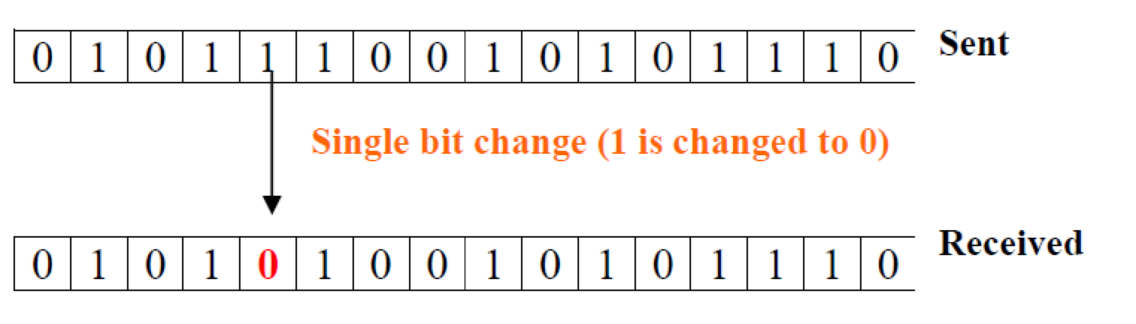
Error Handling

Dr. Amrit Pal

Error

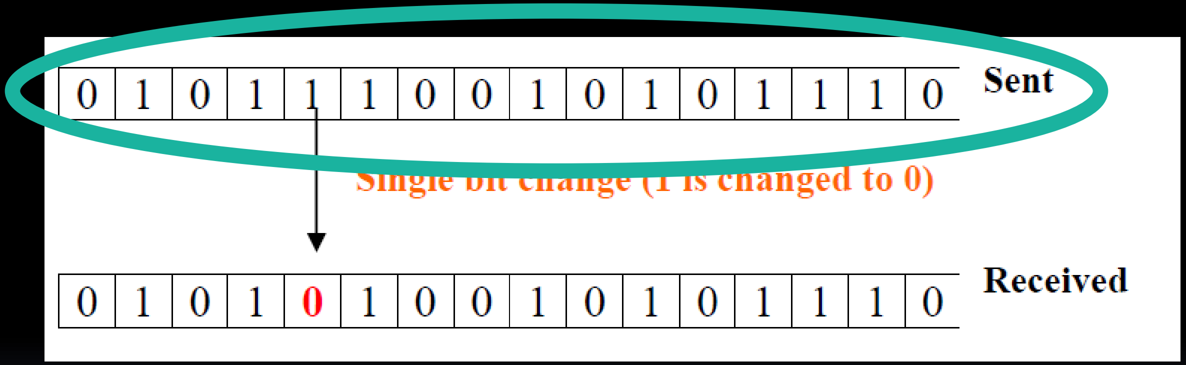
- For most applications, **a system must guarantee** that the data received are identical to the data transmitted

Error



0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0
Sent
Single bit change (1 is changed to 0)
0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0
Received

Error



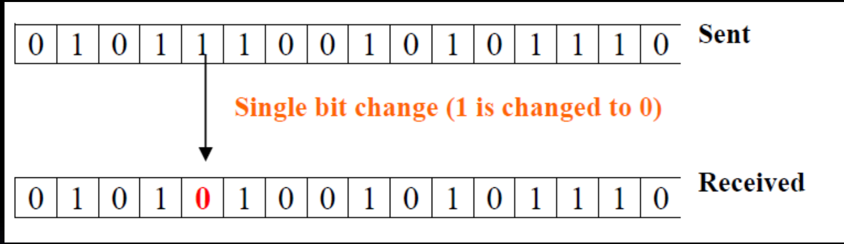
0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0
Sent
Single bit change (1 is changed to 0)
0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0
Received

Question

- How one can know whether the received data is correct or not?

Question

- How one can know whether the received data is correct or not?



Error Tolerance

Dr. Amrit Pal

Understand Error Types

- Relation between noise and data transmission

Understand Error Types

- Relation between noise and data transmission

Understand Error Types

- Relation between noise and data transmission
- Effect of the noise and duration of the noise

Understand Error Types

- Effect of the noise and duration of the noise
  - Single bit error
  - Multiple bit error
    - Burst Error

Understand Error Types

- Takeaway
  - The duration of noise is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits.

Handling errors

Dr. Amrit Pal

Handling errors

- By Detecting
- By Correcting

To detect

- We need some extra information, to check whether the data is correct or not.

To detect

- We need some extra information, to check whether the data is correct or not.

- EXTRA → Redundant information

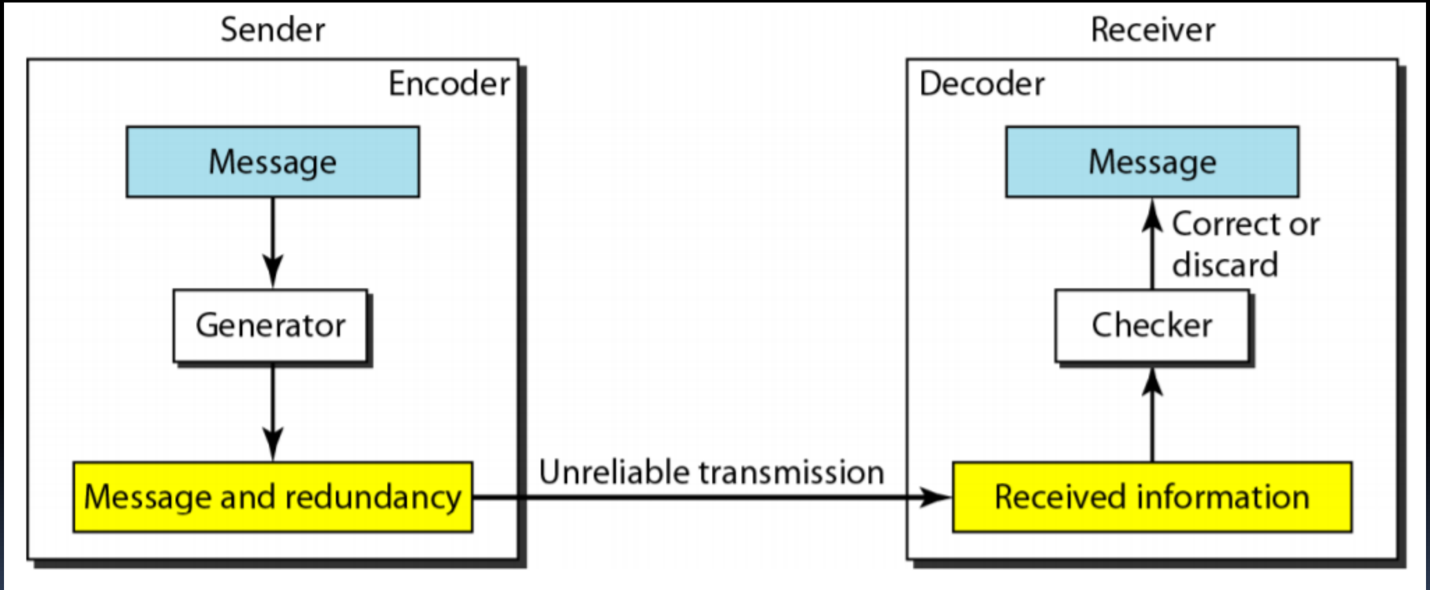
Detection Versus Correction

- Which is difficult

Detection Versus Correction

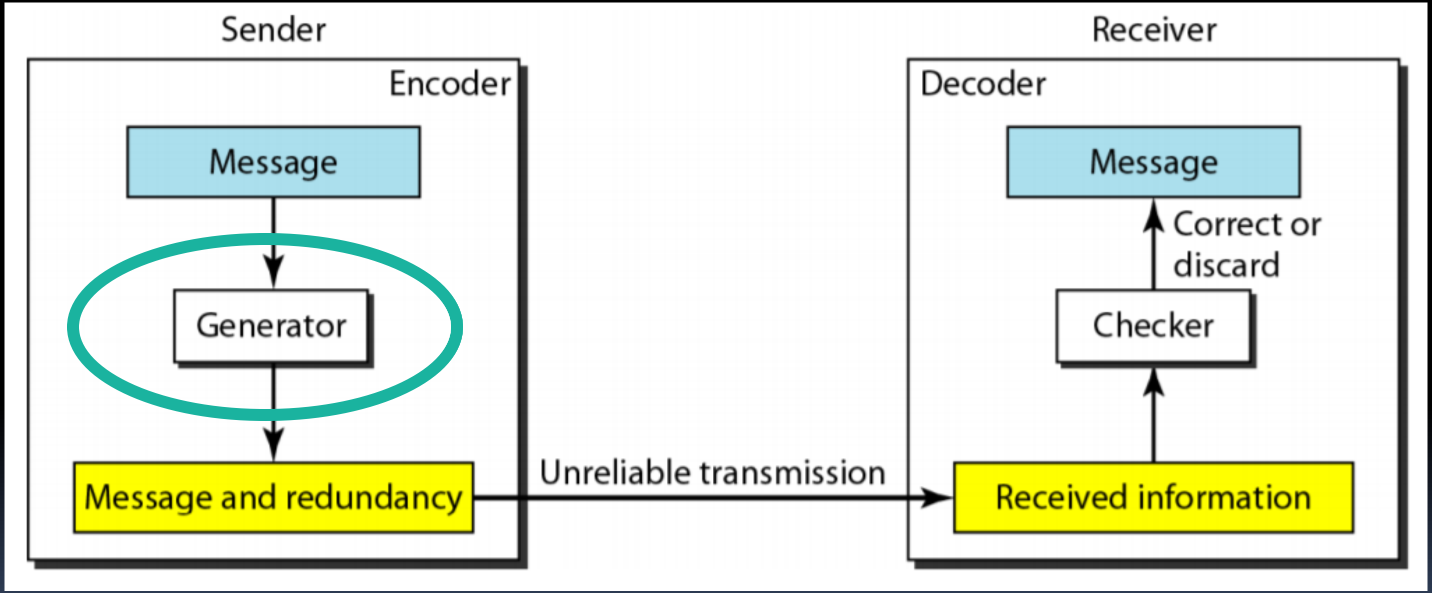
- Which is difficult
- How to correct (broad categories)
  - Forward Error Correction
  - Retransmission

