UNIT 5

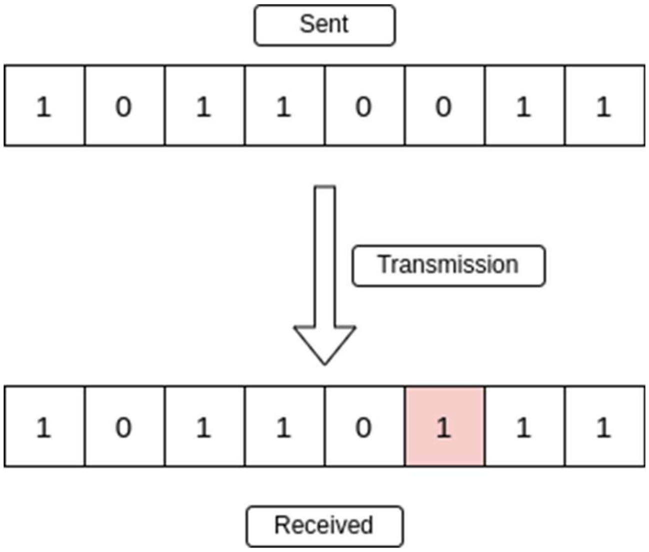
DATA COMMUNICATION

Error is a condition when the receiver's information does not match the sender's information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits traveling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.
Data (Implemented either at the Data link layer or Transport Layer of the OSI Model) may get scrambled by noise or get corrupted whenever a message is transmitted. To prevent such errors, error-detection codes are added as extra data to digital messages. This helps in detecting any errors that may have occurred during message transmission.

Types of Errors

Single-Bit Error

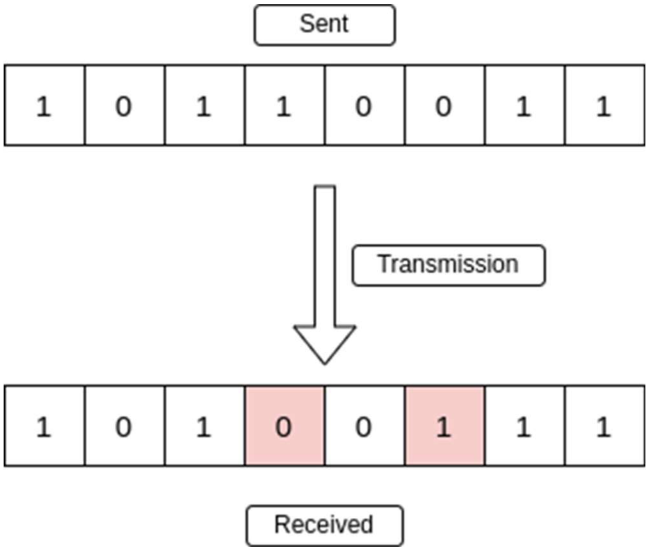
A single-bit error refers to a type of data transmission error that occurs when one bit (i.e., a single binary digit) of a transmitted data unit is altered during transmission, resulting in an incorrect or corrupted data unit.



Single-Bit Error

Multiple-Bit Error

A multiple-bit error is an error type that arises when more than one bit in a data transmission is affected. Although multiple-bit errors are relatively rare when compared to single-bit errors, they can still occur, particularly in high-noise or high-interference digital environments.



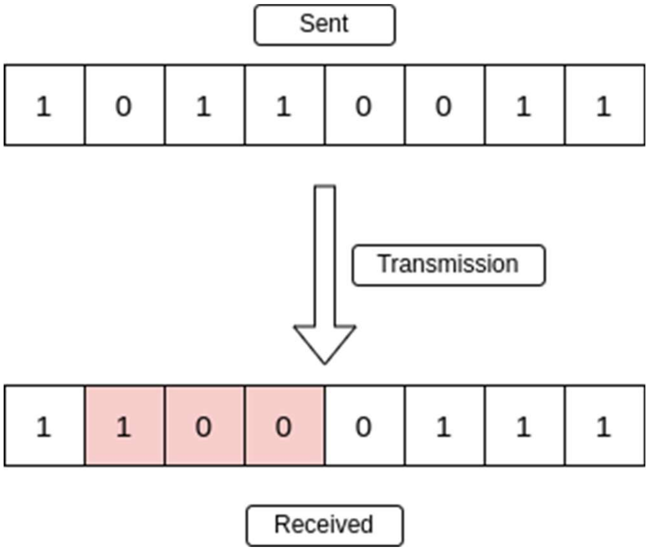
1 0 1 1 0 0 1 1

1 0 1 0 0 1 1 1

Multiple-Bit Error

Burst Error

When several consecutive bits are flipped mistakenly in digital transmission, it creates a burst error. This error causes a sequence of consecutive incorrect values.



