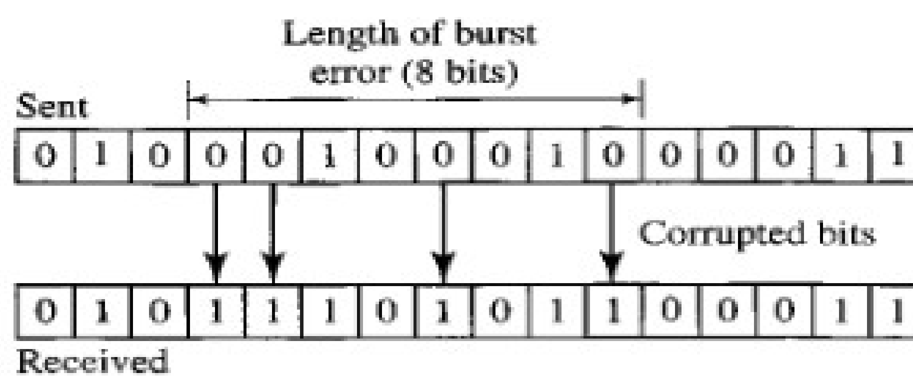
Chapter 10

Error Detection and Correction

- Data can be corrupted during transmission
- Types of Errors
  - Single-bit errors – only 1 bit is changed in the data unit
  - Burst errors – two or more bits have changed, length is measured from the first corrupted bit to the last corrupted bit,
    - Affected bits depend on data rate and noise duration



Error Detection and Correction

- Redundancy
  - To be able to detect or correct errors, extra bits are needed with data
  - Redundant bits are added by the sender and removed by the receiver
- Detection vs. Correction
  - Detection – did errors occur?
  - Correction – need to know the exact number of corrupted bits and exact location

Error Detection and Correction

- Foward Error Correction
  - Receiver tries to guess the message by using redundant bits
- Retransmission
  - Receiver detects an error and asks the sender to resend the message
- Coding
  - Creates a relationship between redundant bits and actual data bits; block coding and convolution coding

Modular Arithmetic

- We use a limited range of integers
- Modulus N – upper limit, use the integers from 0 to N-1
- Ex. Clock system is modulo-12
- If a number is greater than N, it is divided by N and the remainder is the result
- If a number is negative, as many Ns are added to make it positive

Modular Arithmetic

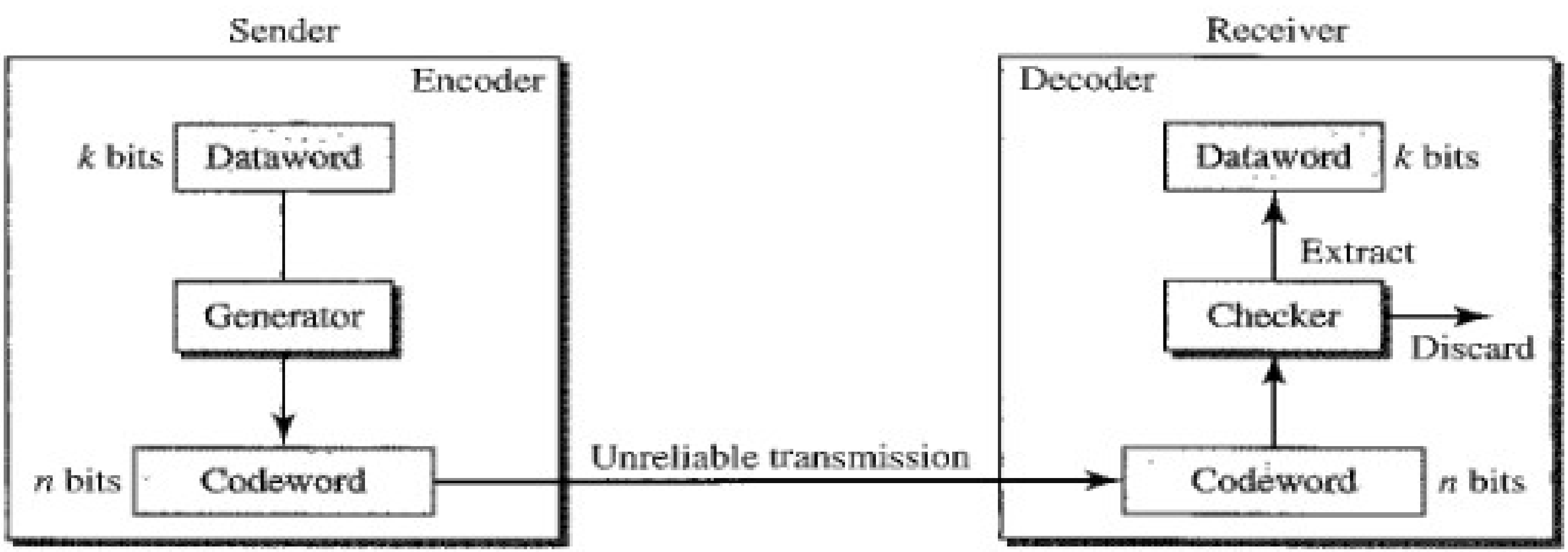
- Modulo-2
  - Adding
    - 0+0=0, 0+1=1, 1+0=1, 1+1=0
  - Subtracting
    - 0-0=0, 0-1=1, 1-0=1, 1-1=0
  - Adding and subtracting yields the same results
  - We can use XOR for both addition and subtraction

Block Coding

- Divide message into blocks, each of k bits, called datawords
- We add r redundant bits to each block to make the length $n = k + r$, the n-bit blocks are called codewords
- With k bits, we can have 2^k data words, with n bits we can have 2^n codewords
- Same dataword is always encoded as the same codeword

Block Coding: Error Detection

- Conditions to detect
  - Receiver has/can find a list of valid codewords
  - Original codeword has changed to an invalid one
- Can only detect single bit errors



Block Coding: Error Detection

- Example: Sender encodes 01 as 011
- Cases:
  - Receiver gets 011 - valid
  - Receiver gets 111 - invalid
  - Receiver gets 000 – undetected error

Table A

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

Block Coding: Error Correction

- Sender encodes 01 as 01011
- Receiver gets 01001
- Codeword not in table
- Assumes that 1 bit is corrupted
- Not first, third, fourth – 2 bits in error
- Must be second!