Detection

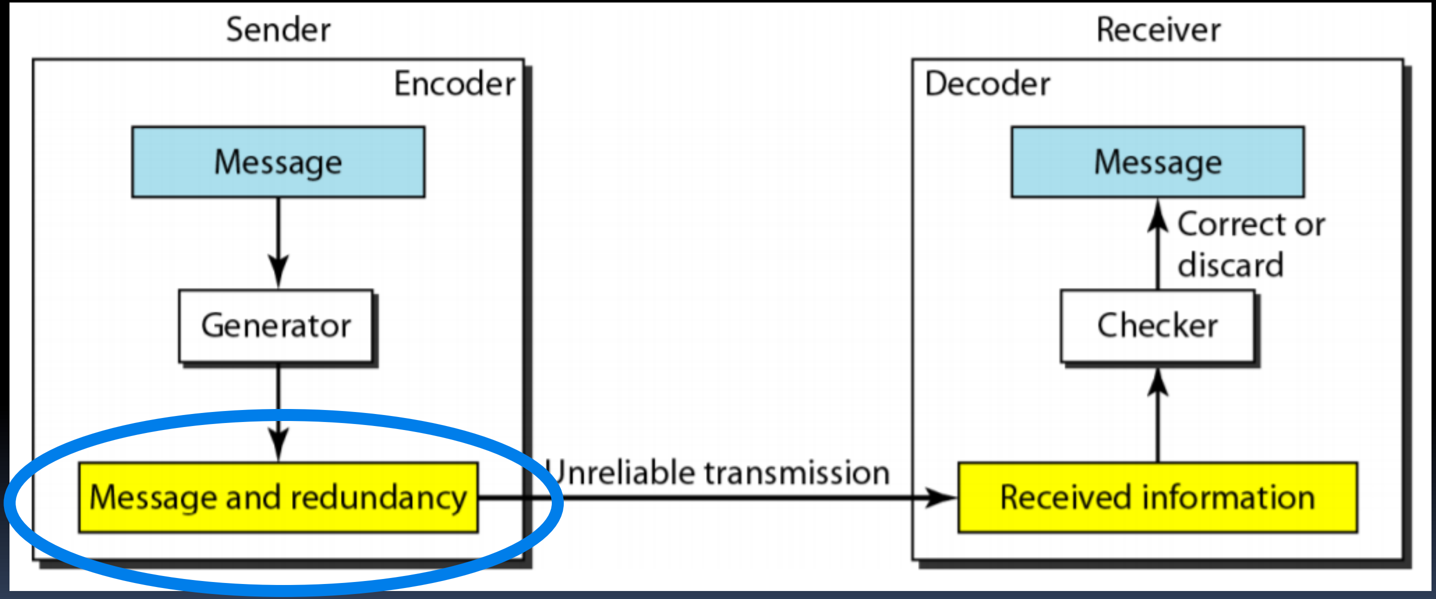


Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or discard
Checker
Received information

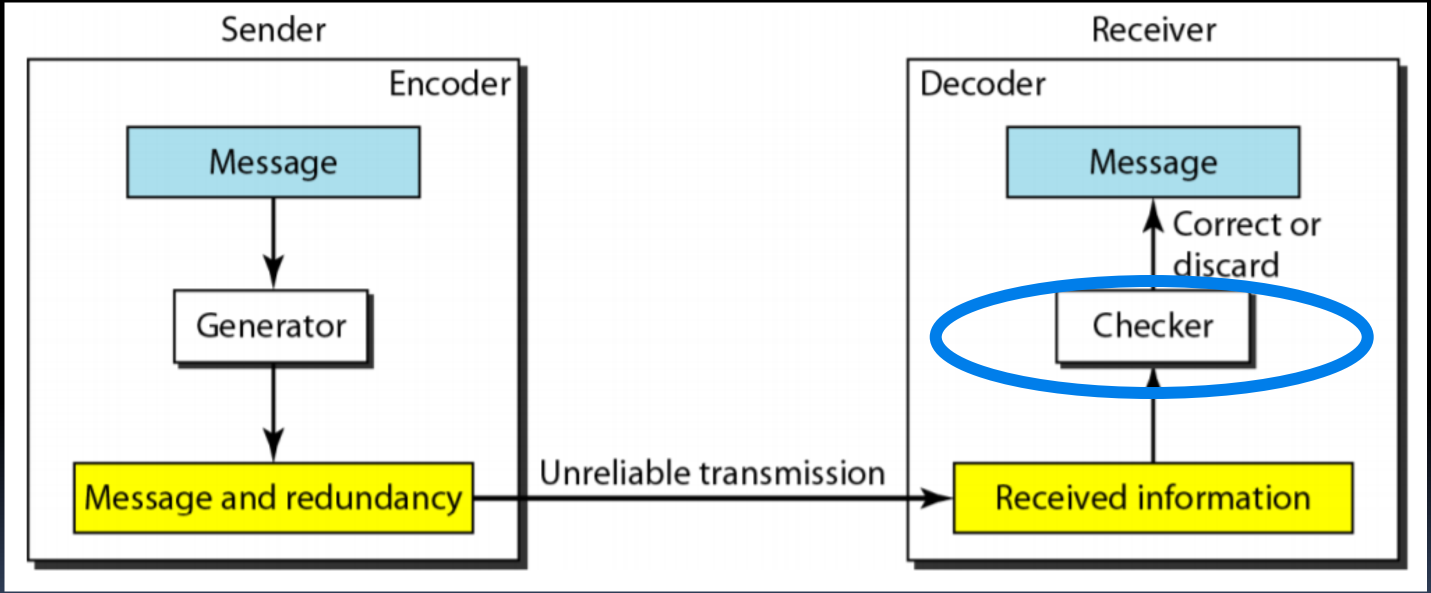
Detection


Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or discard
Checker
Received information

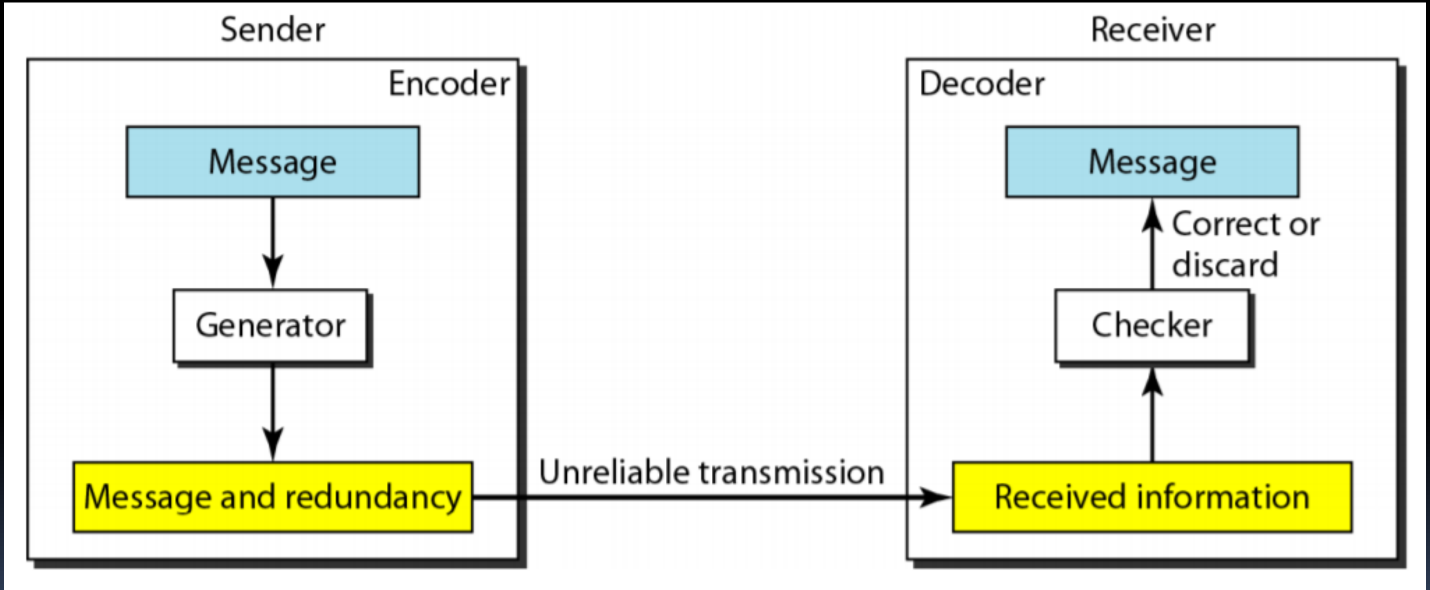
Detection


Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or
discard
Checker
Received information

Detection

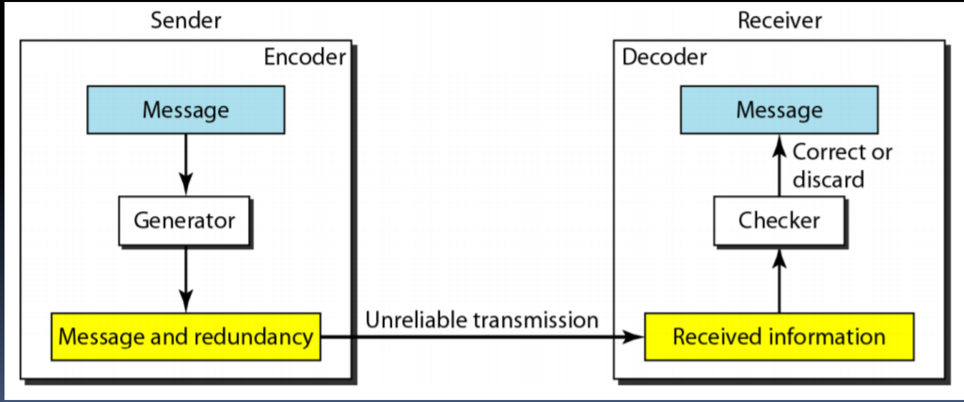

Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or discard
Checker
Received information