1 0 1 1 0 0 1 1

1 1 0 0 0 1 1 1

Burst Error

To detect errors, a common technique is to introduce redundancy bits that provide additional information. Various techniques for error detection include:

1. Simple Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)

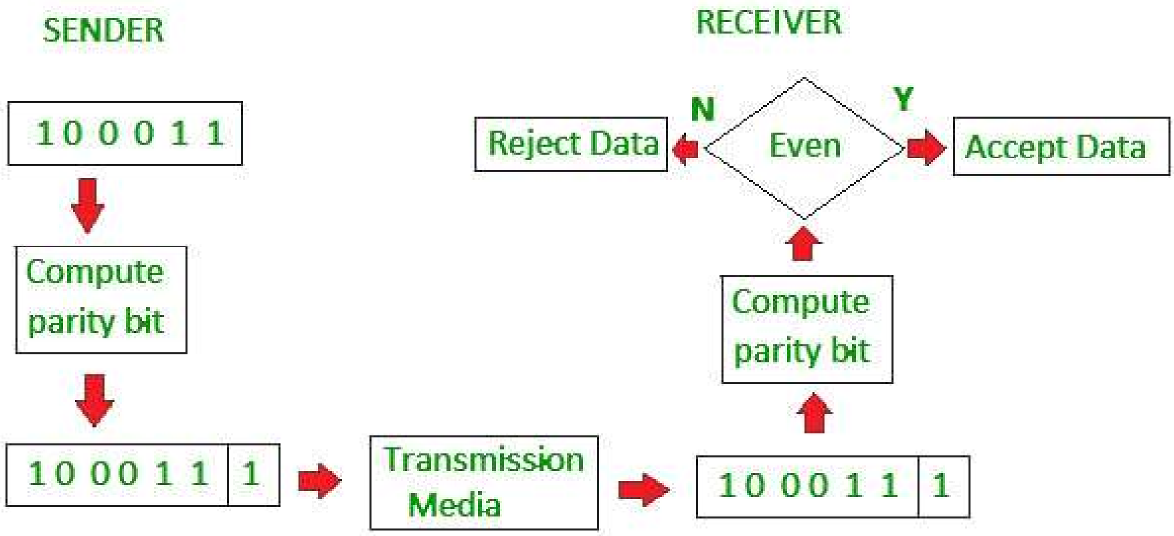
Error Detection Methods

Simple Parity Check

Simple-bit parity is a simple error detection method that involves adding an extra bit to a data transmission. It works as:

- 1 is added to the block if it contains an odd number of 1's, and
- 0 is added if it contains an even number of 1's

This scheme makes the total number of 1's even, that is why it is called even parity checking.



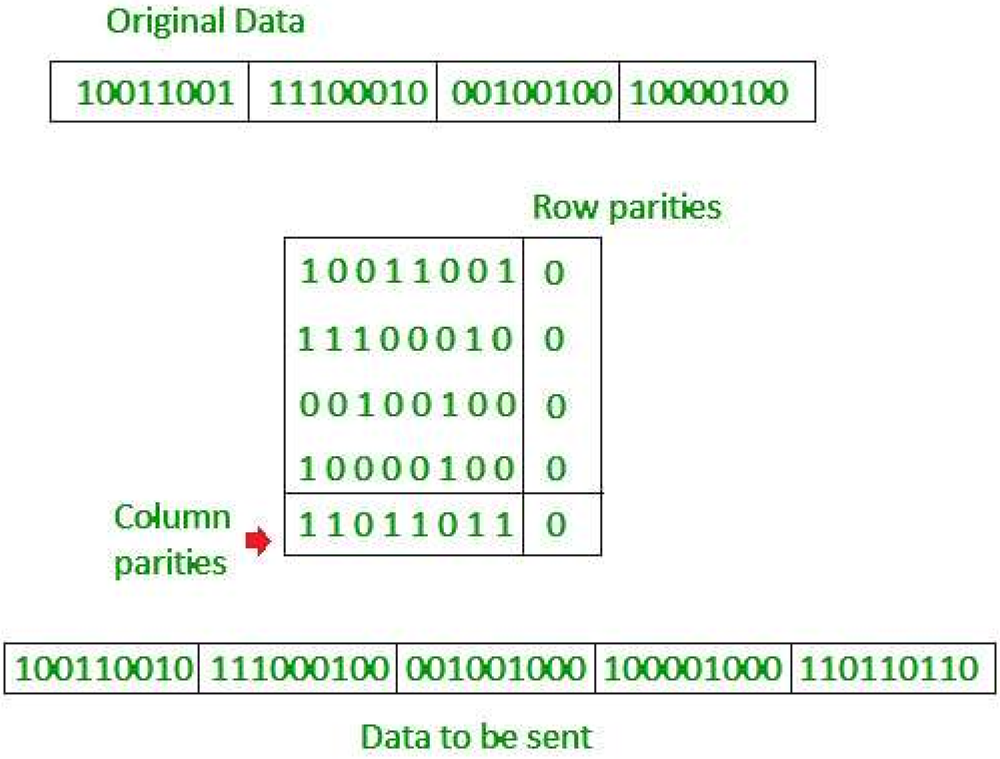
Disadvantages

- Single Parity check is not able to detect even no. of bit error.
- **For example,** the Data to be transmitted is **101010**. Codeword transmitted to the receiver is 1010101 (we have used even parity). Let's assume that during transmission, two of the bits of code word

flipped to 1111101.
On receiving the code word, the receiver finds the no. of ones to be even and hence **no error,** *which is a wrong assumption.*

Two-dimensional Parity Check

Two-dimensional Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns, then both are sent along with the data. At the receiving end, these are compared with the parity bits calculated on the received data.



