-bit data strings, and (b) the Hamming distance for all pairs of codewords.

8.9 The Hamming Distance Among Strings in a Codebook

- In the SPC example of Figure 8.6, the set consists of the Hamming distance between each pair of codewords, and $d_{min}=2$
  - The definition means that there is at least one valid codeword that can be transformed into another valid codeword, if 2-bit errors occur during transmission

- To find the minimum number of bit changes that can transform a valid codeword into anther valid codeword, compute the minimum Hamming distance between all pairs in the codebook.

8.10 The Tradeoff Between Error Detection and Overhead

- A large value of d_{min} is desirable
  - because the code is immune to more bit errors, if fewer than d_{min} bits are changed, the code can detect that error(s) occurred
- Equation 8.1 specifies the relationship between d_{min} and e, the maximum number of bit errors that can be detected:

$$e = d_{\min} - 1$$

- A code with a higher value of d_{min} sends more redundant information than an error code with a lower value of d_{min}
- Code rate (R) that gives the ratio of a dataword size to the codeword size as shown in Equation 8.2

$$R = \frac{k}{n}$$

  - Where k = size of actual data, n=size of dataword

8.11 Error Correction with Row and Column (RAC) Parity

- Imagine an array of 3-rows and 4-columns, with a parity bit added for each row and for each column
  - Figure 8.7 illustrates the arrangement, which is known as a Row and Column (RAC) code
    - Example RAC has n= 20, which means that it is a (20, 12) code

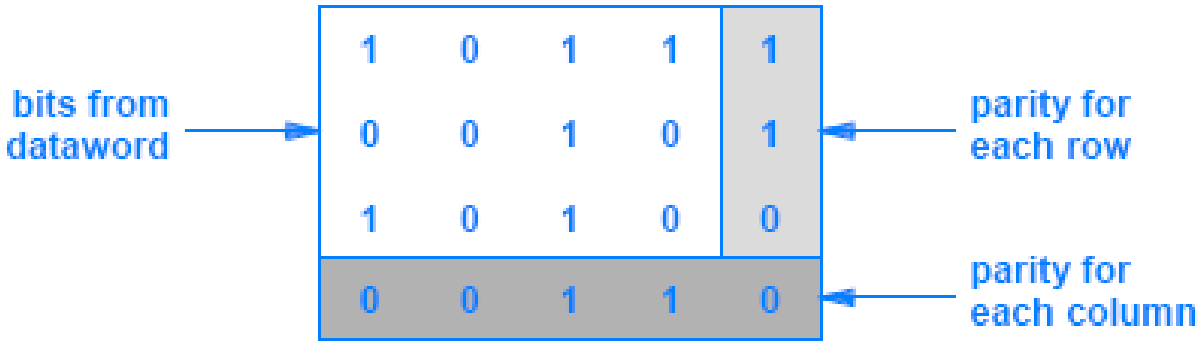
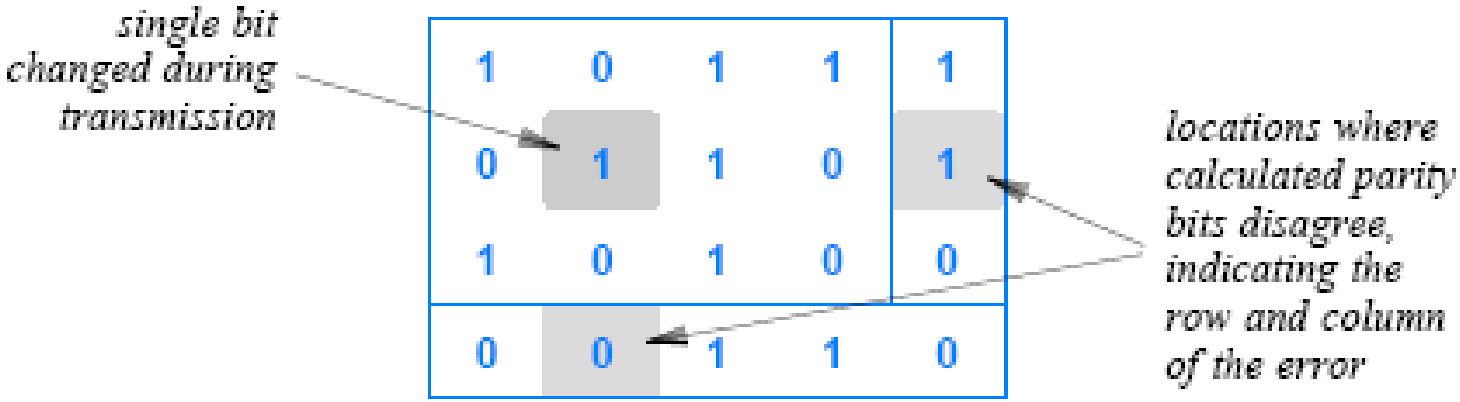


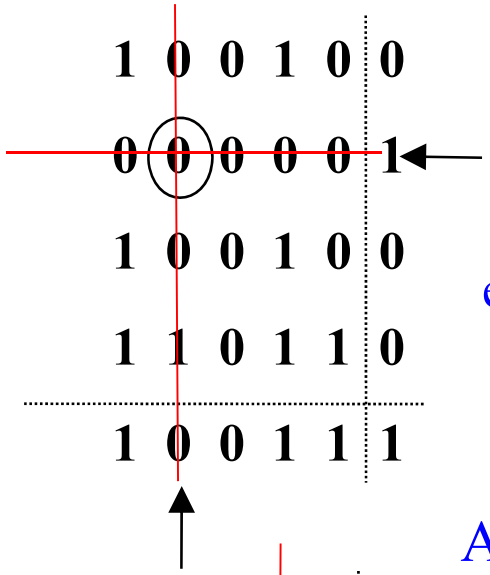
Figure 8.7 An example of row and column encoding with data bits arranged in a 3 × 4 array and an even parity bit added for each row and each column.