Table B

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Hamming Distance

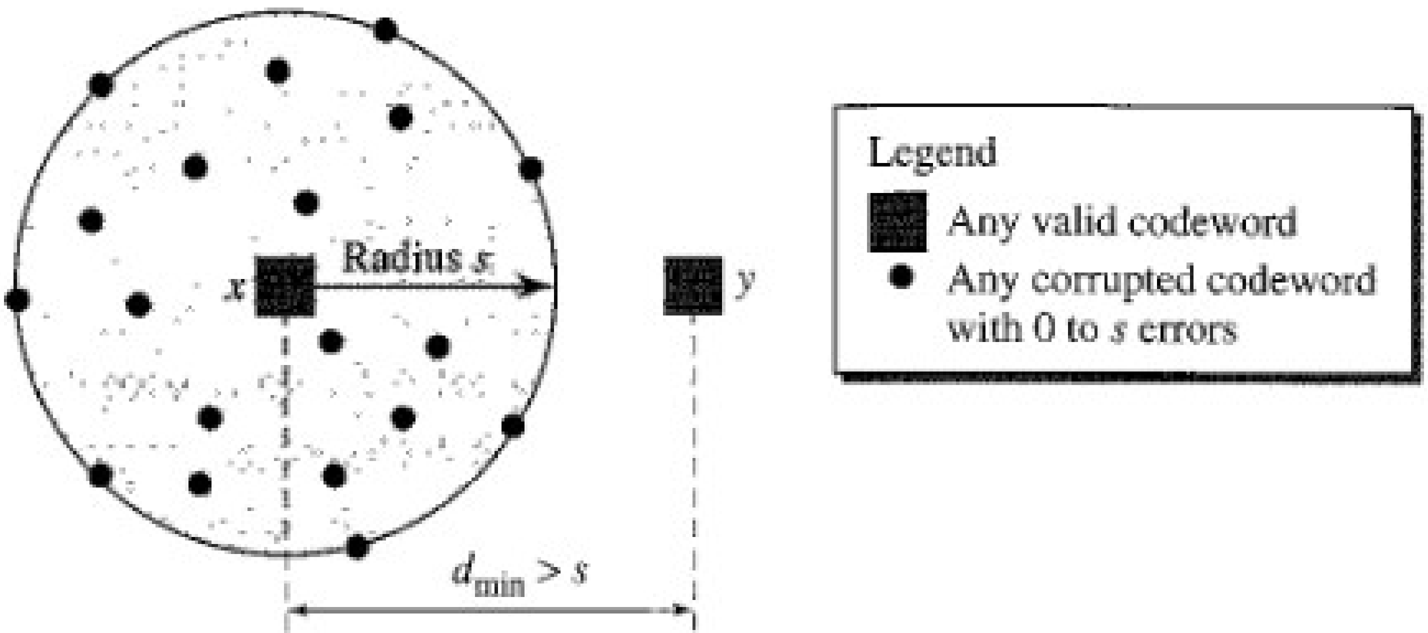
- The number of differences between the corresponding bits between two words
- d(x,y)
- Use XOR to and count the number of 1's in the result
- Ex.: d(000,011)=2, d(10101,11110)=3

Minimum Hamming Distance

- Smallest hamming distance between all possible pairs of the codewords
- d_{min}
- d(000,011)=2, d(000,101)=2, d(000,110)=2, d(011,101)=2, d(011,110)=2, d(101,110)=2
  - $d_{min} = 2$
- Any coding scheme needs to have at least three parameters: codeword size n, dataword size k, and d_{min}
- C(3,2), d_{min}=2

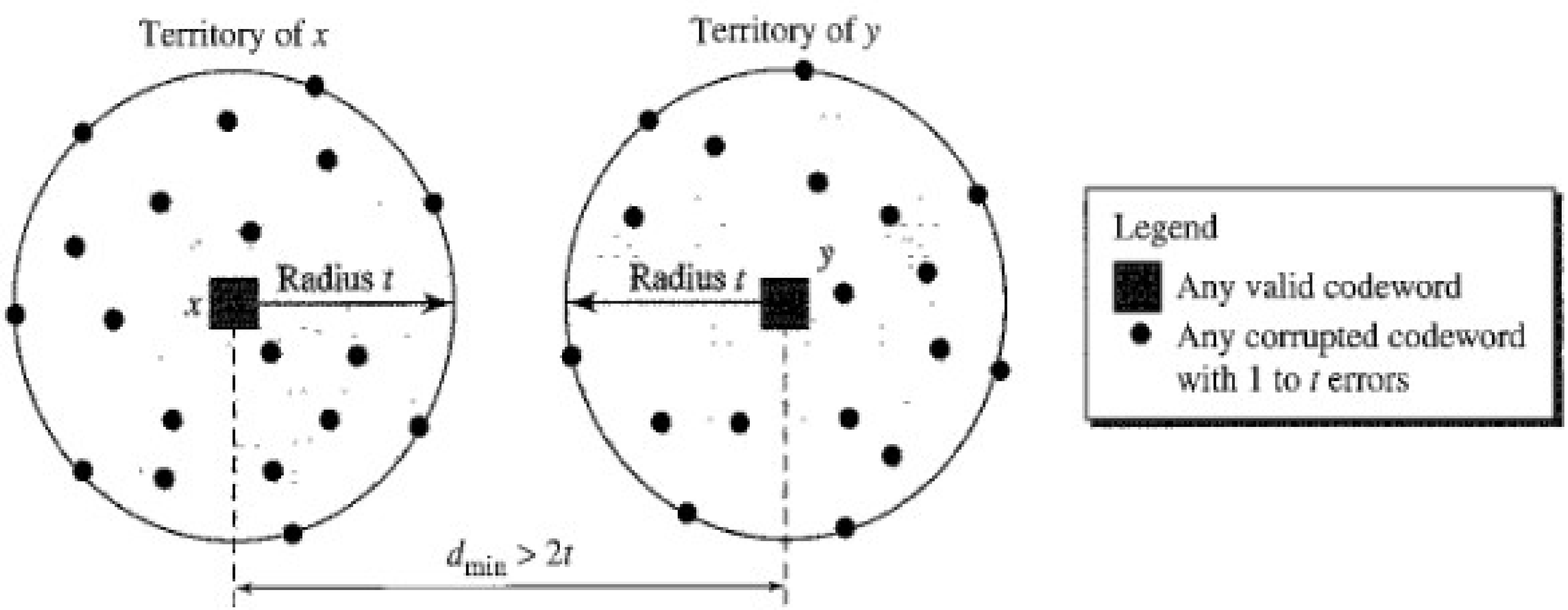
Minimum Hamming Distance

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



Minimum Hamming Distance

- To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min}=2t+1$



Linear Block Codes

- A code in which the XOR of two valid codewords creates another valid codeword
- Minimum hamming distance is the number of 1's in the nonzero valid codeword with the smallest number of 1's
- In Table A, number of 1's in the nonzero codewords are 2,2, and 2; $d_{min}=2$
- In Table B, number of 1's in the nonzero codewords are 3,3, and 4; $d_{min}=3$