10011001	11100010	00100100	10000100

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

100110010	111000100	001001000	100001000	110110110

Checksum

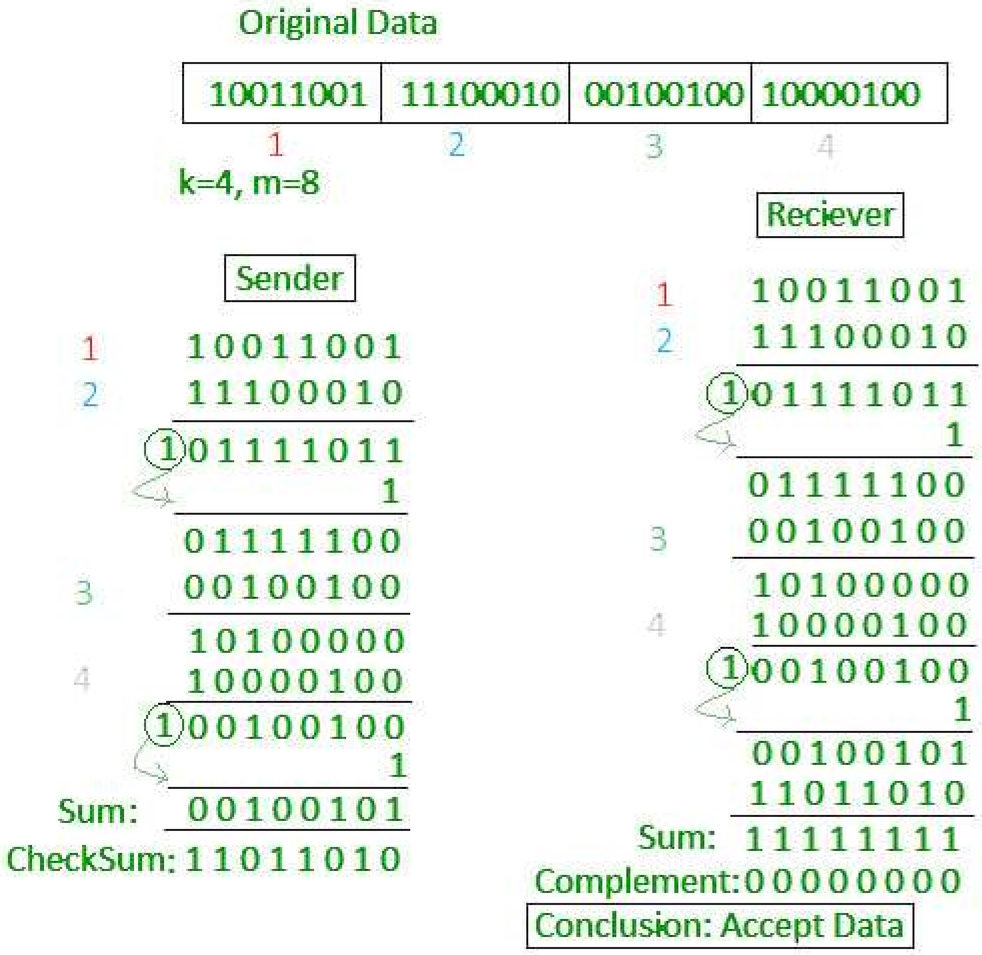
Checksum error detection is a method used to identify errors in transmitted data. The process involves dividing the data into equally sized segments and using a 1's complement to calculate the sum of these segments. The calculated sum is then sent along with the data to the receiver. At the receiver's end, the same process is repeated and if all zeroes are obtained in the sum, it means that the data is correct.

Checksum – Operation at Sender's Side

- Firstly, the data is divided into k segments each of m bits.
- On the sender's end, the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.

Checksum – Operation at Receiver's Side

- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded.