8.11 Error Correction with Row and Column (RAC) Parity

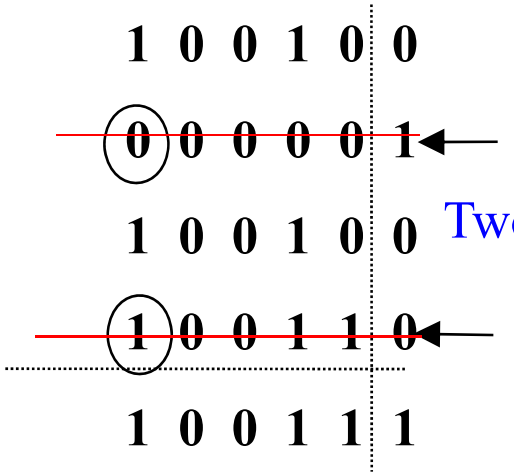
- How error correction works? Assume that one of the bits in Figure 8.7 (below) is changed during transmission:
  - When the receiver arranges the bits into an array and parity bits are recalculated
    - two of the calculations will disagree with the parity bits received, as Figure 8.8 illustrates
    - a single bit error will cause two calculated parity bits to disagree with the parity bit received



Example of RAC

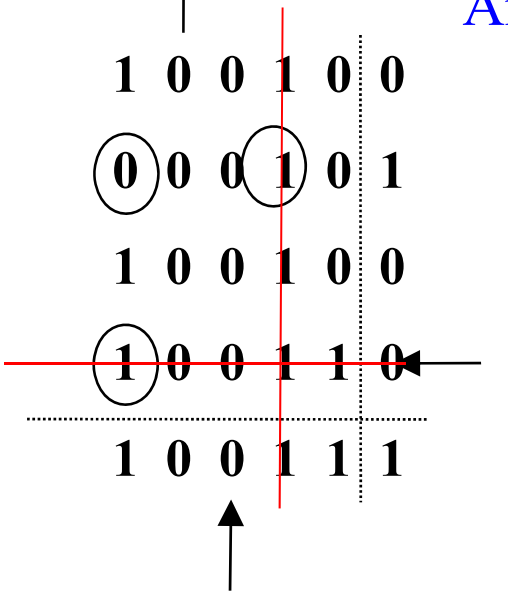


One error

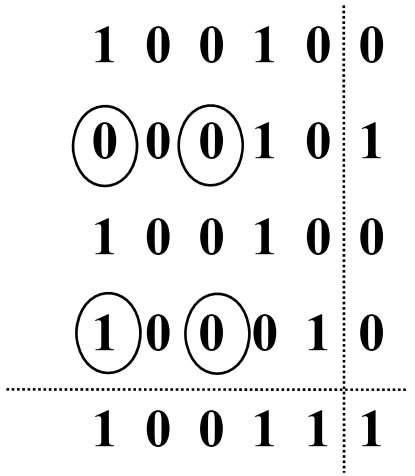


Two errors

Arrows indicate error checked bits



Three errors



Four errors

8.11 Error Correction with Row and Column (RAC) Parity

- The two disagreements correspond to the row and column position of the error
- A receiver uses the calculated parity bits to determine exactly which bit is in error, and then corrects the bit
  - Thus, an RAC can correct any error that changes a single data bit
- What happens to an RAC code if an error changes more than one bit in a given block?
  - RAC can only correct single-bit errors
  - In cases where two or three bits are changed
    - an RAC encoding will be able to detect an odd number of errors

8.12 The 16-Bit Checksum Used in the Internet

- A particular coding scheme plays a key role in the Internet
  - Known as the Internet checksum, the code consists of a 16-bit 1s complement checksum
- The Internet checksum does not impose a fixed size on a dataword
  - the algorithm allows a message to be arbitrarily long
  - and computes a checksum over the entire message
- The Internet checksum treats data in a message as a series of 16-bit integers, as Figure 8.9 below illustrates

8.12 The 16-Bit Checksum Used in the Internet

- To compute a checksum, a sender
  - adds the numeric values of the 16-bit integers
  - and it transmits the result
- To validate the message, a receiver performs the same computation
- Algorithm 8.1 gives the details of the computation
  - The checksum is computed in 1s complement arithmetic
  - and uses 16 bit integers instead of 32 or 64 bit integers
- During the for loop, the addition may overflow
  - Thus, following the loop, the algorithm adds the overflow (the high-order bits) back into the sum
- Why is a checksum computed as the arithmetic inverse of the sum instead of the sum?