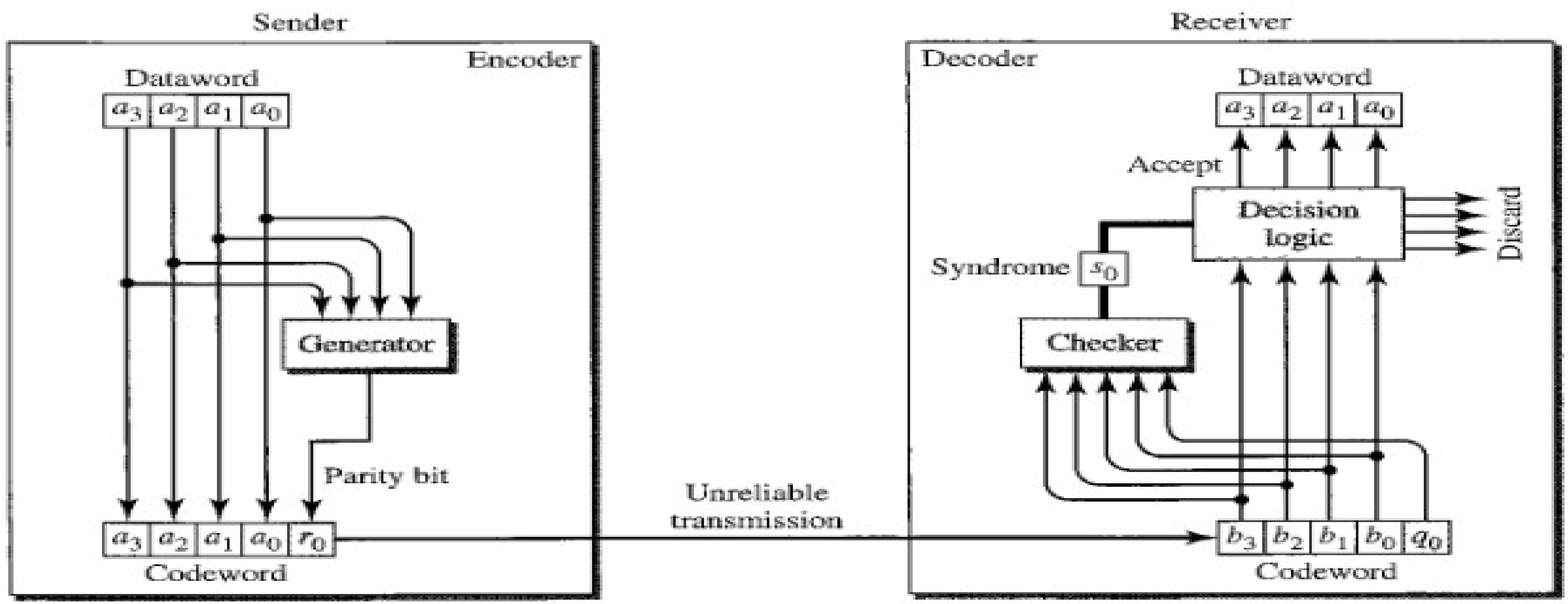
Simple Parity Check

- A single-bit error-detecting code in which $n=k+1$ with $d_{min}=2$

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

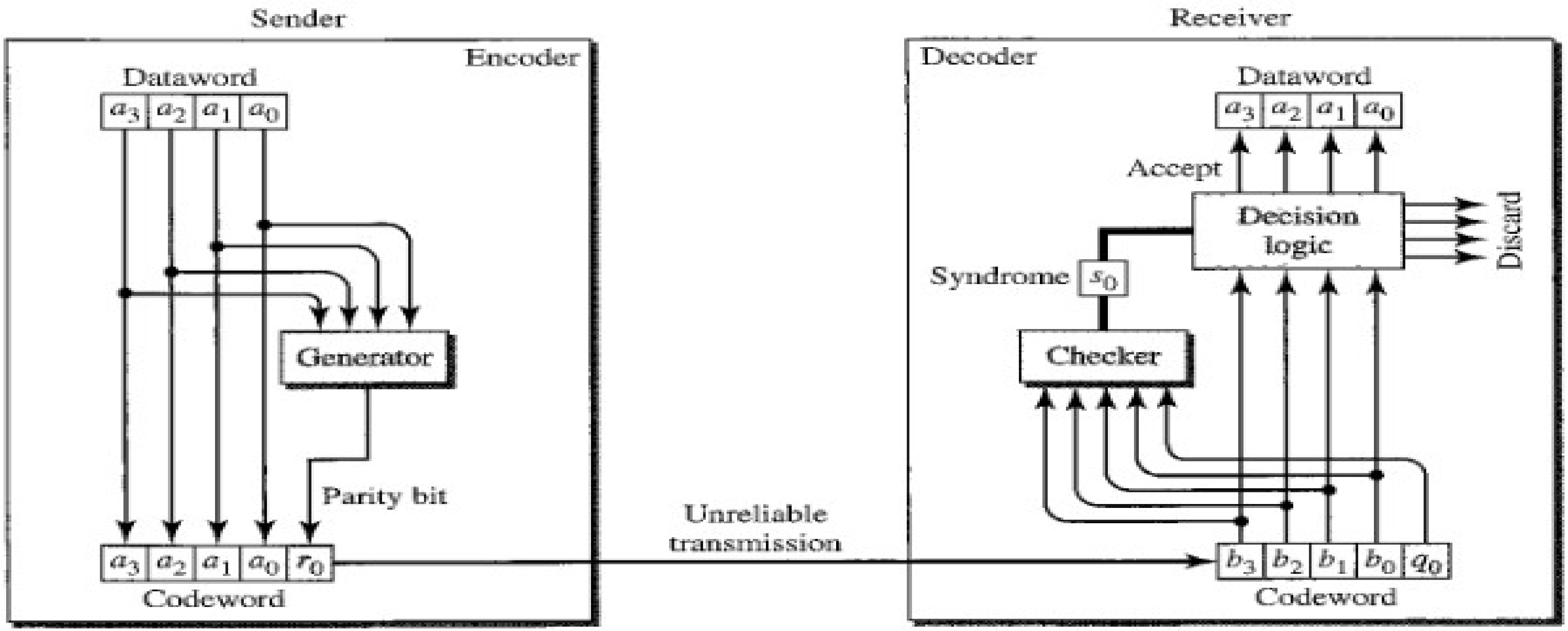
Simple Parity Check