Detection


Sender
Encoder
Message
Generator
Message and redundancy
Unreliable transmission
Receiver
Decoder
Message
Correct or discard
Checker
Received information

Detection

- Redundant Data
  - Code
    - Relationship between data and code



Modulo 2

$0 \oplus 0 = 0$ $\qquad$ $1 \oplus 1 = 0$

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

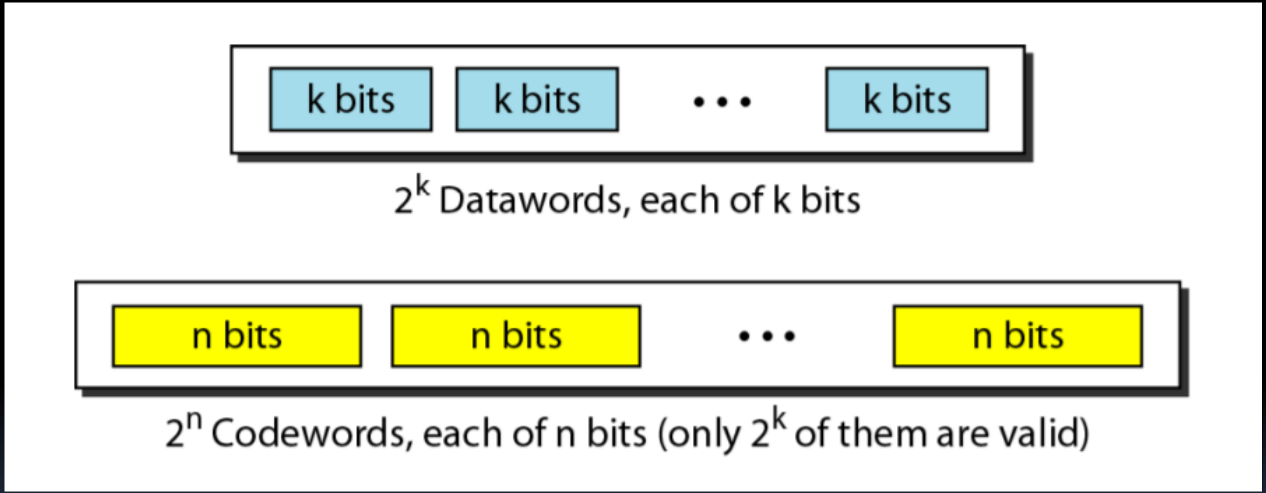
$0 \oplus 1 = 1$ $\qquad$ $1 \oplus 0 = 1$

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

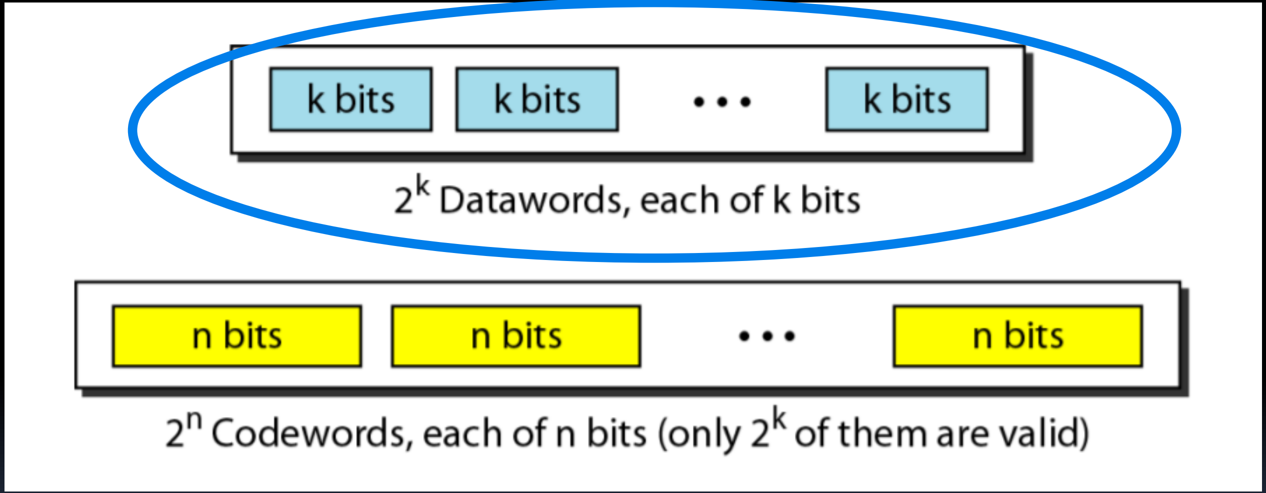
$$\begin{array}{r} 1\ 0\ 1\ 1\ 0 \\ \oplus\ 1\ 1\ 1\ 0\ 0 \\ \hline 0\ 1\ 0\ 1\ 0 \end{array}$$