8.12 The 16-Bit Checksum Used in the Internet

Algorithm 8.1

Given:

A message, M, of arbitrary length

Compute:

A 16-bit 1s complement checksum, C, using 32-bit arithmetic

Method:

```
Pad M with zero bits to make an exact multiple of 16 bits
Set a 32-bit checksum integer, C, to 0;
for (each 16-bit group in M) {
    Treat the 16 bits as an integer and add to C;
}
Extract the high-order 16 bits of C and add them to C;
The inverse of the low-order 16 bits of C is the checksum;
If the checksum is zero, substitute the all 1s form of zero.
```

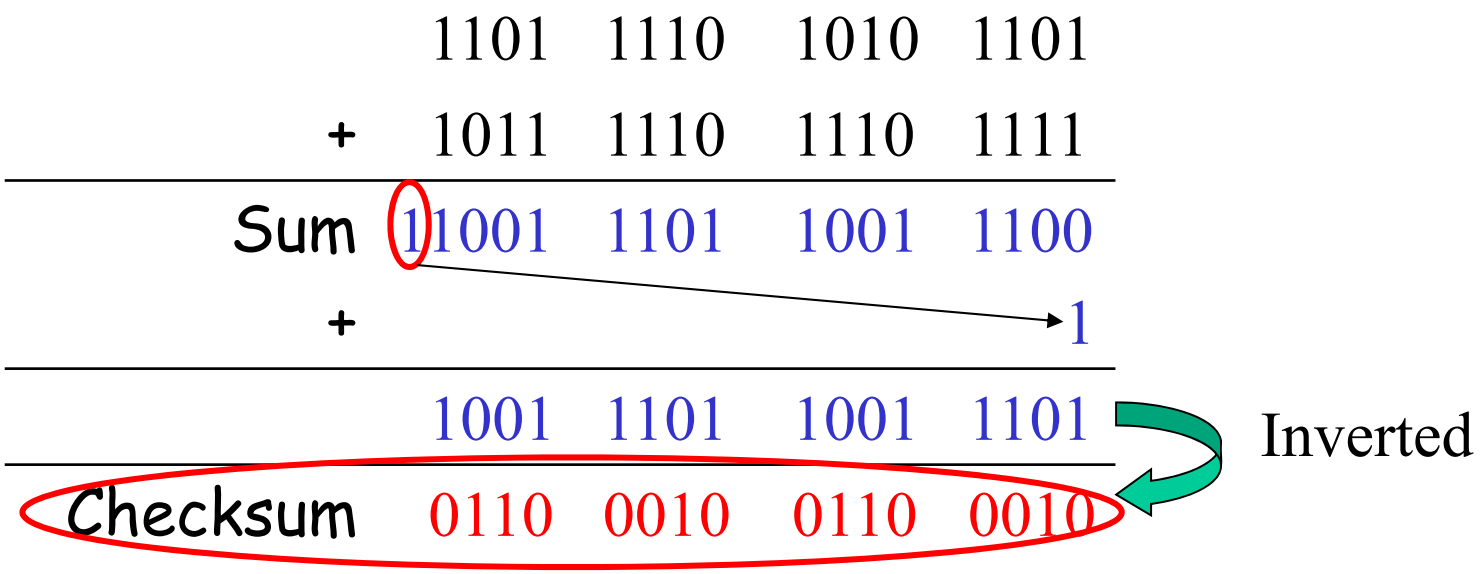
Algorithm 8.1 The 16-bit checksum algorithm used in the Internet protocols.

Internet Checksum Example

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

An example of UDP checksum

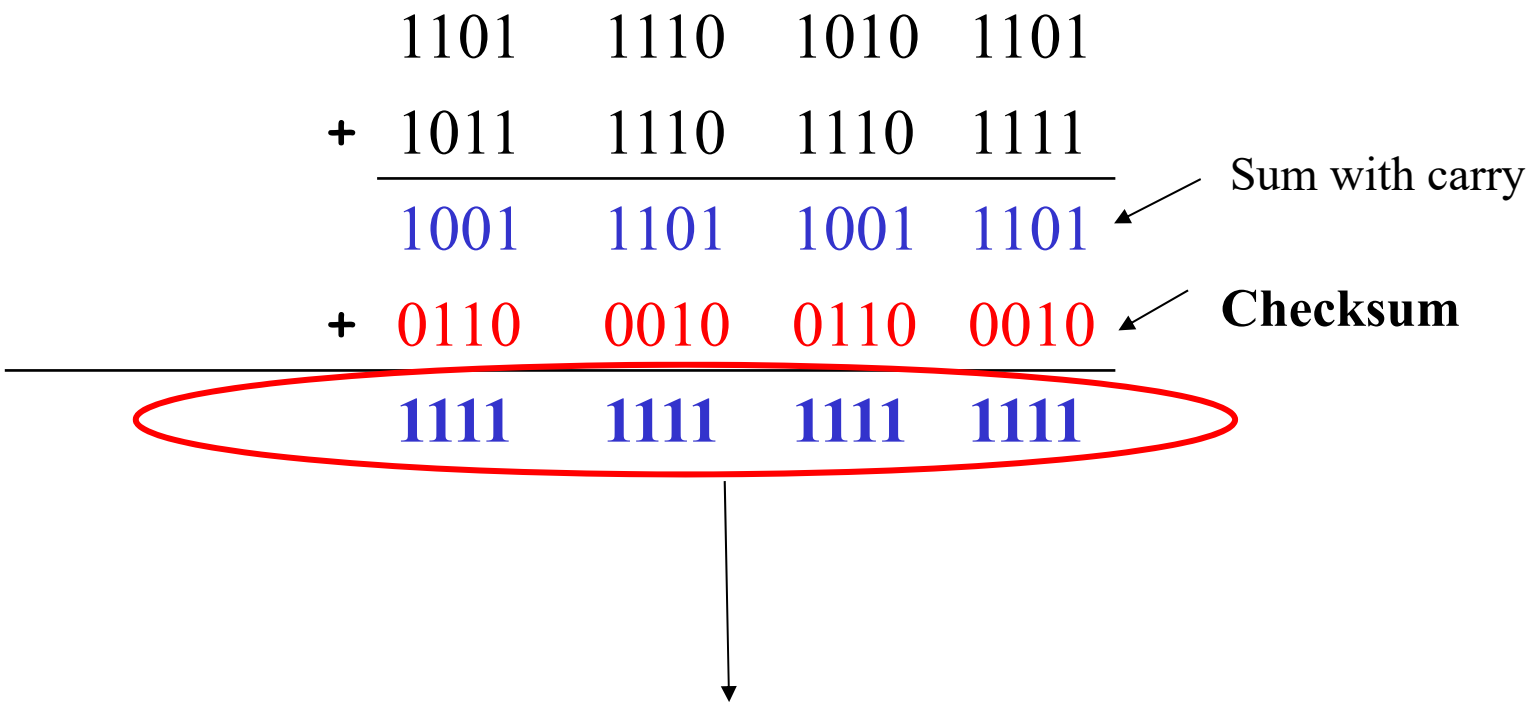
- Assume you have two messages to send:
  - MSG1: DEAD (1101 1110 1010 1101)
  - MSG2: BEEF (1011 1110 1110 1111)
- Checksum value?