- $r_0=a_3+a_2+a_1+a_0$ (modulo-2), number of 1's is even, result is 0;number of 1's is odd result is 1



Simple Parity Check

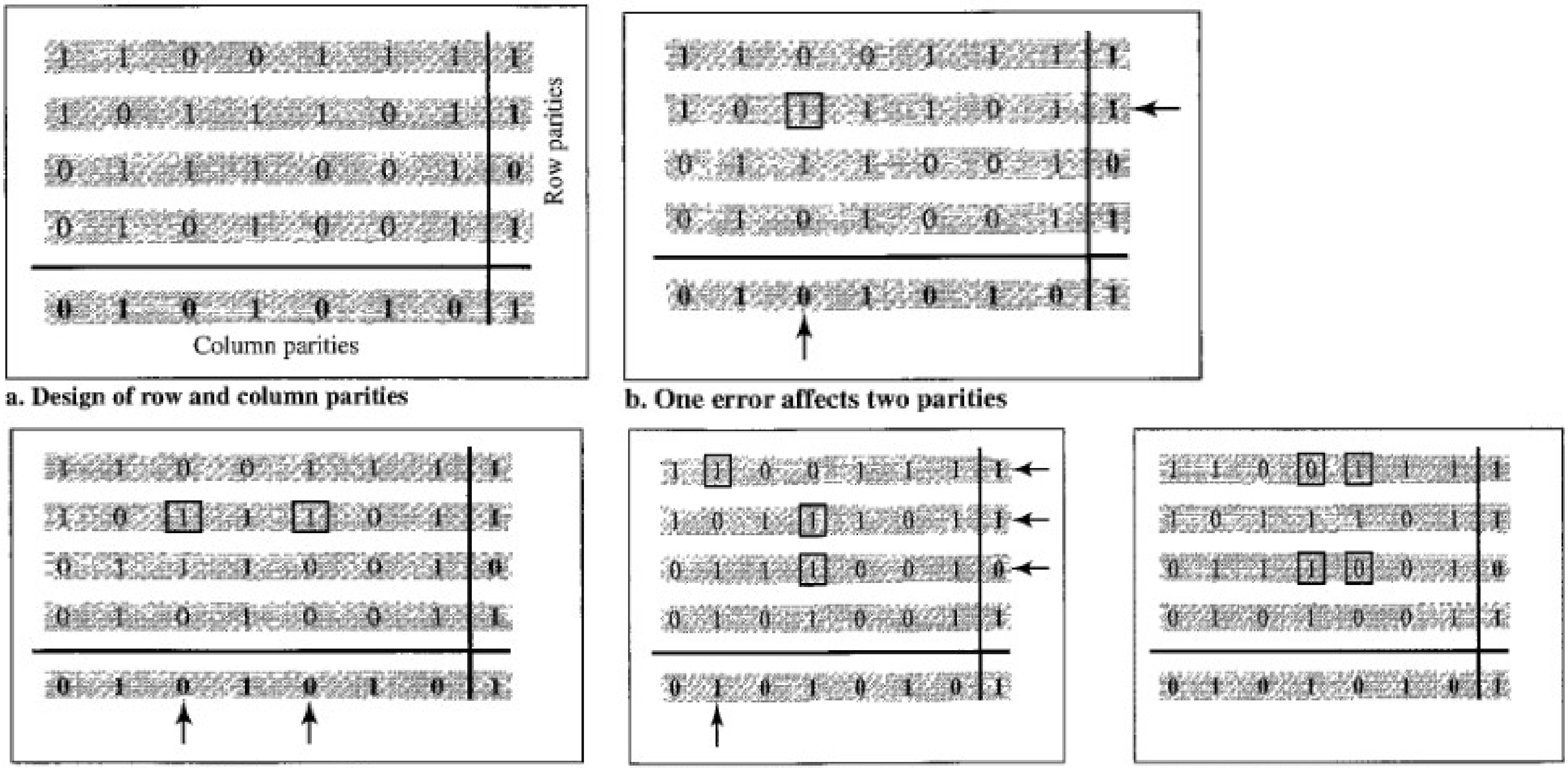
- $s_0 = b_3 + b_2 + b_1 + b_0 + q_0$, syndrome is 0 when the number of 1's in the received codeword is even; otherwise, it is 1