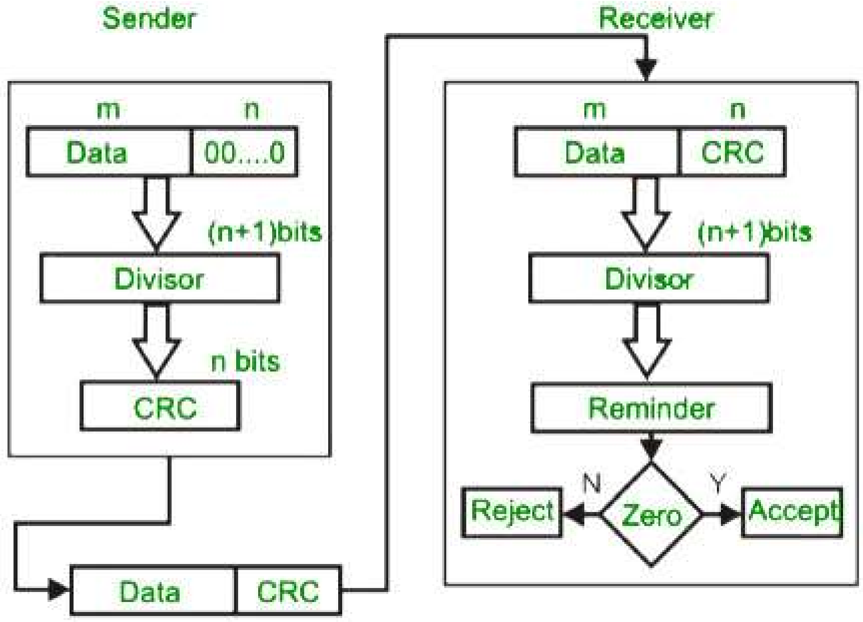


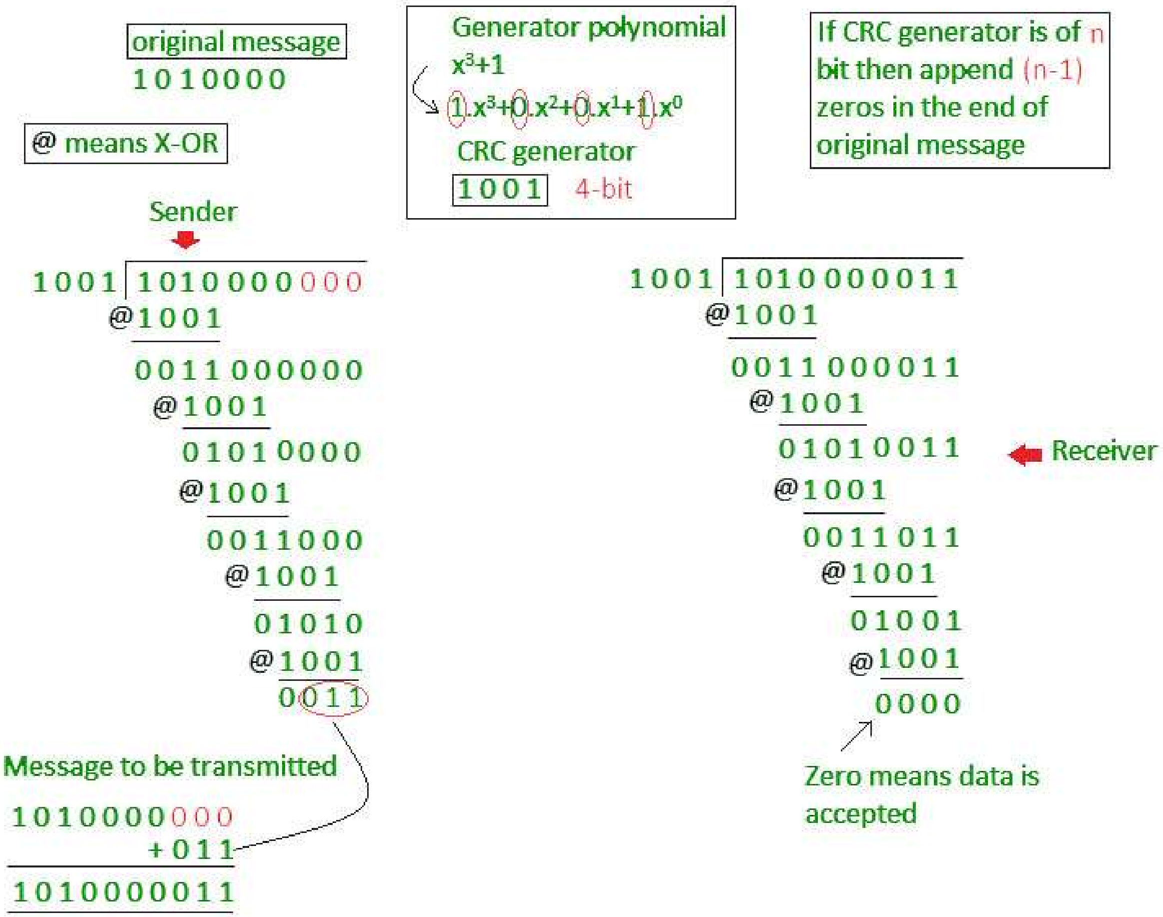
Disadvantages

- If one or more bits of a segment are damaged and the corresponding bit or bits of opposite value in a second segment are also damaged.

Cyclic Redundancy Check (CRC)

- Unlike the checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of the data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.





Advantages:

Increased Data Reliability: Error detection ensures that the data transmitted over the network is reliable, accurate, and free from errors. This ensures that the recipient receives the same data that was transmitted by the sender.

Improved Network Performance: Error detection mechanisms can help to identify and isolate network issues that are causing errors. This can help to improve the overall performance of the network and reduce downtime.

Enhanced Data Security: Error detection can also help to ensure that the data transmitted over the network is secure and has not been tampered with.

Disadvantages:

Overhead: Error detection requires additional resources and processing power, which can lead to increased overhead on the network. This can result in slower network performance and increased latency.

False Positives**:** Error detection mechanisms can sometimes generate false positives, which can result in unnecessary retransmission of data. This can further increase the overhead on the network.

Limited Error Correction**:** Error detection can only identify errors but cannot correct them. This means that the recipient must rely on the sender to retransmit the data, which can lead to further delays and increased network overhead.

Hamming Code in Computer Network

Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is a **technique developed by R.W. Hamming for error correction**. **Redundant bits –** Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

```
2^r ≥ m + r + 1

where, r = redundant bit, m = data bit
```