c. Result of XORing two patterns

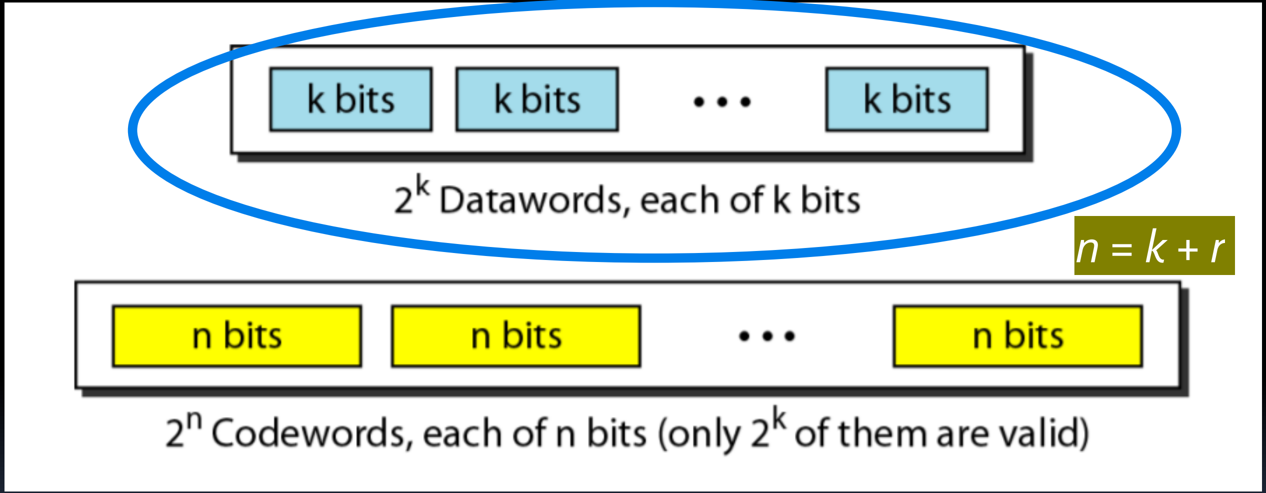
BLOCK CODING



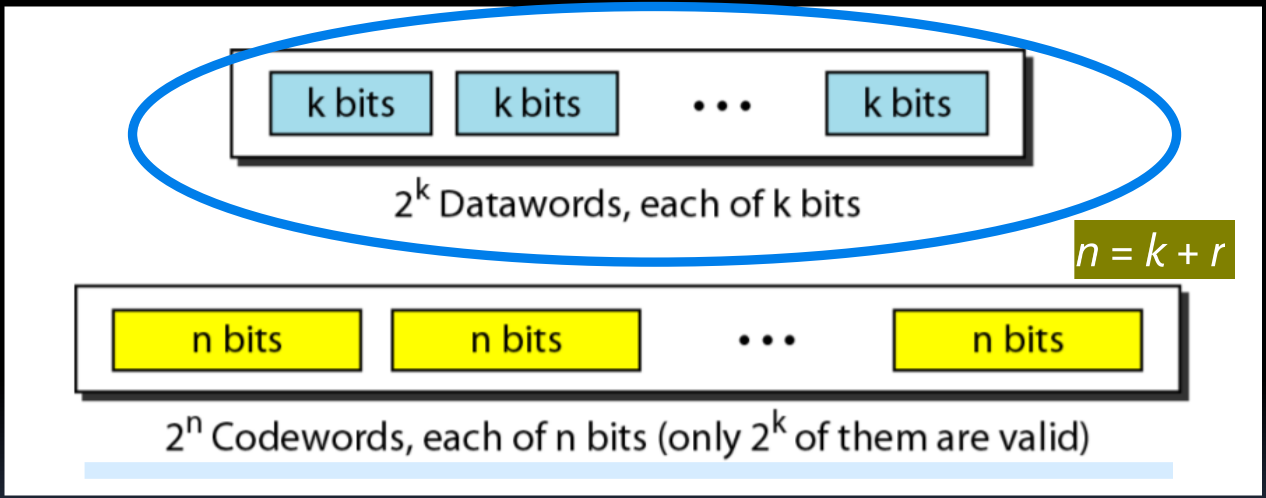
BLOCK CODING



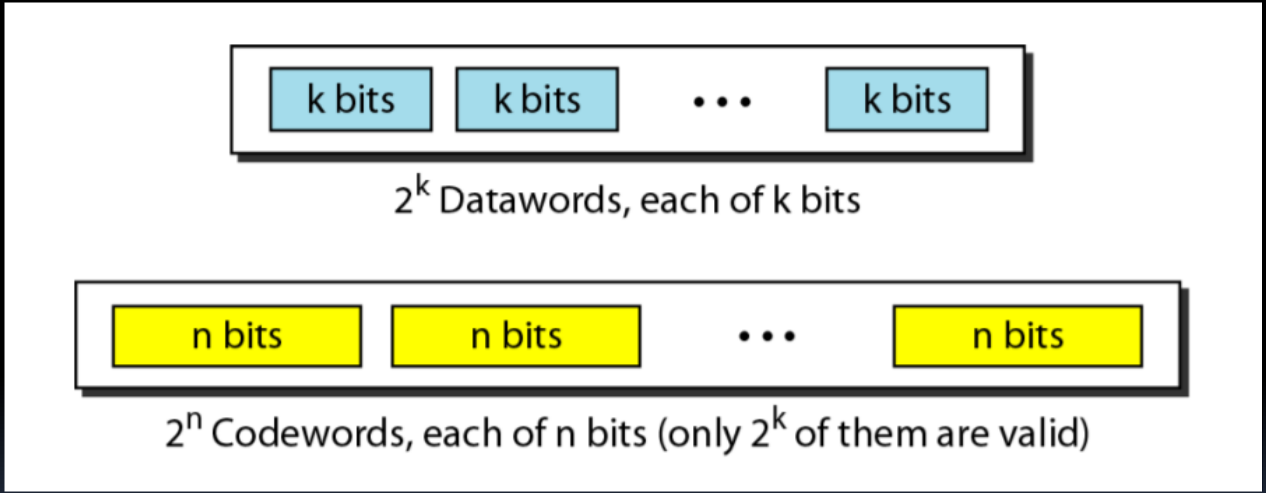
BLOCK CODING



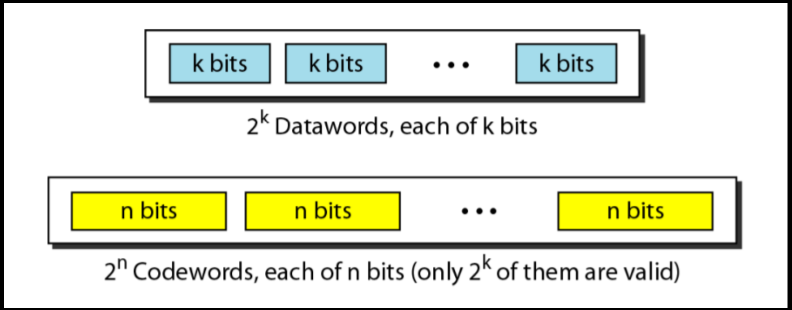
BLOCK CODING



BLOCK CODING

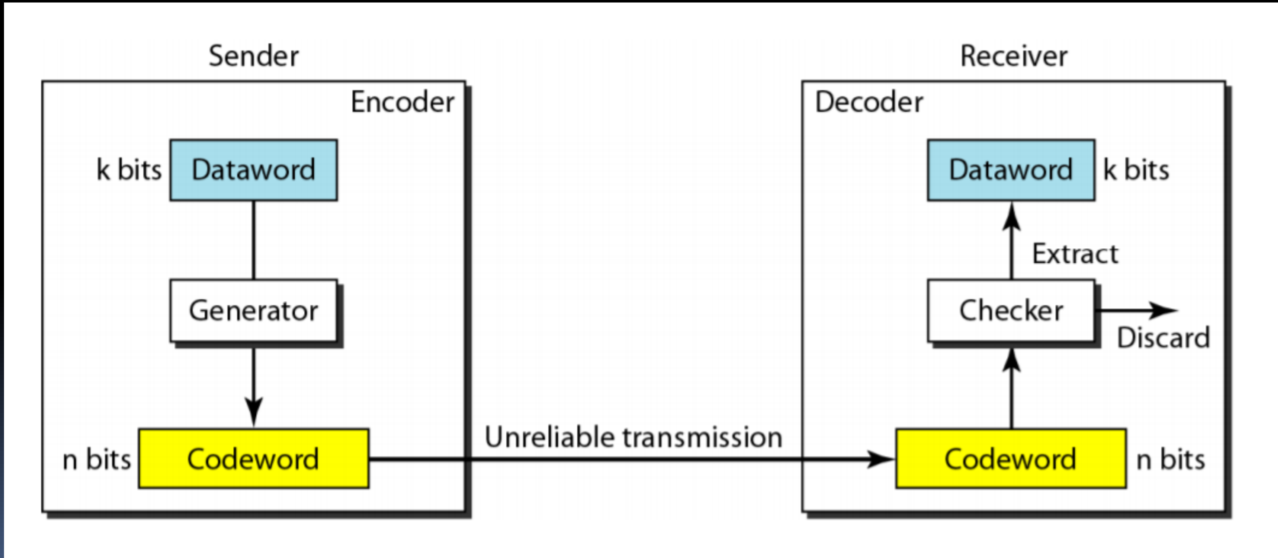


BLOCK CODING



- We divide our message **into blocks, each of *k* bits**
- Add *r* redundant bits to each block to make the length $n = k + r$

Process of error detection in block coding



BLOCK CODING

- Valid and Invalid code

Datawords	Codewords
00	000
01	011
10	101
11	110

BLOCK CODING

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

- Consider the following cases:
  - Sender sends 01→011
  - What if receiver receives
    - 011
    - 111
    - 000

BLOCK CODING

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

- Consider the following cases:
  - Sender sends 01→011
  - What if receiver receives
    - ~~011~~
    - 111
    - 000

BLOCK CODING

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

- Consider the following cases:
  - Sender sends 01→011
  - What if receiver receives
    - ~~011~~
    - ~~111~~
    - 000

BLOCK CODING

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

- Consider the following cases:
  - Sender sends 01→011
  - What if receiver receives
    - ~~011~~
    - ~~111~~
    - ~~000~~

BLOCK CODING

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

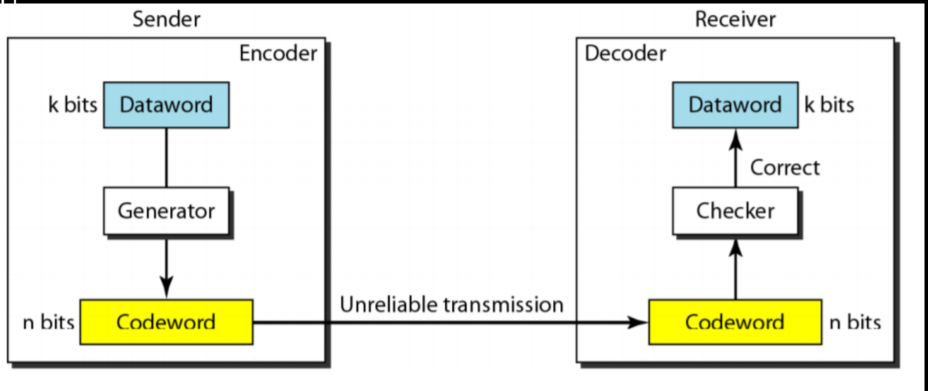
- Consider the following cases:
  - Sender sends 01→011
  - What if receiver receives
    - 011
    - 111
    - 000

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

BLOCK CODING

- Error Correction
  - Where is the error
    - We requires more information to decide this.
    - More redundant data is required

BLOCK CODING



- Error Correction
  - Where is the error
    - We requires more information to decide this.
    - More redundant data is required

Hamming Distance

- The Hamming distance between two words is the number of differences between corresponding bits.

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

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

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

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

Hamming Distance

- The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

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

The d_{min} in this case is 3.

Hamming Distance

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

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

Hamming Distance

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

Linear Block Codes

- A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword

Simple Parity-Check Code

- The extra bit, called the parity bit, is selected to make the total number of **1s** in the codeword even.

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

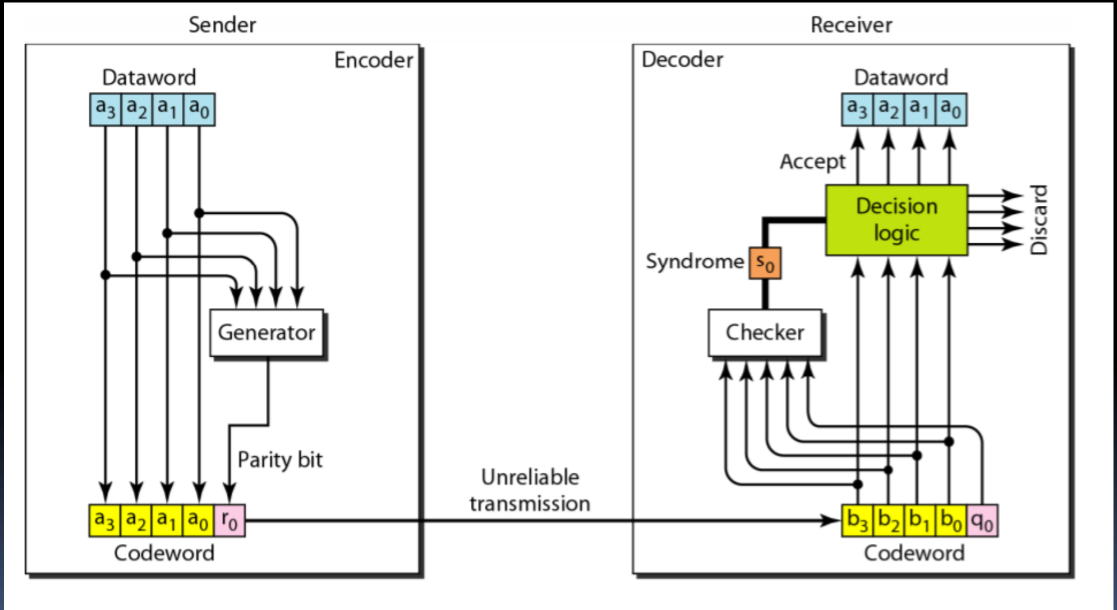
Simple Parity-Check Code

- The extra bit, called the parity bit, is selected to make the total number of **1s** in the codeword even.

Simple Parity-Check Code

$r_0 = a_3 + a_2 + a_1 + a_0$

$S_0 = a_3 + a_2 + a_1 + a_0 + r_0$



Simple Parity-Check Code

- If the even number of bits get changed
- A simple parity-check code can detect an odd of number of errors.

Simple Parity-Check Code

- If the even number of bits get changed
- A simple parity-check code can detect an odd of number of errors.