Simple Parity Check

- Can only detect an odd number of errors

Two-Dimensional Parity Check



a. Design of row and column parities

b. One error affects two parities

c. Two errors affect two parities

d. Three errors affect four parities

e. Four errors cannot be detected

Hamming Codes

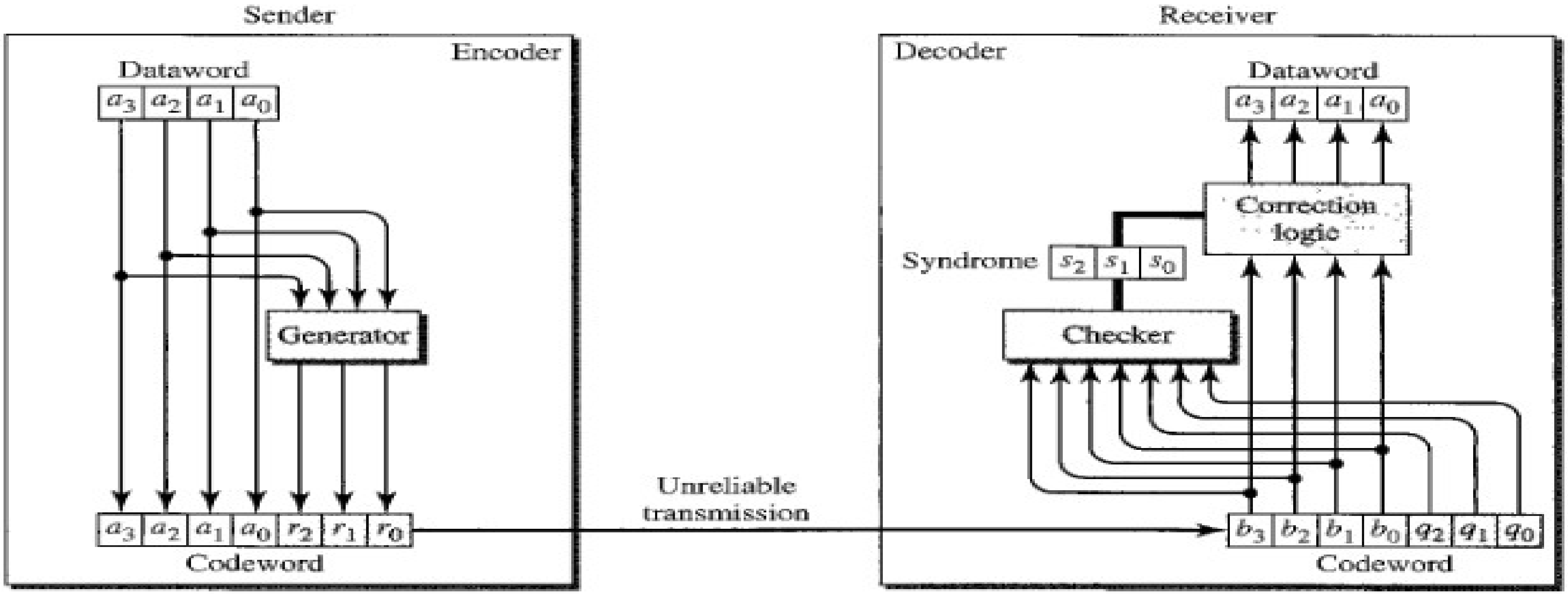
- $d_{min}=3$, can detect up to two errors or correct one single error
- We will focus on single-bit error-correcting code
- $n=2^m-1$, $k=n-m$, $m>=3$, $r=m$

Hamming Codes

- Ex. If m=3, Hamming code C(7,4), d_{min}=3

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Hamming Codes



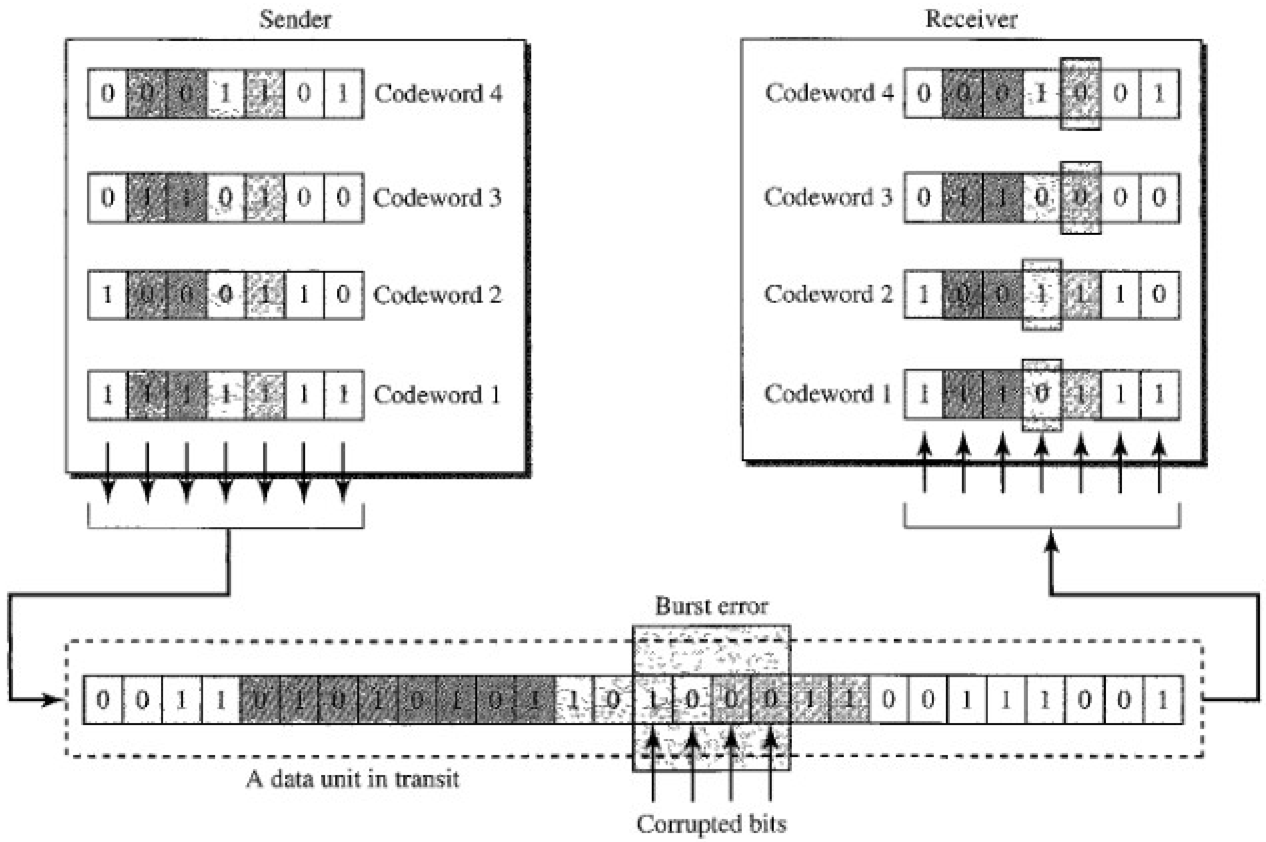
$r_0 = a_2 + a_1 + a_0$	modulo-2	$s_0 = b_2 + b_1 + b_0 + q_0$	modulo-2
$r_1 = a_3 + a_2 + a_1$	modulo-2	$s_1 = b_3 + b_2 + b_1 + q_1$	modulo-2
$r_2 = a_1 + a_0 + a_3$	modulo-2	$s_2 = b_1 + b_0 + b_3 + q_2$	modulo-2