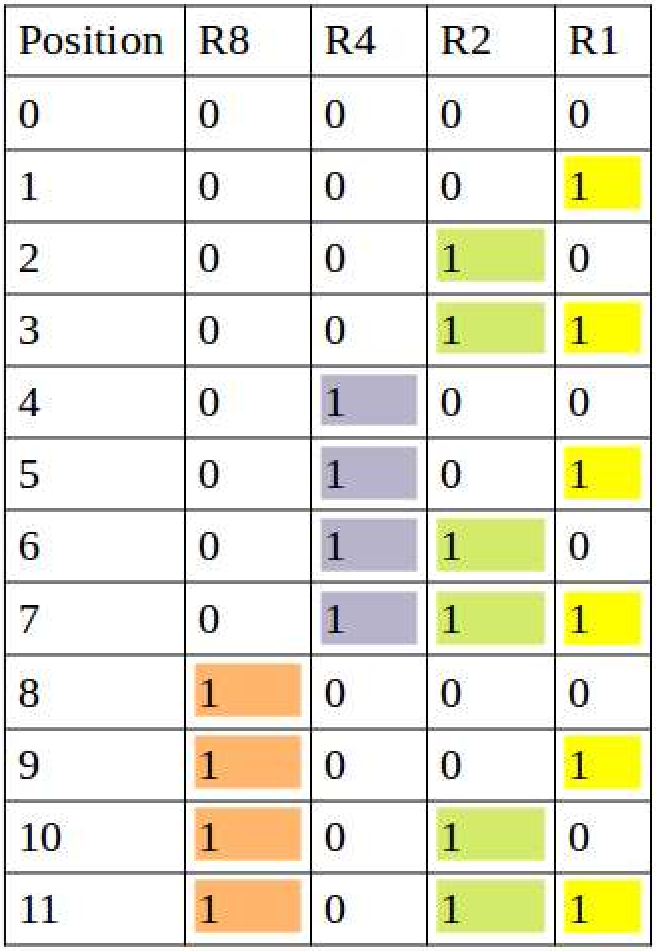
Suppose the number of data bits is 7, then the number of redundant bits can be calculated using: = 2^4 ≥ 7 + 4 + 1 Thus, the number of redundant bits= 4 **Parity bits.** A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1’s in the data is even or odd. Parity bits are used for error detection. There are two types of parity bits:

1. **Even parity bit:** In the case of even parity, for a given set of bits, the number of 1’s are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1’s an even number. If the total number of 1’s in a given set of bits is already even, the parity bit’s value is 0.
2. **Odd Parity bit –** In the case of odd parity, for a given set of bits, the number of 1’s are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1’s an odd number.

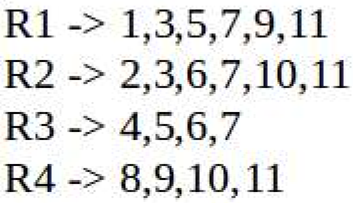
If the total number of 1’s in a given set of bits is already odd, the parity bit’s value is 0.

General Algorithm of Hamming code: Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form. **a.** Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc). **b.** Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from the least significant bit (2, 3, 6, 7, 10, 11, etc). **c.** Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc). **d.** Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit bits (8–15, 24–31, 40–47, etc). **e.** In general, each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.
5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.



Position	R8	R4	R2	R1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

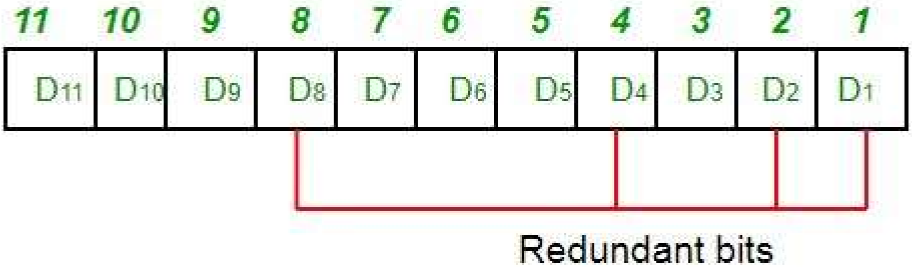


R1 -> 1,3,5,7,9,11
R2 -> 2,3,6,7,10,11
R3 -> 4,5,6,7
R4 -> 8,9,10,11

Determining the position of redundant bits – These redundancy bits are placed at positions that correspond to the power of 2.
As in the above example:

- The number of data bits = 7
- The number of redundant bits = 4
- The total number of bits = 11

- The redundant bits are placed at positions corresponding to power of 2- 1, 2, 4, and 8



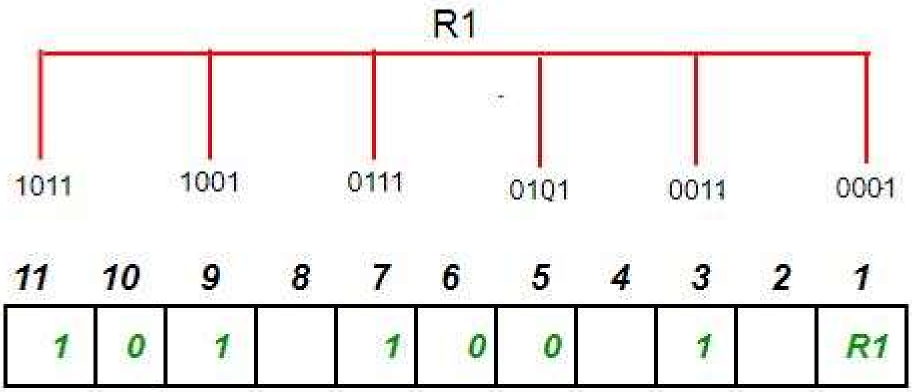
11	10	9	8	7	6	5	4	3	2	1
D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1

- Suppose the data to be transmitted is 1011001, the bits will be placed as follows:

11	10	9	8	7	6	5	4	3	2	1
1	0	1	R8	1	0	0	R4	1	R2	R1

Determining the Parity bits:

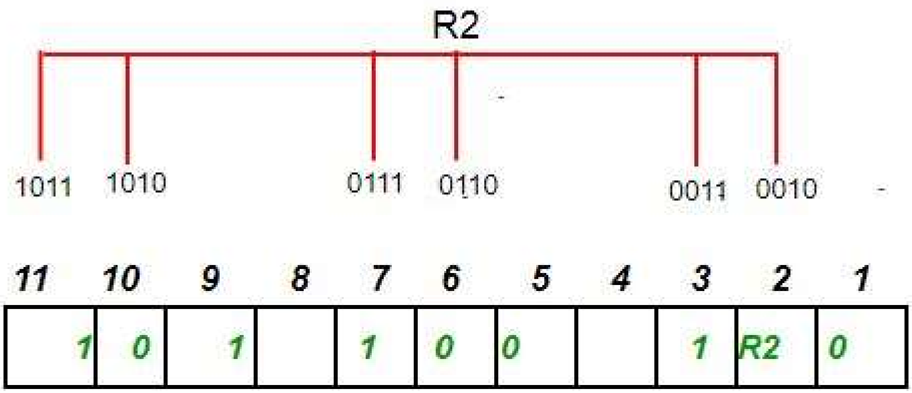
- R1 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the least significant position. R1: bits 1, 3, 5, 7, 9, 11



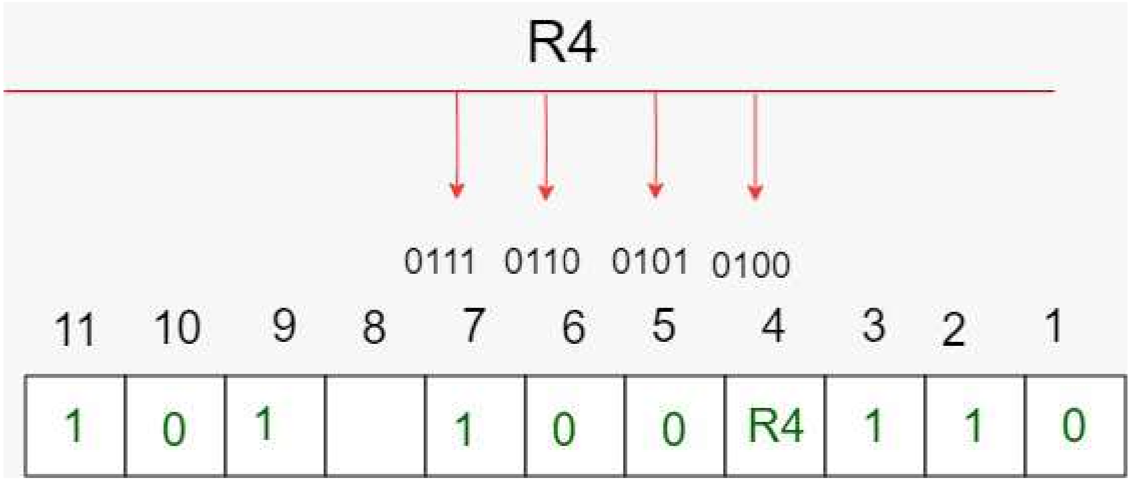
11	10	9	8	7	6	5	4	3	2	1
1	0	1		1	0	0		1		R1

- To find the redundant bit R1, we check for even parity. Since the total number of 1’s in all the bit positions corresponding to R1 is an even number the value of R1 (parity bit’s value) = 0

- R2 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the second position from the least significant bit. R2: bits 2,3,6,7,10,11

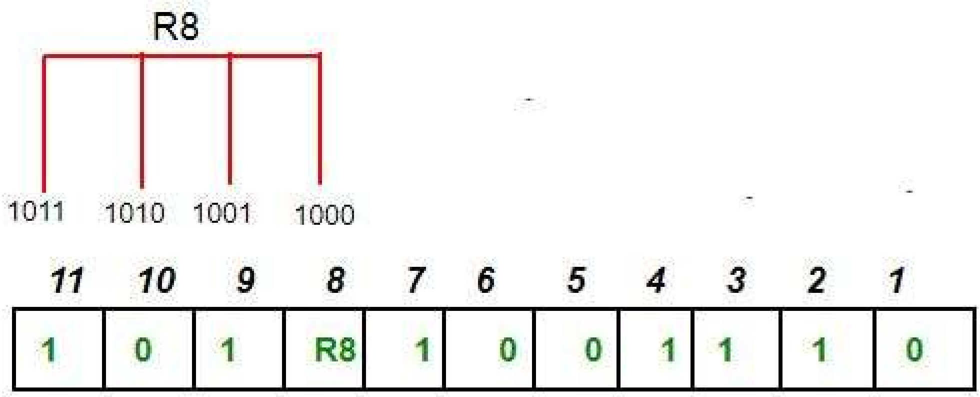


- To find the redundant bit R2, we check for even parity. Since the total number of 1’s in all the bit positions corresponding to R2 is odd the value of R2(parity bit’s value)=1
- R4 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the third position from the least significant bit. R4: bits 4, 5, 6, 7



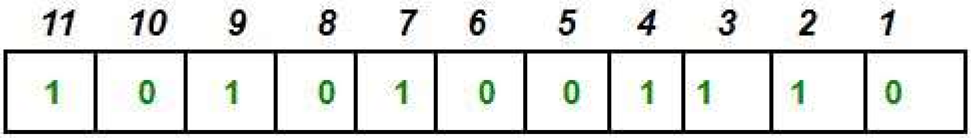
1. To find the redundant bit R4, we check for even parity. Since the total number of 1’s in all the bit positions corresponding to R4 is odd the value of R4(parity bit’s value) = 1
2. R8 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the fourth position from the least