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	$\boxed{1}$	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	$\boxed{1}$	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	0	1	0	1	1
0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Simple Parity-Check Code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

1	1	0	0	1	1	1	1
1	0	0	0	1	0	1	1
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

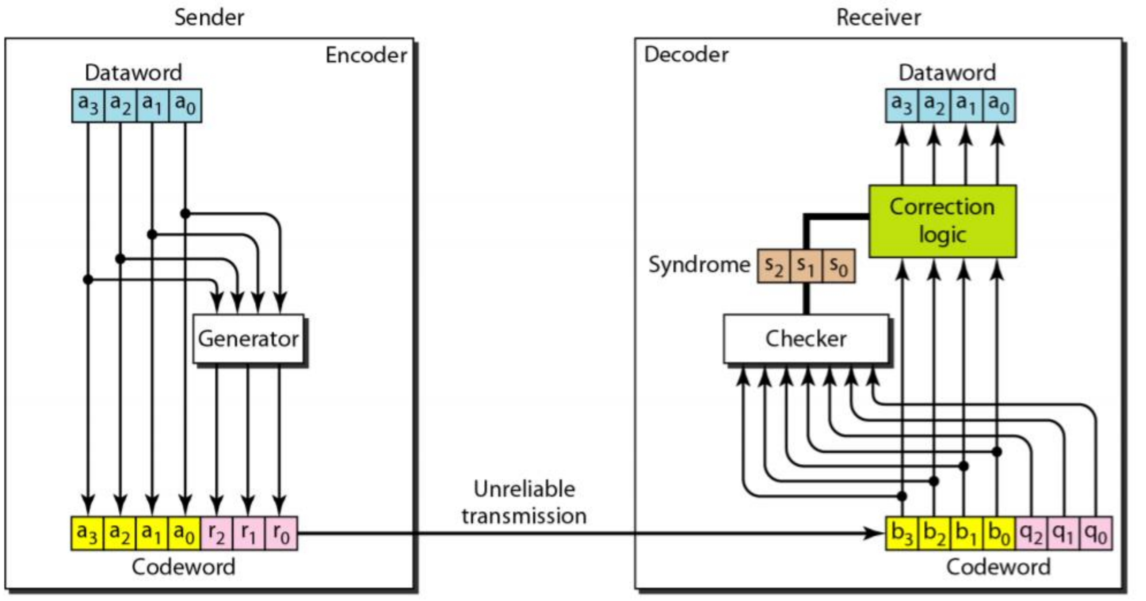
Hamming Codes

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

Dr. Amrit Pal

Hamming Codes

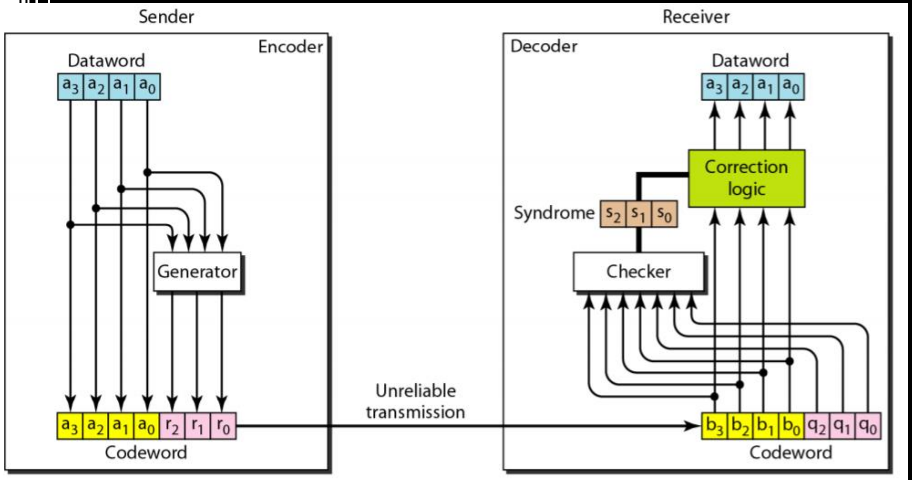


r2=a3+a2+a1

r1=a2+a1+ao

ro=ao+a3+a1

Hamming Codes



r2=a3+a2+a1

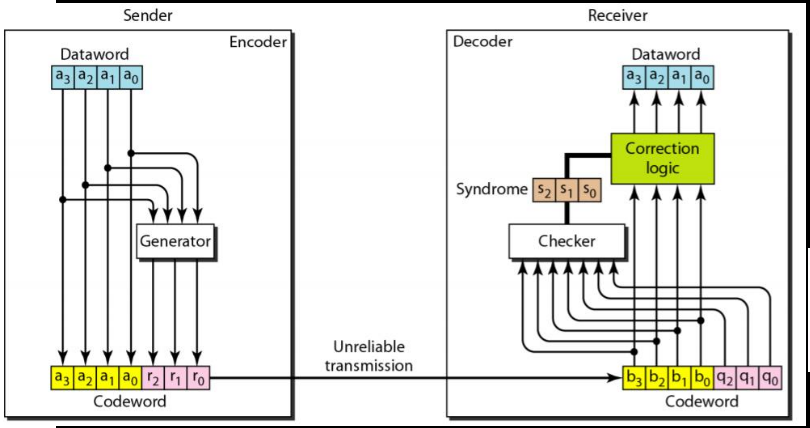
r1=a2+a1+a0

r0=a0+a3+a1

s2=a3+a2+a1+r2

s1=a2+a1+a0+r1

s0=a0+a3+a1+r0



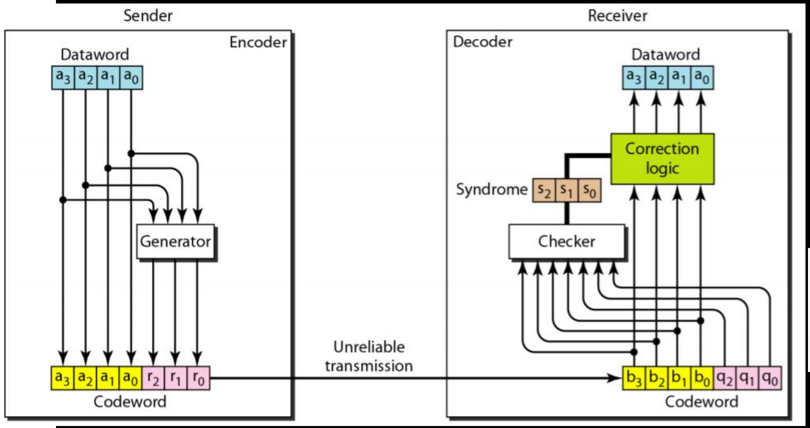
Hamming Codes

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

s2=a3+a2+a1+r2

s1=a2+a1+a0+r1

s0=a0+a3+a1+r0



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

s2=a3+a2+a1+r2

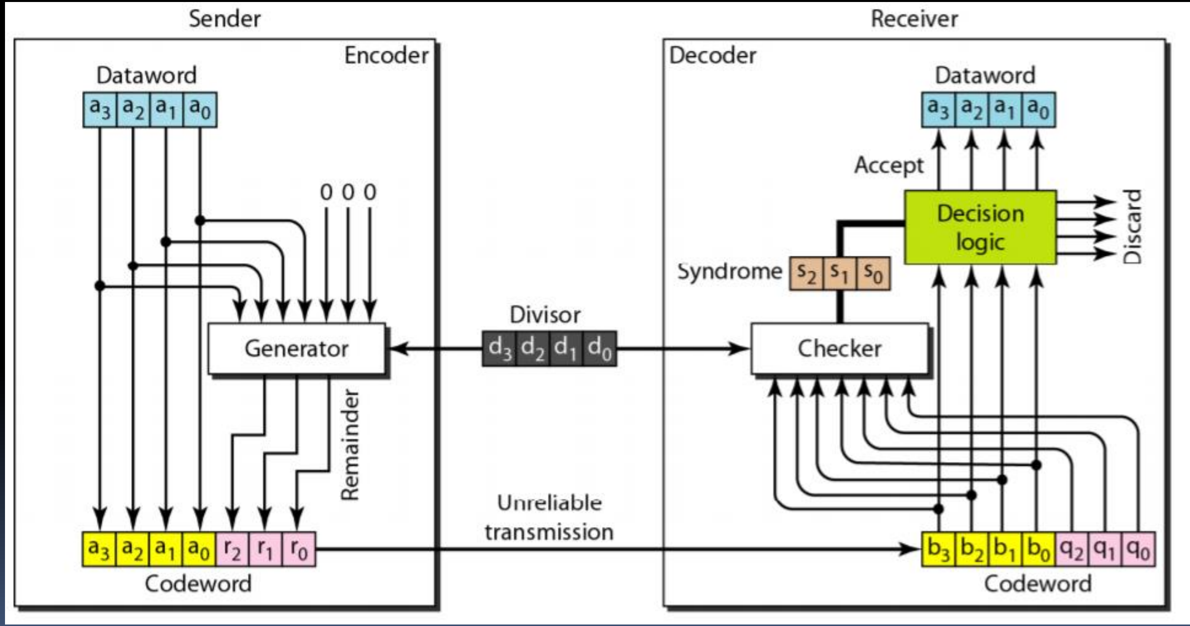