Senders put 0x6262 into checksum field and send it to IP datagram

An example of UDP checksum

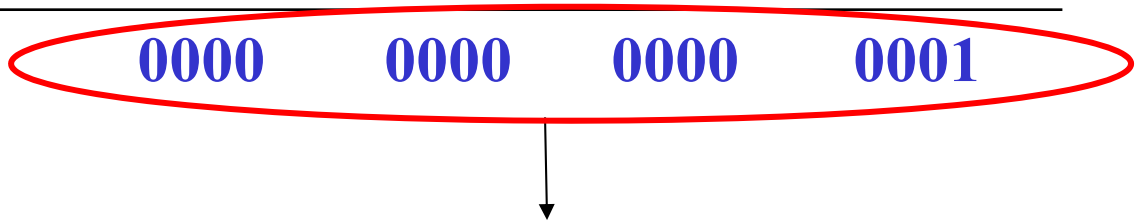
- Check on receiver side



An example of UDP checksum

- What if 1 bit of only one MSG was changed ?
  - Can we detect error bits ?

DEAF(DEAD)	1101	1110	1010	1111
+ BEEF	1011	1110	1110	1111
Sum with carry	1001	1101	1001	1111
+ Checksum	0110	0010	0110	0010
	0000	**0000**	**0000**	**0001**

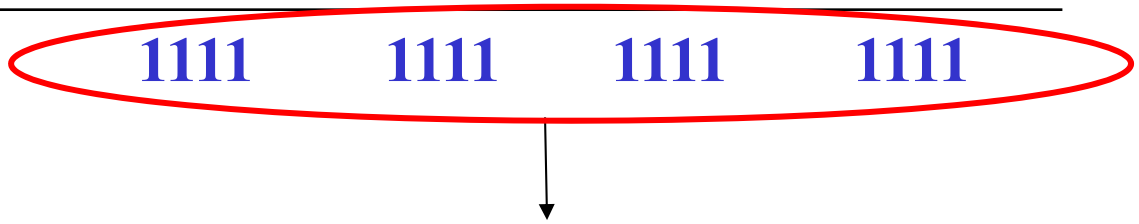


Error !!!

An example of UDP checksum

- What if 1 bit of each MSG were changed ?
  - Can we detect error bits ?

DEAF(DEAD)	1101	1110	1010	1111
+ BEED(BEEF)	1011	1110	1110	1101
Sum with carry	1001	1101	1001	1101
+ Checksum	0110	0010	0110	0010
	1111	**1111**	**1111**	**1111**



No errors !
Oops !!!

8.13 Cyclic Redundancy Codes (CRC)

- A form of channel coding known as a Cyclic Redundancy Code (CRC) is used in high-speed data networks
- Key properties of CRC are summarized below

Arbitrary Length Message	As with a checksum, the size of a dataword is not fixed, which means a CRC can be applied to an arbitrary length message
Excellent Error Detection	Because the value computed depends on the sequence of bits in a message, a CRC provides excellent error detection capability
Fast Hardware Implementation	Despite its sophisticated mathematical basis, a CRC computation can be carried out extremely fast by hardware

Figure 8.10 The three key aspects of a CRC that make it important in data networking.