significant bit. R8: bit 8,9,10,11



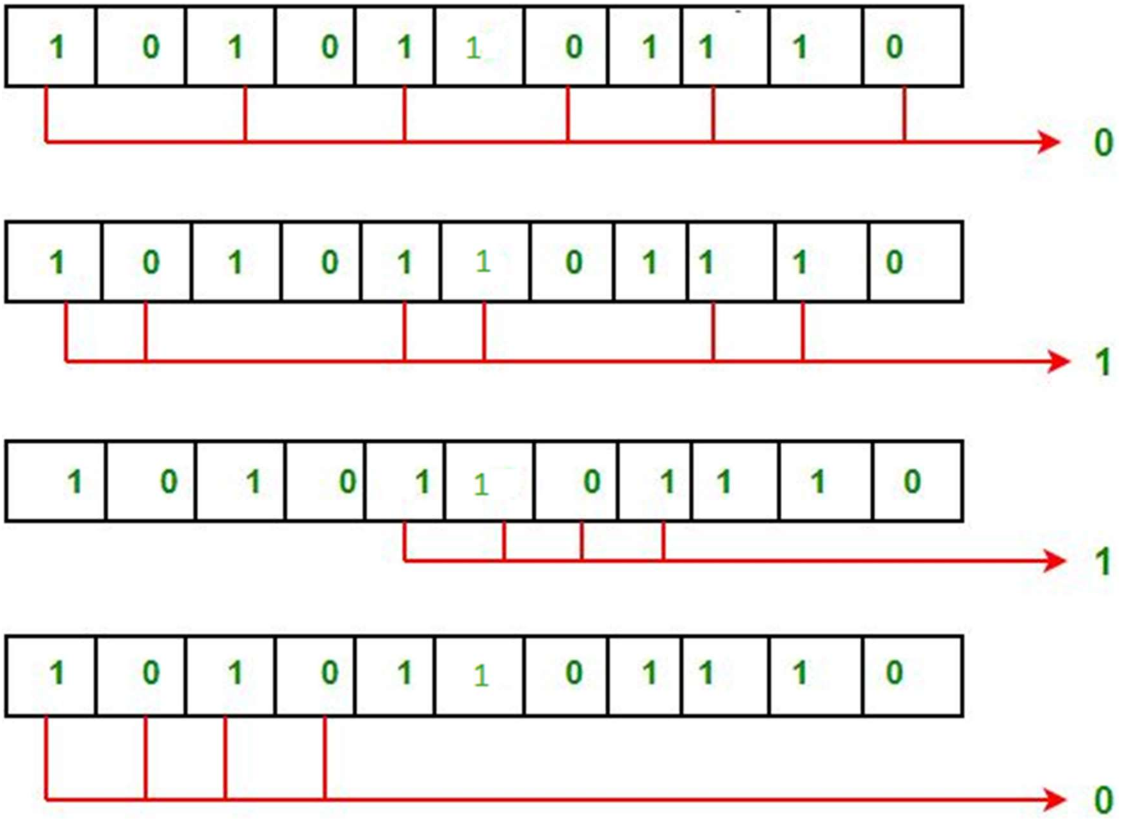
11	10	9	8	7	6	5	4	3	2	1
1	0	1	R8	1	0	0	1	1	1	0

- To find the redundant bit R8, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R8 is an even number the value of R8(parity bit's value)=0. Thus, the data transferred is:



11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	0	1	1	1	0

Error detection and correction: Suppose in the above example the 6th bit is changed from 0 to 1 during data transmission, then it gives new parity values in the binary number:



For all the parity bits we will check the number of 1’s in their respective bit positions.

For R1: bits 1, 3, 5, 7, 9, 11. We can see that the number of 1’s in these bit positions are 4 and that’s even so we get a 0 for this.

For R2: bits 2,3,6,7,10,11 . We can see that the number of 1’s in these bit positions are 5 and that’s odd so we get a 1 for this.

For R4: bits 4, 5, 6, 7 . We can see that the number of 1’s in these bit positions are 3 and that’s odd so we get a 1 for this.

For R8: bit 8,9,10,11 . We can see that the number of 1’s in these bit positions are 2 and that’s even so we get a 0 for this.

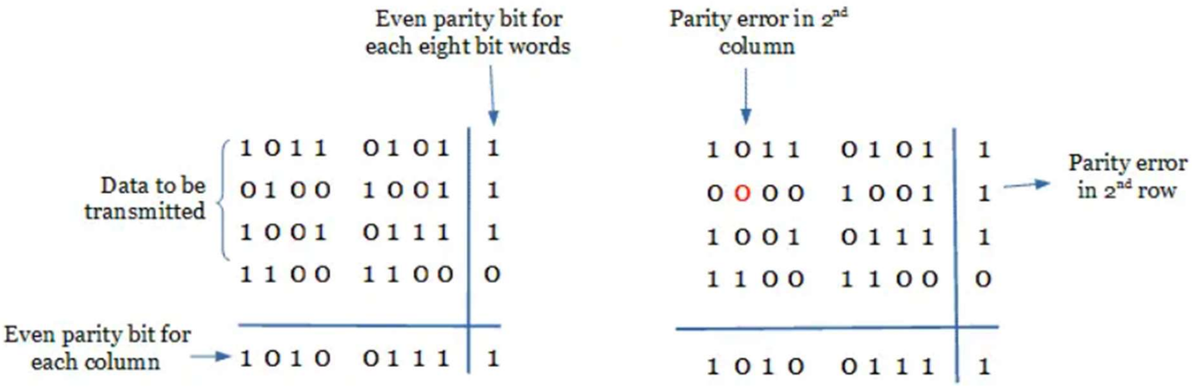
The bits give the binary number 0110 whose decimal representation is 6. Thus, bit 6 contains an error. To correct the error the 6th bit is changed from 1 to 0.