s1=a2+a1+a0+r1

s0=a0+a3+a1+r0

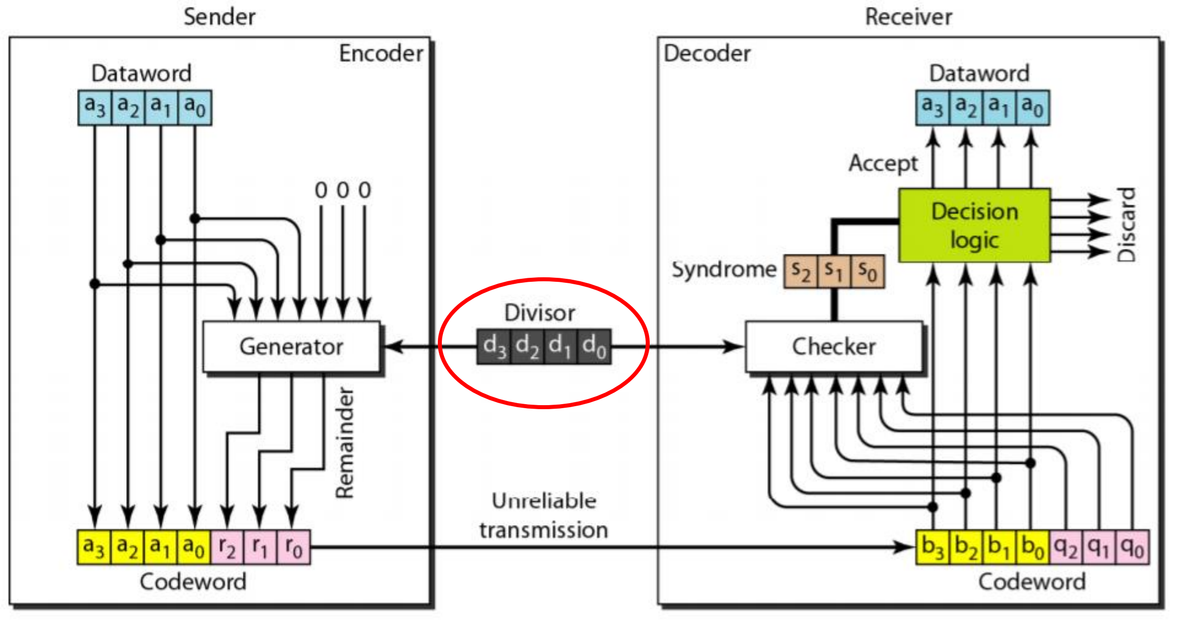
Cyclic Redundancy Check

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

Cyclic Redundancy Check

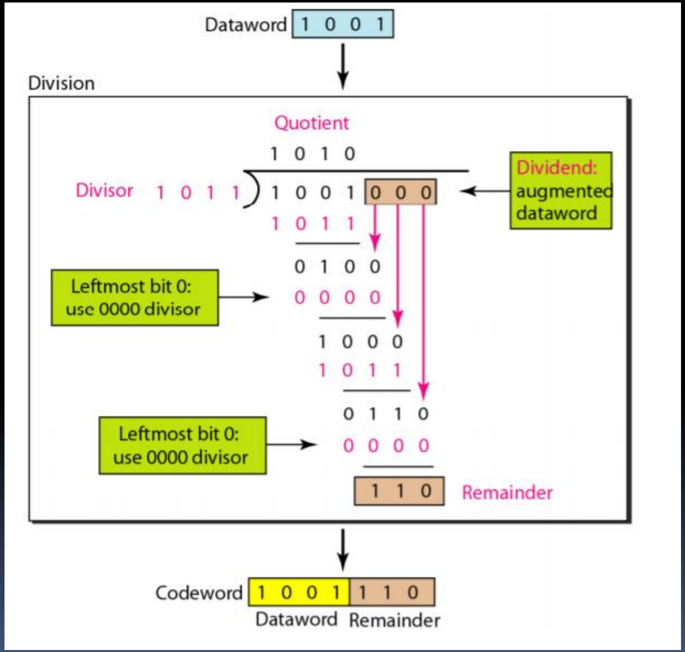


Cyclic Redundancy Check



Cyclic Redundancy Check

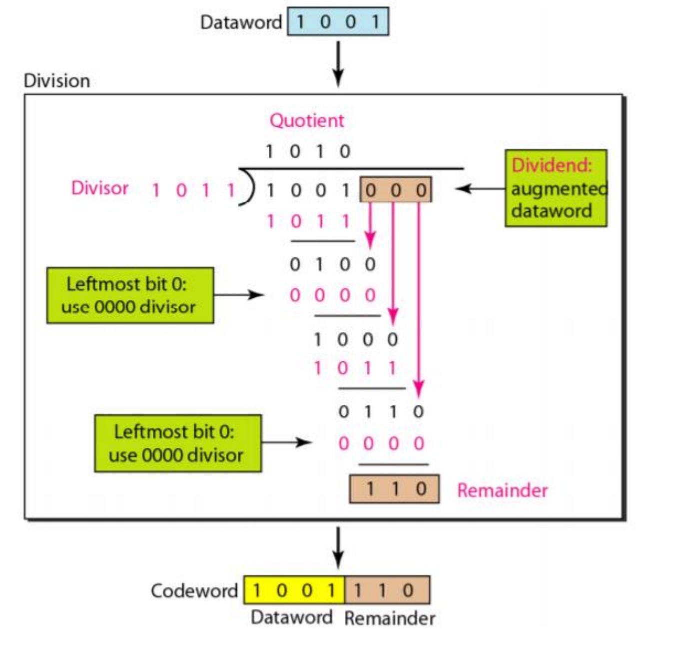
Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation



Cyclic Redundancy Check

Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation

1011|1001000



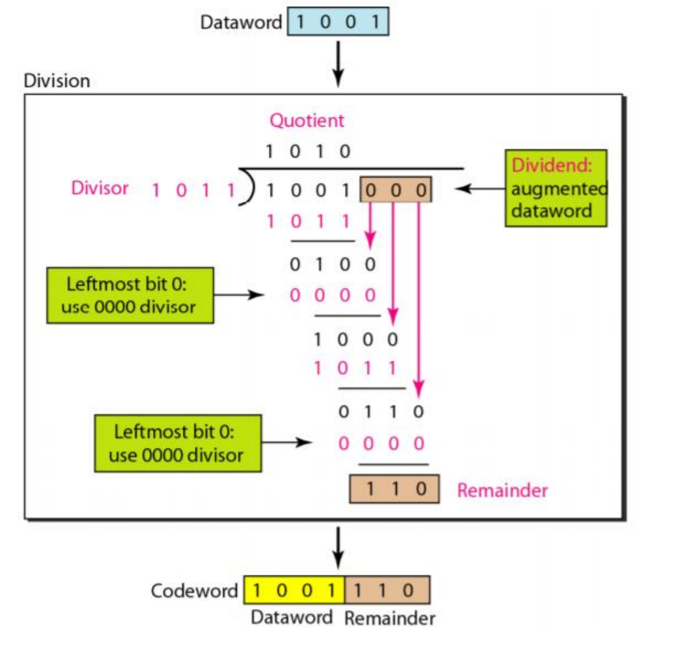
Cyclic Redundancy Check

Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation

1

1011 | 1001000

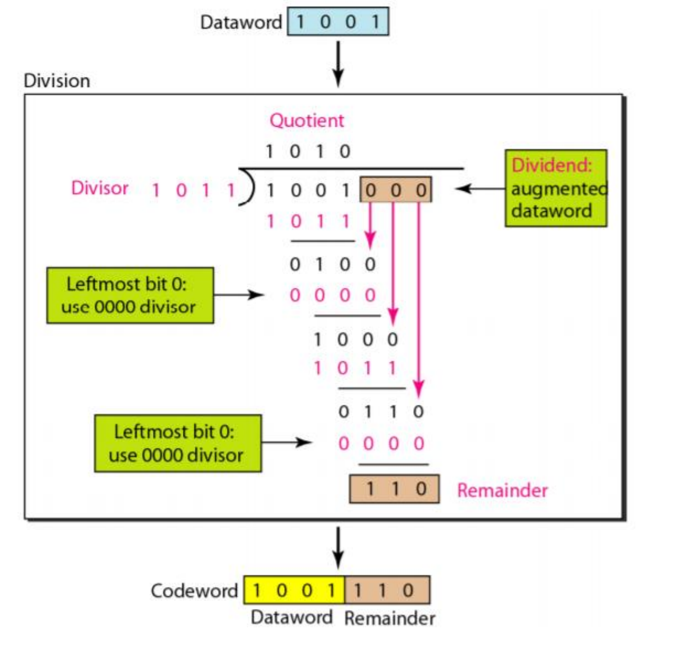
1011



Cyclic Redundancy Check

Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation

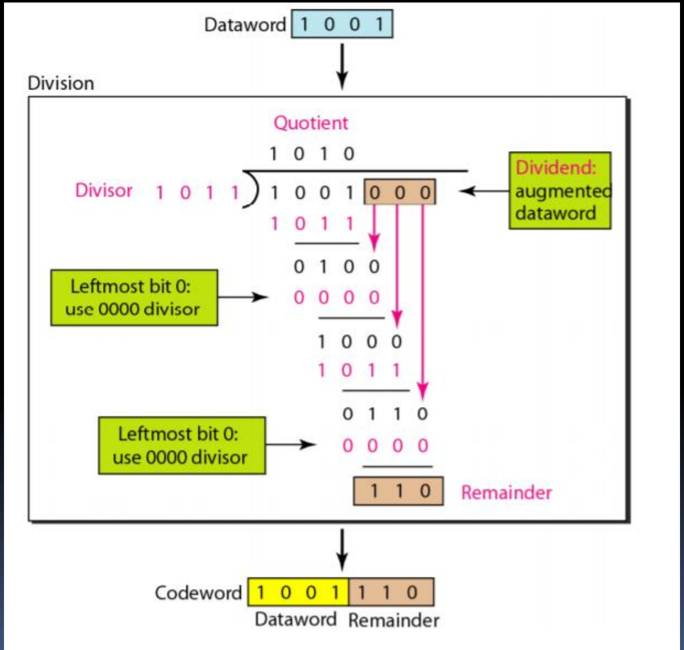
```
          1
1011|1001000
     1011
      010
```



Cyclic Redundancy Check

Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation

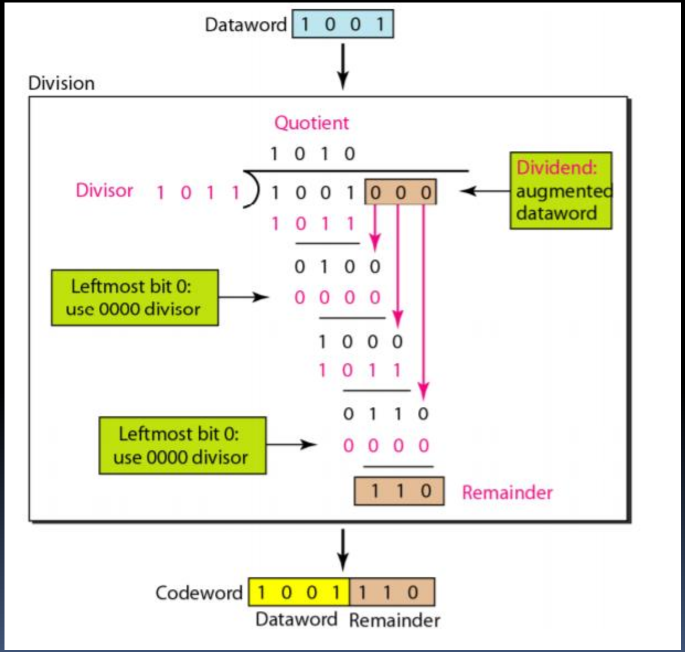
```
      10
1011|1001000
     1011
      0100
      0000
```