8.13 Cyclic Redundancy Codes (CRC)

- Term cyclic is derived from a property of the codewords:
  - A circular shift of the bits of any codeword produces another one
- Figure 8.11 (below) show a (7, 4) CRC by Hamming

Dataword	Codeword
0000	0000 000
0001	0001 011
0010	0010 110
0011	0011 101
0100	0100 111
0101	0101 100
0110	0110 001
0111	0111 010

Dataword	Codeword
1000	1000 101
1001	1001 110
1010	1010 011
1011	1011 000
1100	1100 010
1101	1101 001
1110	1110 100
1111	1111 111

8.13 Cyclic Redundancy Codes (CRC)

- CRC codes have been studied extensively
  - A variety of mathematical explanations and computational techniques have been produced
  - The descriptions seem so disparate that it is difficult to understand how they can all refer to the same concept

Principal views include:

- Mathematicians
  - explain a CRC computation as the remainder from a division of two polynomials with binary coefficients
    - one representing the message and another representing a fixed divisor
- Theoretical Computer Scientists
  - explain a CRC computation as the remainder from a division of two binary numbers
    - one representing the message and the other representing a fixed divisor

8.13 Cyclic Redundancy Codes (CRC)

- Cryptographers
  - explain a CRC computation as a mathematical operation in a Galois field of order 2, written GF(2)
- Computer programmers
  - explain a CRC computation as an algorithm that iterates through a message
  - and uses table lookup to obtain an additive value for each step
- Hardware architects
  - explain a CRC computation as a small hardware pipeline unit that takes as input a sequence of bits from a message
  - and produces a CRC without using division or iteration

8.13 Cyclic Redundancy Codes (CRC)

- As an example of the views above
  - consider the division of binary numbers under the assumption of no carries
- Figure 8.12 illustrates the division of 1010
  - which represents a message, by a constant chosen for a specific CRC, 1011

Evariste Galois

William Wesley Peterson

8.13 Cyclic Redundancy Codes (CRC)

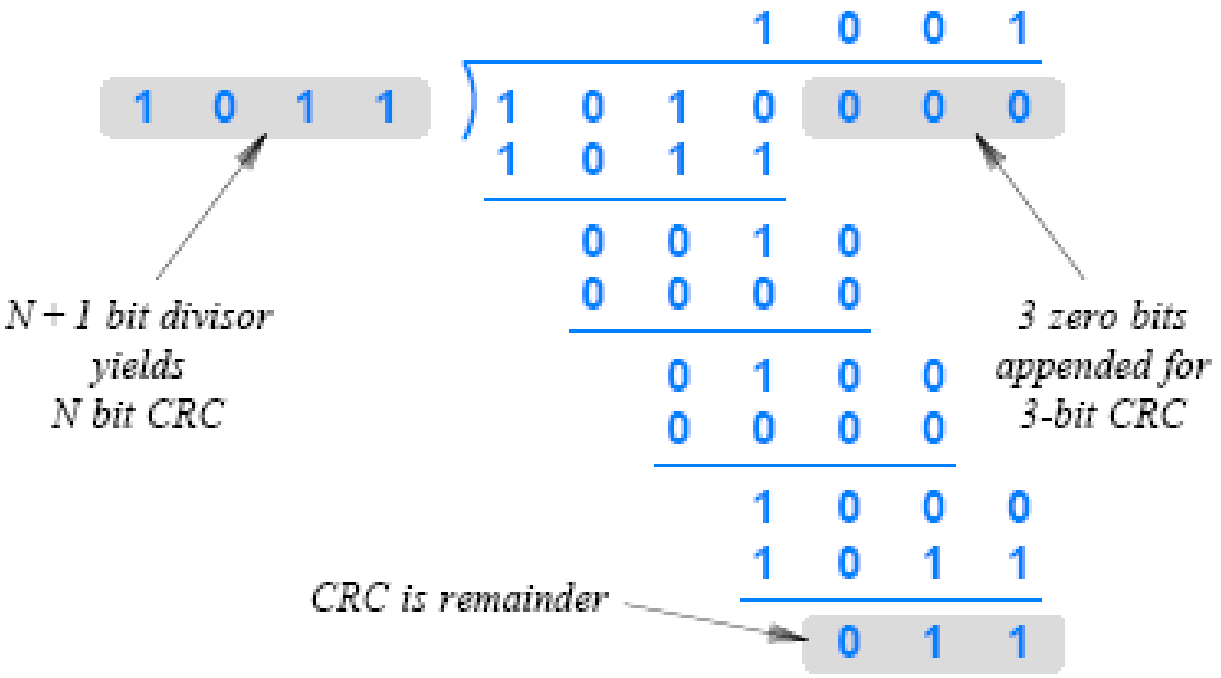


Figure 8.12 Illustration of a CRC computation viewed as the remainder of a binary division with no carries.

8.13 Cyclic Redundancy Codes (CRC)

- How can mathematicians view the above process as a polynomial division?
  - think of each bit in a binary number as the coefficient of a term in a polynomial
- For example, we can think of the divisor in Figure 8.12, 1011, as coefficients in the following polynomial:

$$1 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0 = x^3 + x + 1$$

  - Similarly, the dividend in Figure 8.12, 1010000, represents the polynomial:

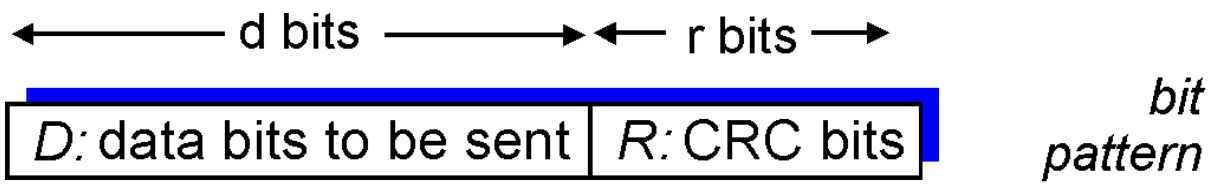
$$x^6 + x^4$$

8.13 Cyclic Redundancy Codes (CRC)

- Term generator polynomial to describe a polynomial that corresponds to a divisor
  - The selection of a generator polynomial is key to creating a CRC with good error detection properties
- An ideal polynomial is irreducible (i.e., can only be divided evenly by itself and 1)
- A polynomial with more than one non-zero coefficient can detect all single-bit errors

Checksumming: Cyclic Redundancy Check

- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (ATM, HDCL)



$D * 2^r$ XOR R *mathematical formula*

CRC Example

Want:

$D \cdot 2^r$ XOR R = nG

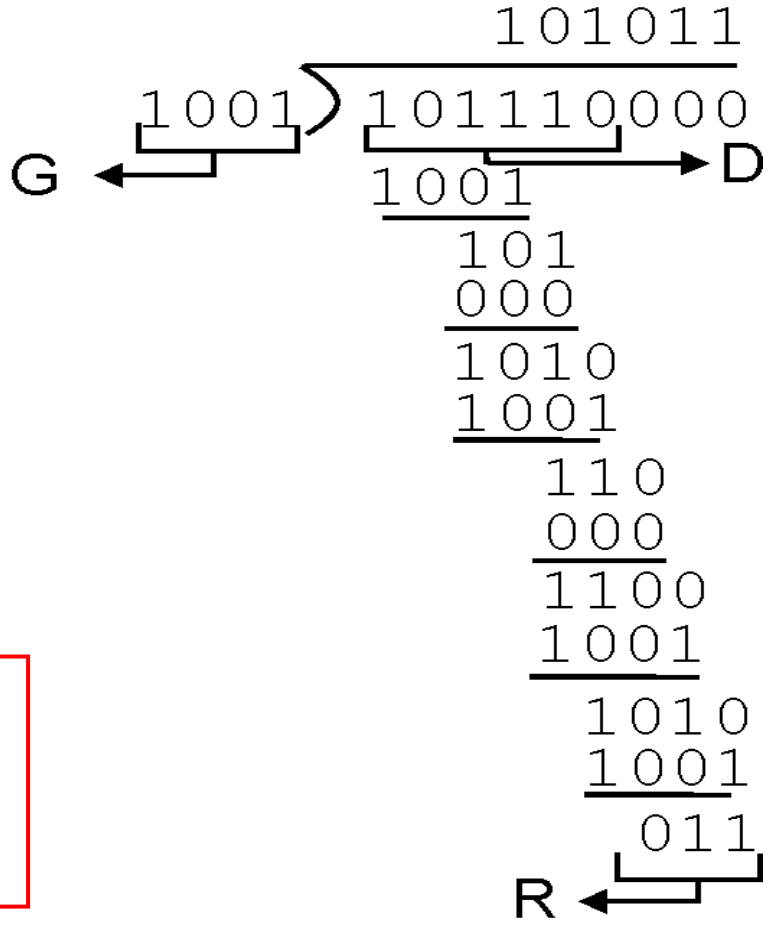
equivalently:

$D \cdot 2^r$ = nG XOR R

equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$



CRC Example (transmit)

Frame contents: 111011

Polynomial: 11101 ($x^4 + x^3 + x^2 + x^0$)

Frame with 0s: 1110110000

```
          100001
      ------------
11101 | 1110110000
        11101
        -----
         10000
         11101
         -----
          1101
```

Frame to send: 1110111101

CRC Example (receive)

Frame contents: 1110111101

Polynomial: 11101 ($x^4 + x^3 + x^2 + x^0$)

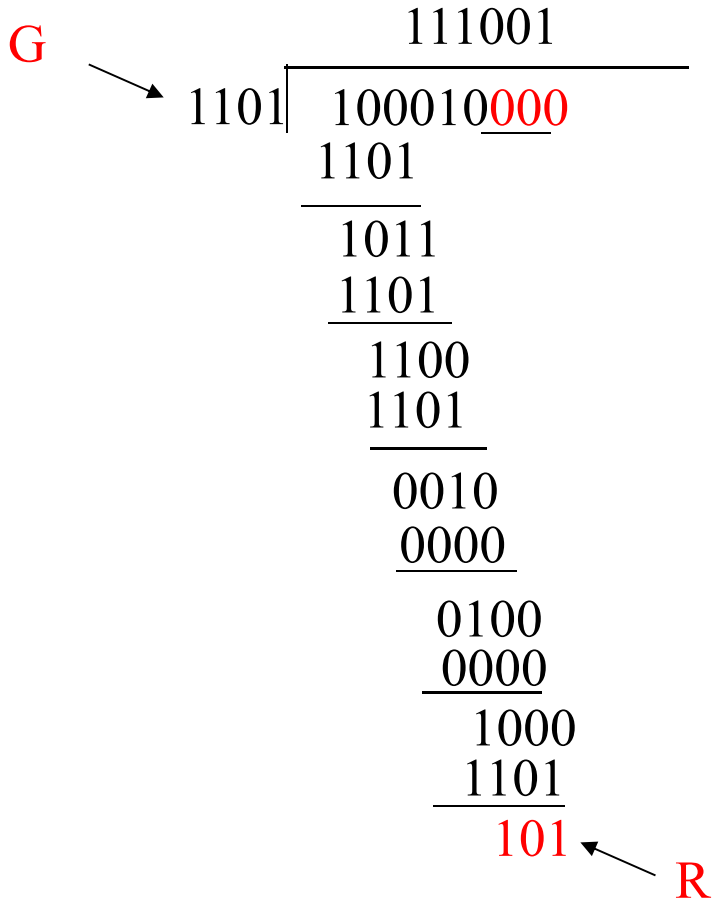
```
          100001
      ------------
11101 | 1110111101
          11101
          -----
             11101
             11101
             -----
                 0
```

Example of using CRC

- G=1101 (=$X^3 + X^2 + 1$), D=100010, S = ?