Here are some of the features of Hamming code:

Error Detection and Correction: Hamming code is designed to detect and correct single-bit errors that may occur during the transmission of data. This ensures that the recipient receives the same data that was transmitted by the sender.
Redundancy: Hamming code uses redundant bits to add additional information to the data being transmitted. This redundancy allows the recipient to detect and correct errors that may have occurred during transmission.
Efficiency: Hamming code is a relatively simple and efficient error-correction technique that does not require a lot of computational resources. This makes it ideal for use in low-power and low-bandwidth communication networks.
Widely Used: Hamming code is a widely used error-correction technique and is used in a variety of applications, including telecommunications, computer networks, and data storage systems.
Single Error Correction: Hamming code is capable of correcting a single-bit error, which makes it ideal for use in applications where errors are likely to occur due to external factors such as electromagnetic interference.
Limited Multiple Error Correction: Hamming code can only correct a limited number of multiple errors. In applications where multiple errors are likely to occur, more advanced error-correction techniques may be required.

Block Parity

When several binary words are transmitted and received at a time, then such information is regarded as a block of data, having rows and columns. For example, let us consider four eight-bit words, which are to be transmitted, forms a 4 x 8 block. Parity bits are assigned to both rows and columns.



(a) Transmitted data with block parity

(b) Received data has an error

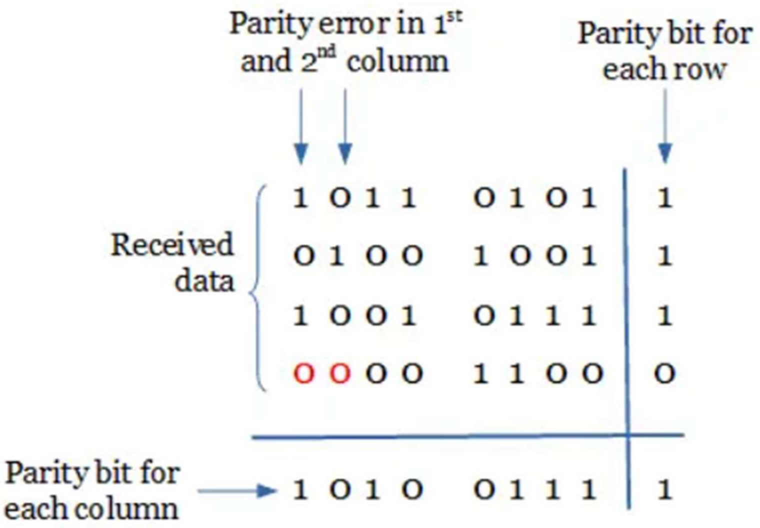
As shown above, even parity is given to each eight-bit word and for each column. In this case, the parity bit is called block parity. The block of data is transmitted from the transmitter side.

Upon analyzing the received block of data(b), the first row has no error as the even parity is maintained. **In the second row, the even parity is changed to odd parity. So there is an error in the 2nd row.** Even parity is maintained in the third and fourth row and thus, there is no error.

To find out the error bit in the second row, let's check the column-wise parity. If you do so, **in the 2nd column, you can notice the even parity is changed to odd parity. So there is an error in the second column.** In all other columns, there is no change in the even parity and hence no error in other columns.

The bit, which is at the intersection of the 2nd row and 2nd column is an error bit. In the above illustration, the error bit is marked in red color. Since the detected error is a single bit, we can change the bit 0 to 1. Thus in block parity, detection and correction of an error are possible with the parity codes.

Now, take a look at the following received message.


Parity error in 1st and 2nd column
Parity bit for each row
Received data
Parity bit for each column

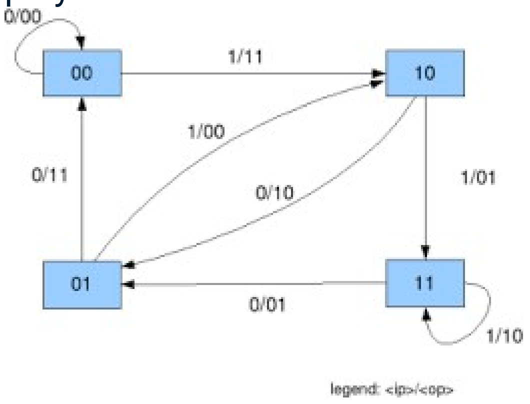
1 0 1 1	0 1 0 1	1
0 1 0 0	1 0 0 1	1
1 0 0 1	0 1 1 1	1
0 0 0 0	1 1 0 0	0
1 0 1 0	0 1 1 1	1

In the above figure, you can observe that there is no parity bit error for each row. But for the 1st and 2nd column the even parity is changed to odd parity, which is an error.

Thus, the error is seen in the 1st and 2nd column. In this case, it is possible only to detect the error. It is not possible to correct the error as there is no information revealing the row where the errors occurred.

Convolution Code

Convolutional coding is a type of error-correcting code that uses memory. It's used in telecommunication to generate parity symbols by applying a boolean polynomial function to a data stream.



Convolutional coding is based on the idea that every coded message must pass through a specific progression of states. The computations and coded output depend on the current set of input symbols and a number of previous input symbols.

Convolutional codes can be used to:

- Detect or correct infinite sequences of errors
- Correct infinite sequences of erasures

In 1973, Viterbi developed an algorithm for maximum likelihood decoding scheme, called Viterbi scheme. This led to modern convolutional codes.