Hamming Codes

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

- If b_2 is in error, s_0 and s_1 are the bits affected, syndrome is 011, bit b_2 must be flipped to correct the error
- Split burst errors between several codewords, one error for each codeword

Hamming Codes



Cyclic Codes

- Linear block codes in which if a codeword is cyclically shifted (rotated), the result is another codeword
- Ex. 1011000, perform a SHL, 0110001
- Ex. Cyclic Redundancy Check (CRC)

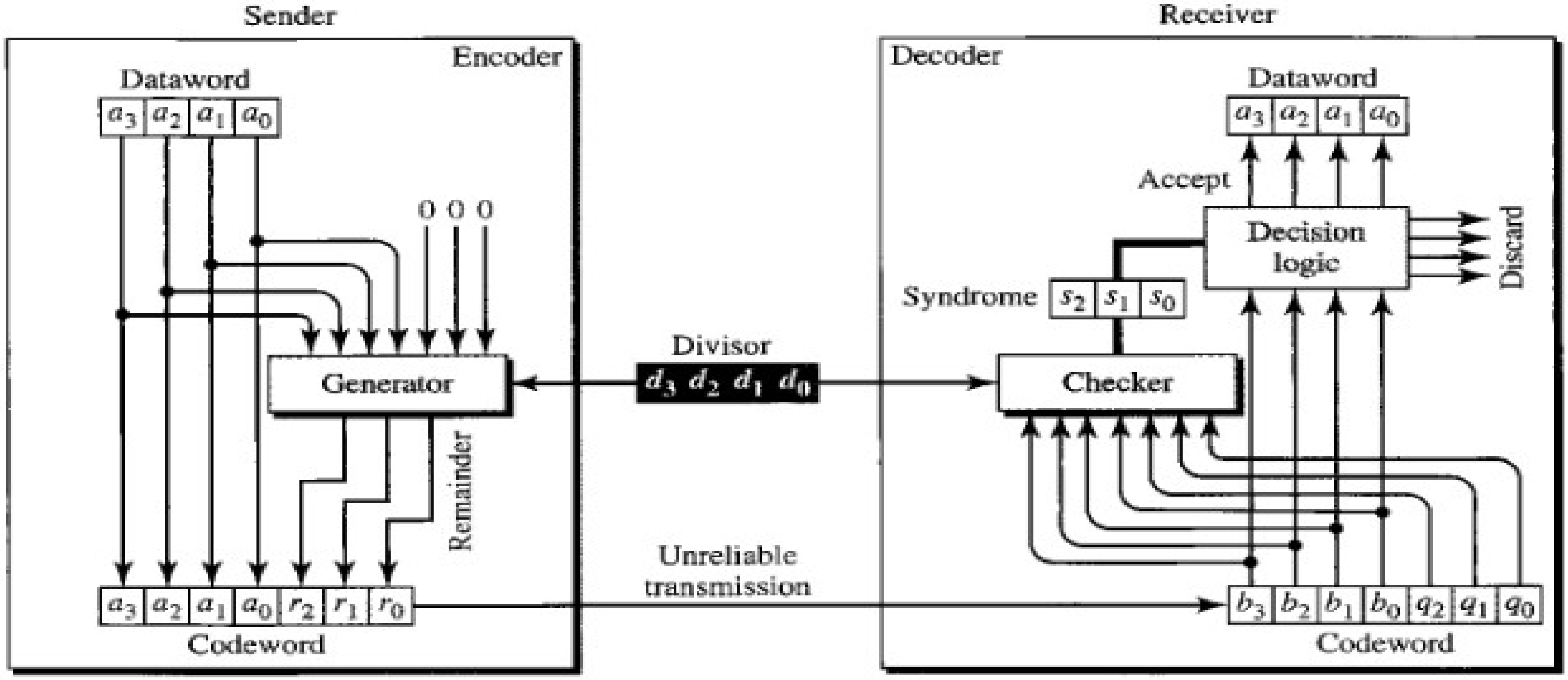
CRC

- 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

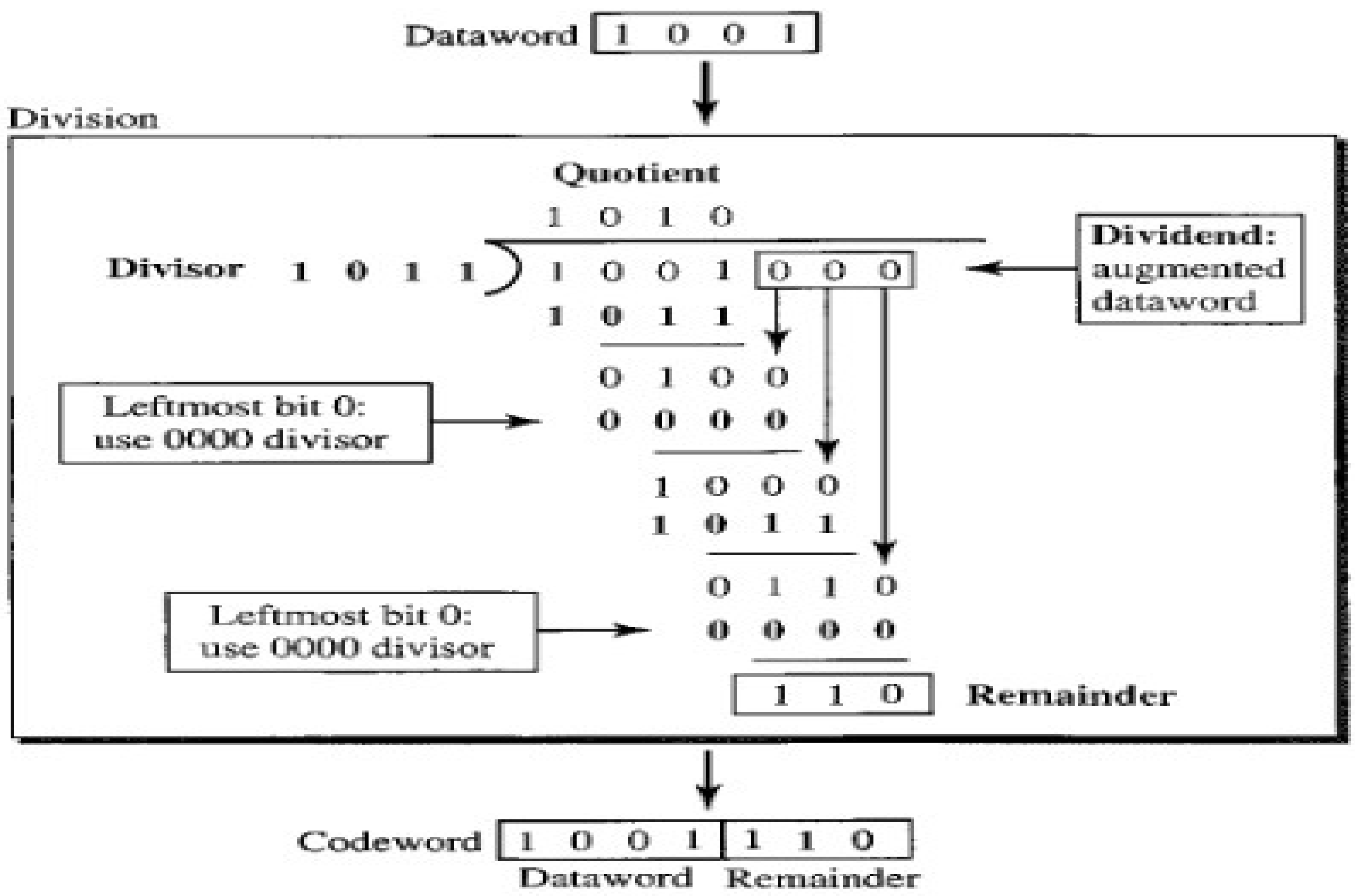
- Encoder and Decoder



CRC Encoder