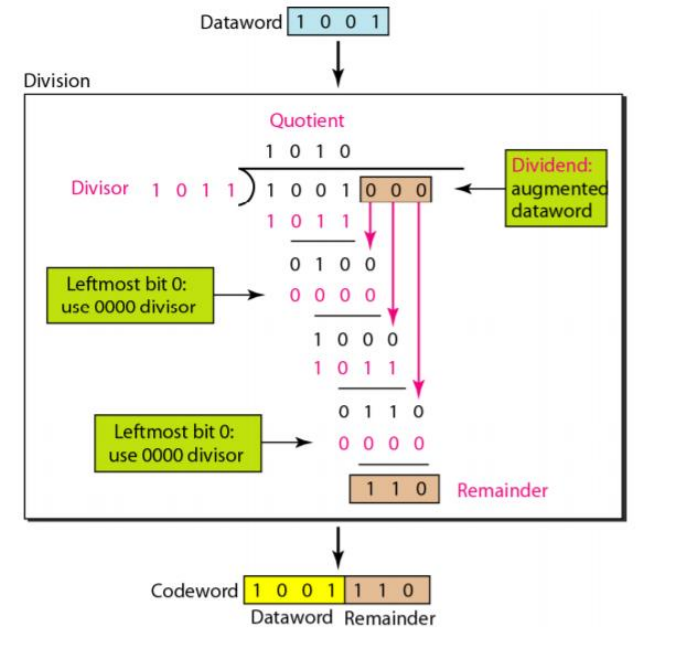
Cyclic Redundancy Check

Divisor=1011
Data=1001
Augmented Bits=000
Remember XOR operation

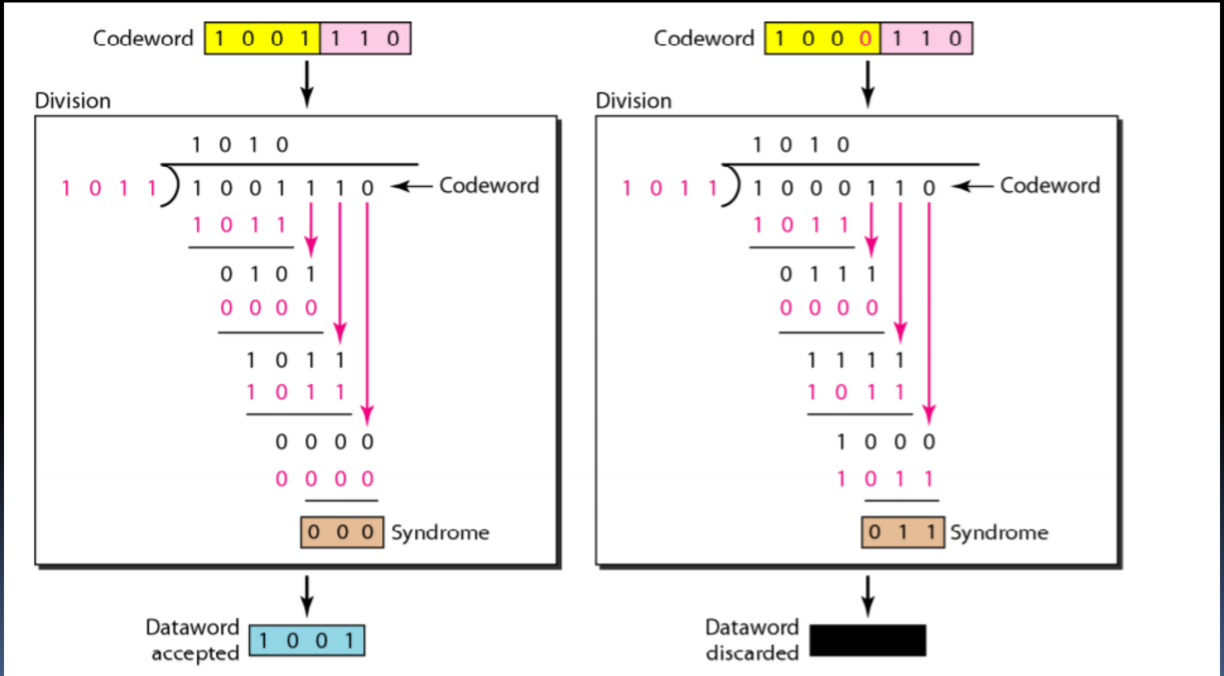
```
        10
1011 | 1001000
       1011
        0100
        0000
```