Example of using CRC

- G=1101 (=$X^3 + X^2 + 1$), D=100010, S = ?

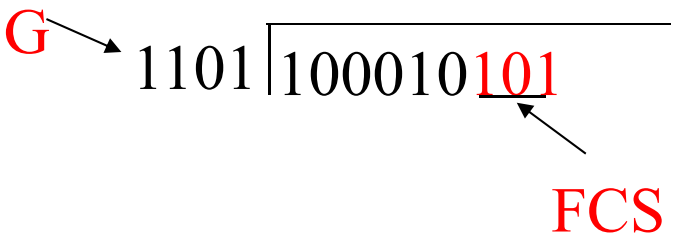


S = 100010101

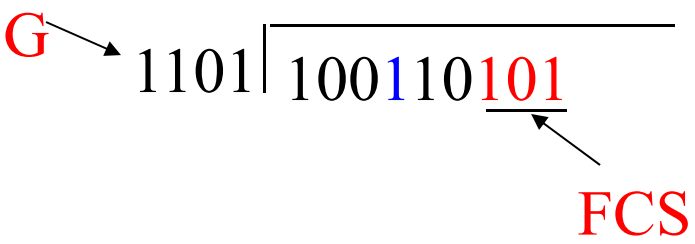
Example of using CRC

- G=1101, D=100010, FCS(Frame Check Sequence)=101
- S = 100010101

R=100010101



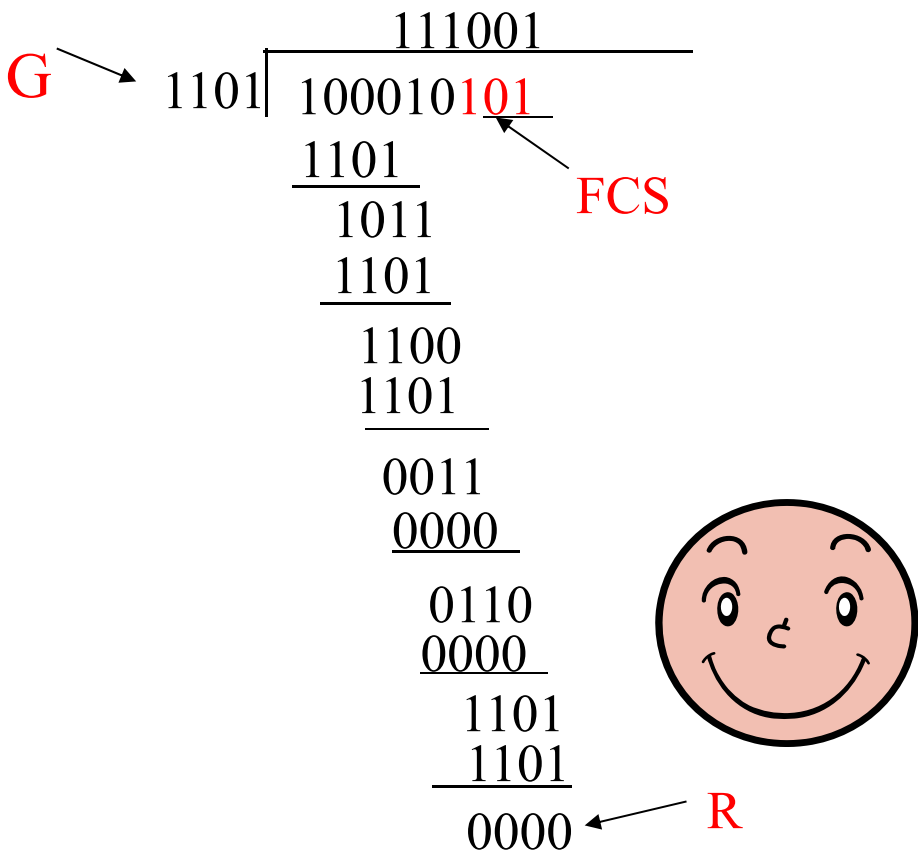
R=100110101



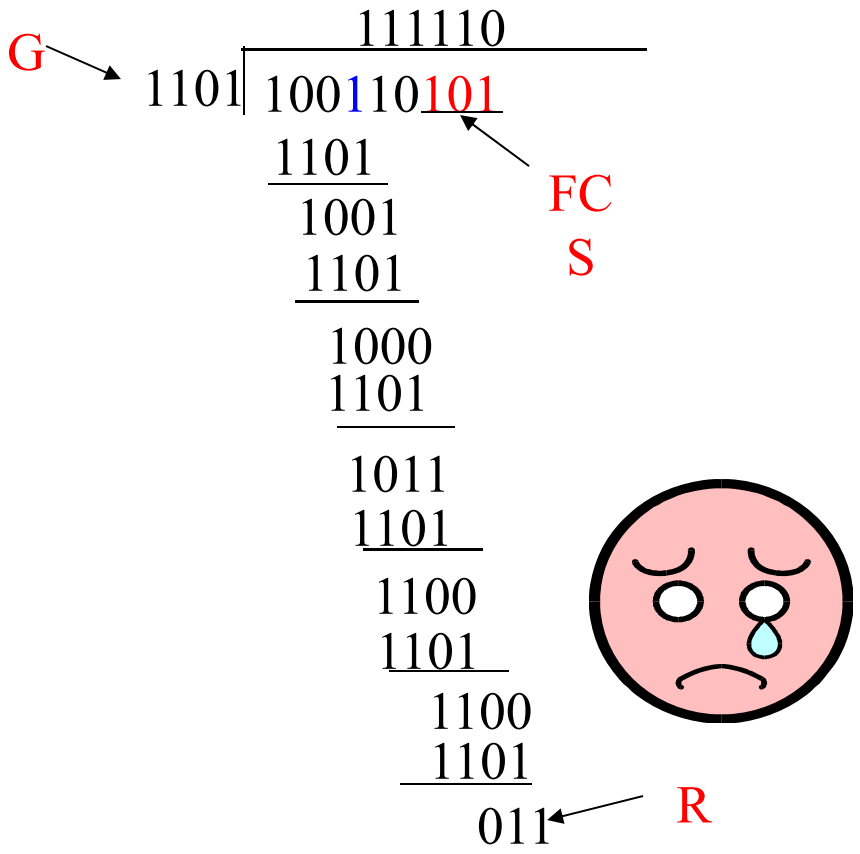
Example of using CRC

- G=1101, D=100010, FCS(Frame Check Sequence)=101
- S = 100010101

R=100010101



R=100110101



Standard Polynomial (C)

CRC	C(x)
CRC-8	$x^8+x^2+x^1+1$
CRC-10	$x^{10}+x^9+x^5+x^4+x^1+1$
CRC-12	$x^{12}+x^{11}+x^3+x^2+x^1+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	$x^{16}+x^{12}+x^5+1$
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+$ $x^8+x^7+x^5+x^4+x^2+x+1$

CRC Properties

- Detect all single-bit errors if coefficients of X^r and X^0 of $C(x)$ are one
- Detect all double-bit errors, if $C(x)$ has a factor with at least three terms
- Detect all number of odd errors, if $C(x)$ contains factor $(x+1)$
- Detect all burst of errors smaller than r+1 bits (All consecutive bit errors of r bits or fewer will be detected)