- Dataword has k bits, codewords has n bits
- Size of dataword is augmented by adding n-k zeros to the righthand side
- n-bit result is fed into generator
- Generator use a divisor of size n-k+1, predefined and agreed upon
- Divisor divides augmented dataword by divisor (modulo-2 division)
- Remainder is appended to dataword to create codeword

CRC Encoder

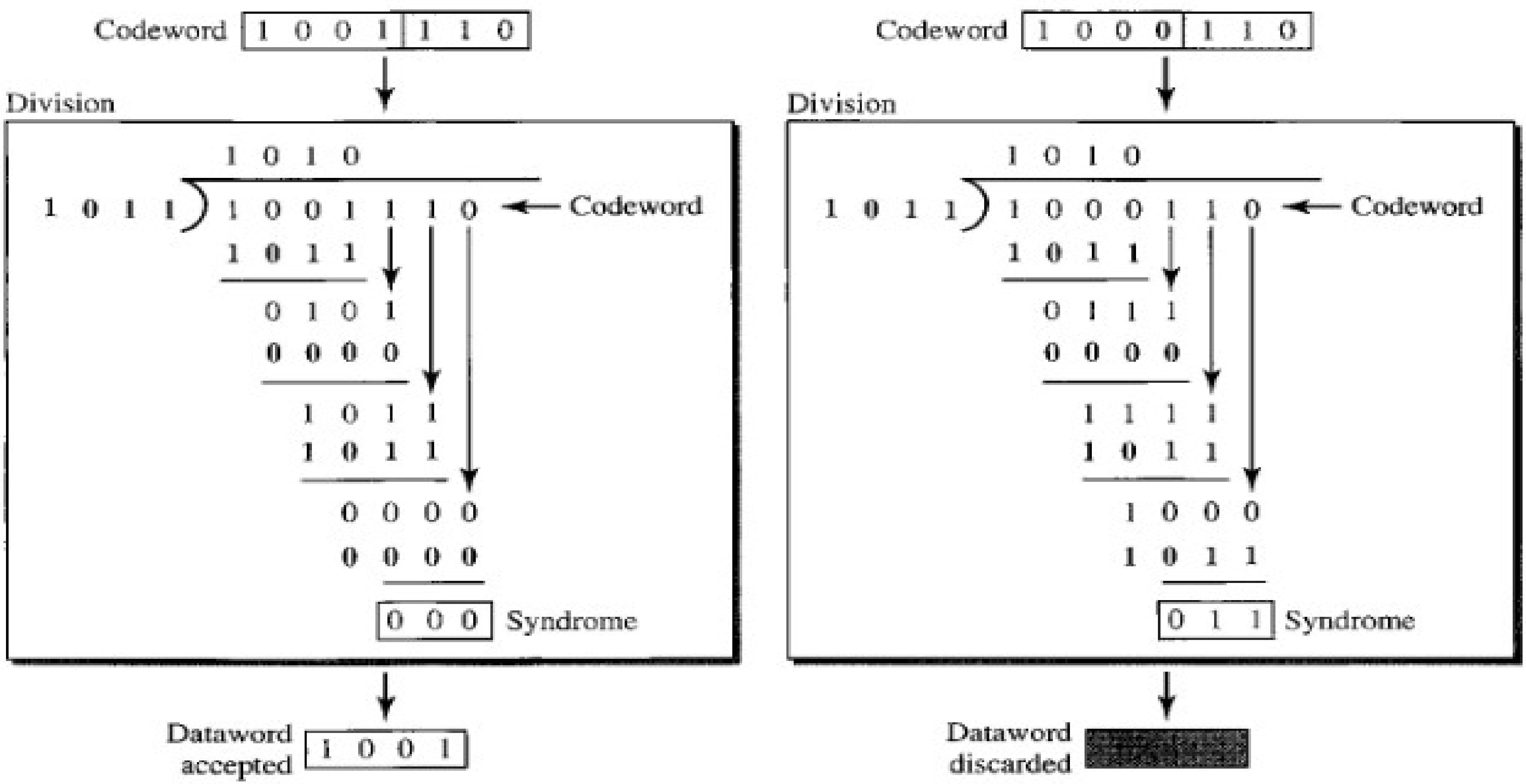


Dataword
1 0 0 1
Division
Quotient
1 0 1 0
Divisor
1 0 1 1
1 0 0 1
0 0 0
Dividend: augmented dataword
1 0 1 1
0 1 0 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 0 0 0
1 0 1 1
0 1 1 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 1 0
Remainder
Codeword
1 0 0 1
1 1 0
Dataword
Remainder