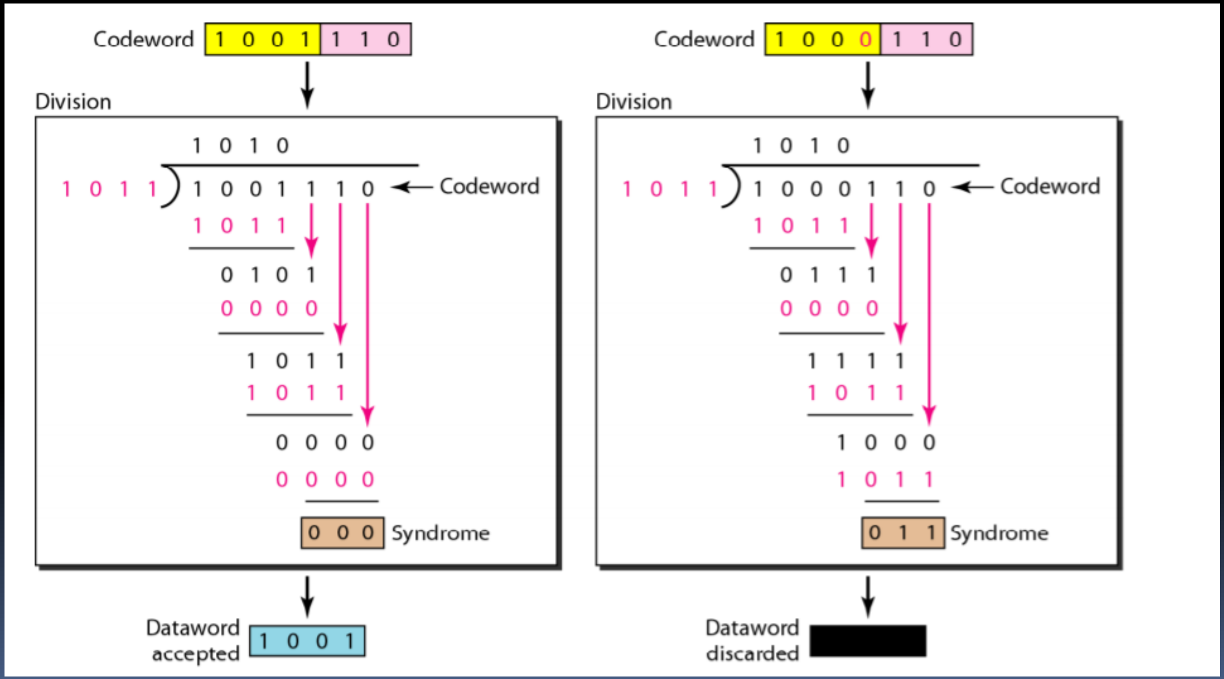
Cyclic Redundancy Check



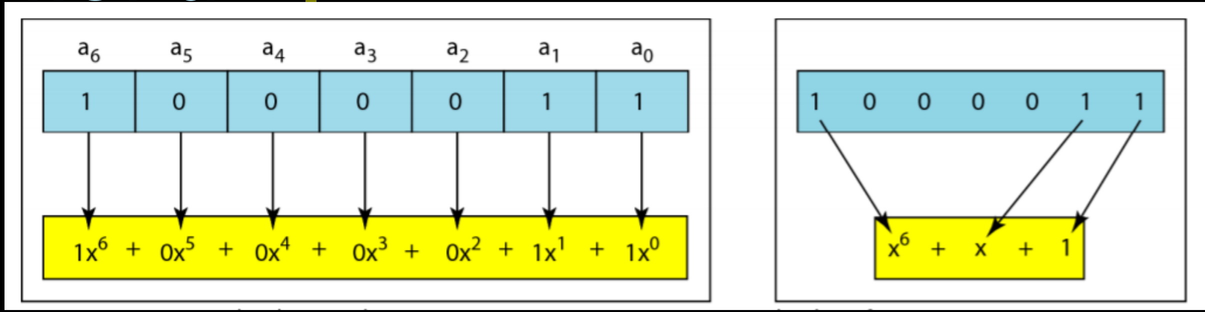
Cyclic Redundancy Check



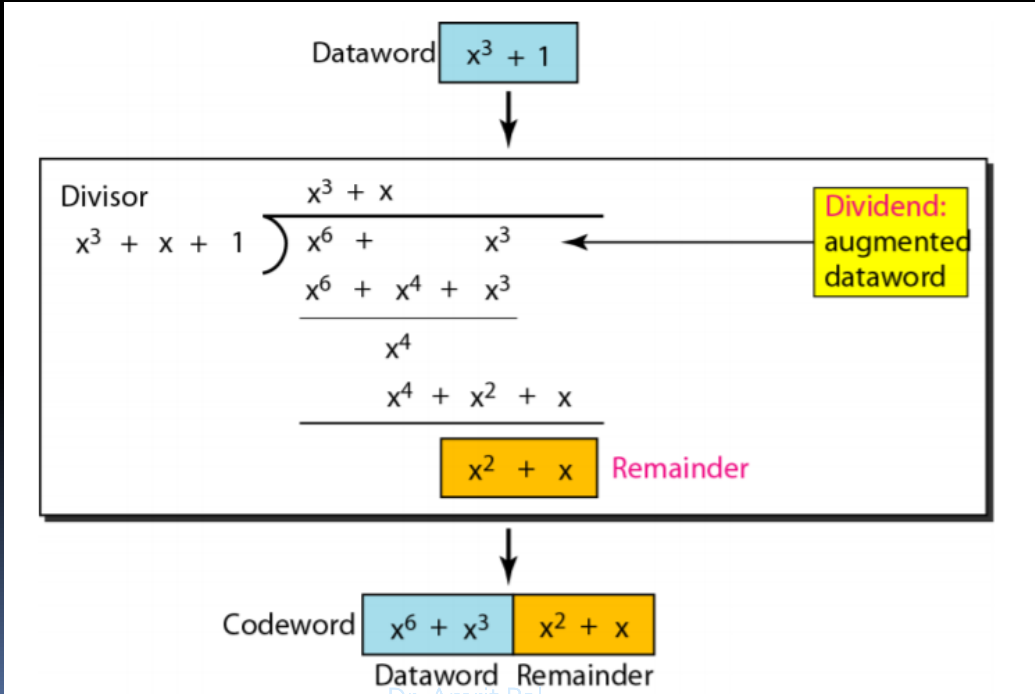
Cyclic Redundancy Check



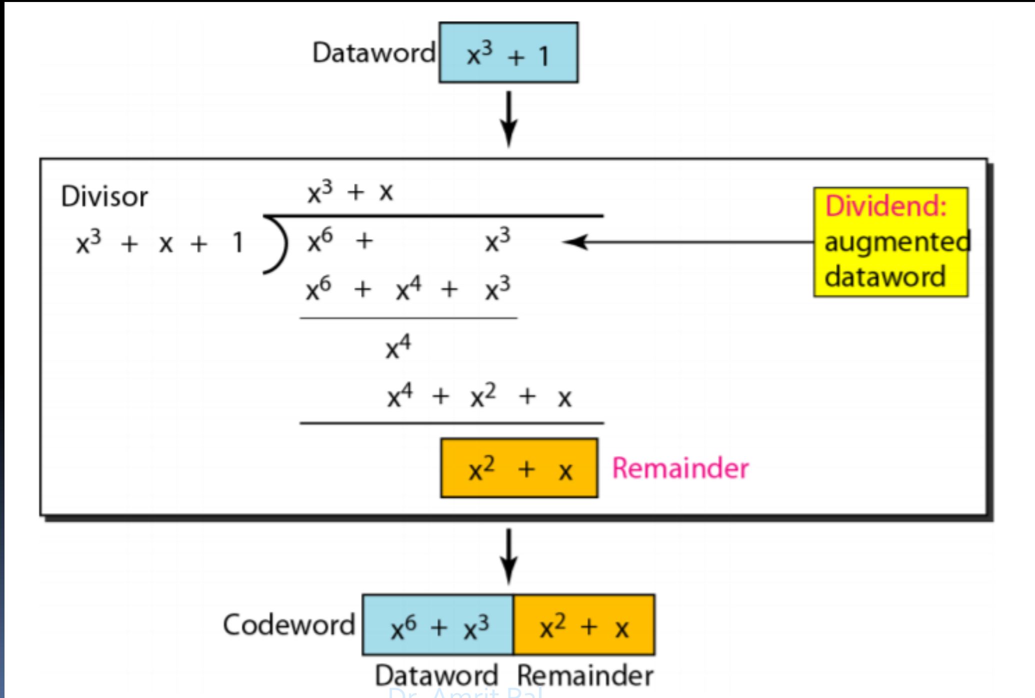
A polynomial to represent a binary word



CRC division using polynomials



CRC division using polynomials



CHECKSUM

- Based on the concept of redundancy
- Example:
- If the set of numbers is (7, 11, 12, 0, 6), we **send** (7, 11, 12,0,6,36), where 36 is the sum of the original numbers.

CHECKSUM

- Based on the concept of redundancy
- Example:
- If the set of numbers is (7, 11, 12, 0, 6), we **send** (7, 11, 12,0,6,36), where 36 is the sum of the original numbers.

- The **receiver** adds the five numbers and compares the result with the sum.

CHECKSUM

Using One's Complement

Dr. Amrit Pal

CHECKSUM

Using One's Complement

- How can we represent the number 21 in one's complement arithmetic using only four bits?

CHECKSUM

Using One's Complement

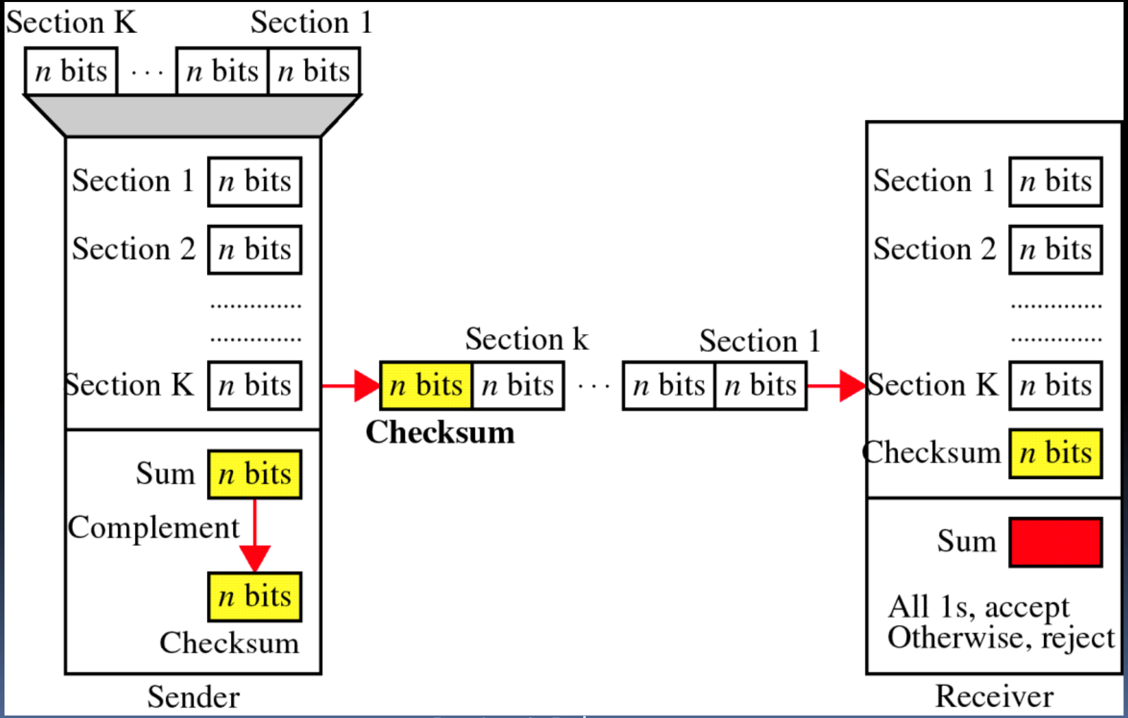
- How can we represent the number 21 in one's complement arithmetic using only four bits?
  - The number 21 in binary is 10101.
  - Wrap the leftmost bit and add it to the four rightmost bits.
  - We have (0101 + 1) = 0110 or 6

CHECKSUM *Using One's Complement*

- The unit is divided into k sections, each of n bits.
- All sections are added together using one's complement to get the sum.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data

CHECKSUM

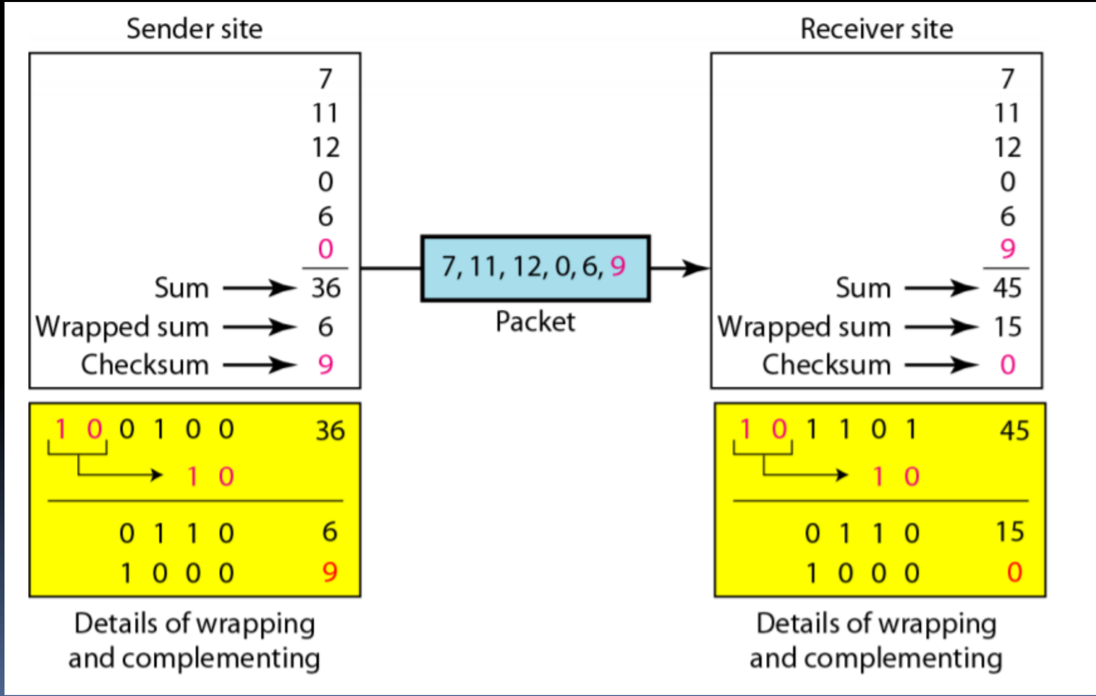
Using One's



CHECKSUM *Using One's Complement*

- The unit is divided into k sections, each of n bits.
- All sections are added together using one's complement to get the sum.
- The sum is complemented.
- If the result is zero, the data are accepted: otherwise, they are rejected.

CHECKSUM
Using One's



References

- Forouzan Behrouz, A. (2008). Data Communication and networking.
- Tanenbaum, A. S. (2011). Computer Networks, /Andrew S. Tanenbaum, David J. Wetherall. *Cloth: Prentice Hall*.