8.14 An Efficient Hardware Implementation of CRC

- CRC hardware is arranged as a shift register with exclusive or (xor) gates between some of the bits
- Figure 8.13 illustrates the hardware needed for the 3-bit CRC computation from Figure 8.12

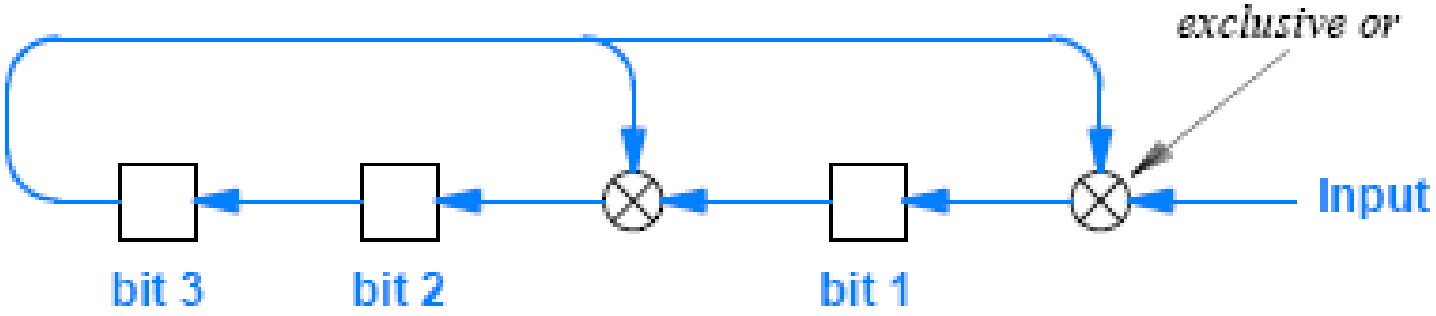


Figure 8.13 A hardware unit to compute a 3-bit CRC for $x^3 + x^1 + 1$.

8.15 Automatic Repeat reQuest (ARQ) Mechanisms

- Whenever one side sends a message to another, the other side sends a short acknowledgement (ACK) message back
  - For example, if A sends a message to B, B sends an ACK back to A
  - Once it receives an ACK, A knows that the message arrived correctly
  - If no ACK is received after T time units, A assumes the message was lost and retransmits a copy
- ARQ is especially useful in cases of dealing with detecting errors
  - but not in cases for error correction
  - many computer networks use a CRC to detect transmission errors
- An ARQ scheme can be added to guarantee delivery if a transmission error occurs
  - the receiver discards the message if an error occurs
  - and the sender retransmits another copy