CRC Decoder

- Decoder receives a possibly corrupted codeword
- n bits fed to the checker, replica of generator
- Remainder produced by checker is a syndrome of n-k bits
- If syndrome bits are all 0's , k leftmost bits of the codeword are accepted as dataword

CRC Decoder


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

CRC

- Good performance in detecting single bit errors, double errors, odd number of errors, burst errors
- Easily implemented in hardware and software
- Hardware implementation is fast

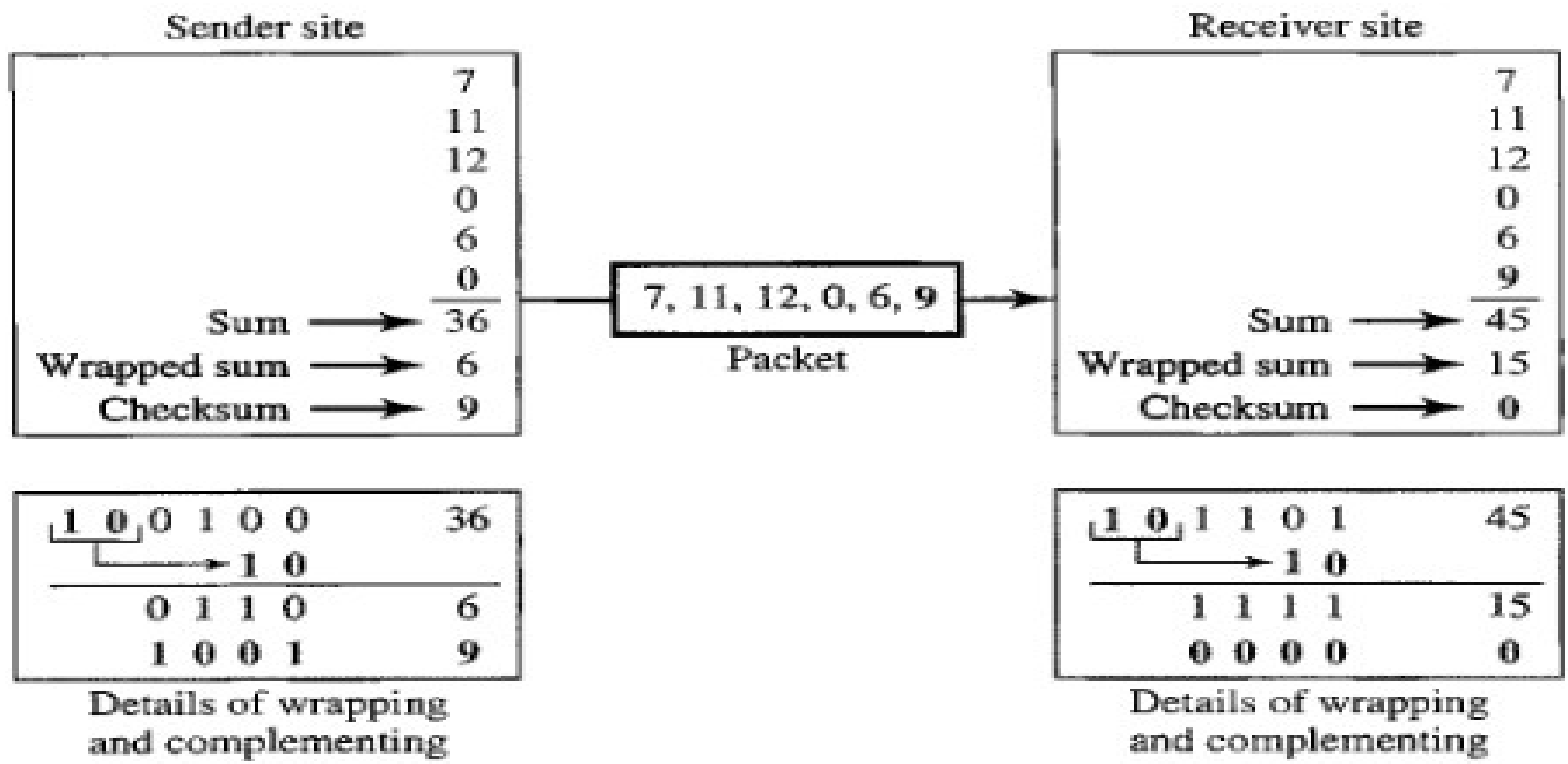
Checksum

- Suppose data is five 4-bit numbers: (7,11,12,0,6)
- We send (7,11,12,0,6, **36**), sum is 36
- Receiver adds the five number and compares result to the sum
- If we send the negative of the sum: (7,11,12,0,6, -**36**), receiver simply adds everything and a result of zero is expected

Checksum

- In the previous example, the checksum,-36, cannot be written as a four bit word
- Use one's complement arithmetic
  - Unsigned numbers between 0 to 2^n-1 using only n bits
  - If the number has more than n bits, extra leftmost bits are added to the n rightmost bits
  - Negative number can be represented by inverting all bits
- Ex. 21 using 4 bits is 6; 10101=>0101+1=0110

Checksum



Sender site
7
11
12
0
6
0
Sum
36
Wrapped sum
6
Checksum
9
7, 11, 12, 0, 6, 9
Packet
Receiver site
7
11
12
0
6
9
Sum
45
Wrapped sum
15
Checksum
0
1 0 0 1 0 0 36
1 0
0 1 1 0 6
1 0 0 1 9
Details of wrapping and complementing
1 0 1 1 0 1 45
1 0
1 1 1 1 15
0 0 0 0 0
Details of wrapping and complementing

Checksum

- Checksum in the Internet, 64-bit, msg divided into 16-bit words

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
0	0	0	0	Checksum (initial)
8	F	C	6	Sum (partial)
			1	
8	F	C	7	Sum
7	0	3	8	Checksum (to send)

a. Checksum at the sender site

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
7	0	3	8	Checksum (received)
F	F	F	E	Sum (partial)
			1	
F	F	F	F	Sum
0	0	0	0	Checksum (new)

b. Checksum at the receiver site

Enjoy! :)