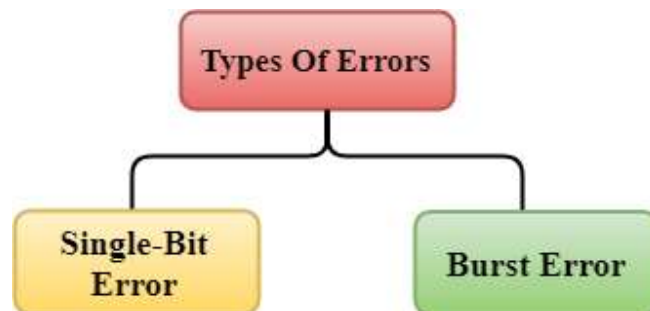# Error Detection

When data is transmitted from one device to another device, the system does not guarantee whether the data received by the device is identical to the data transmitted by another device. An Error is a situation when the message received at the receiver end is not identical to the message transmitted.
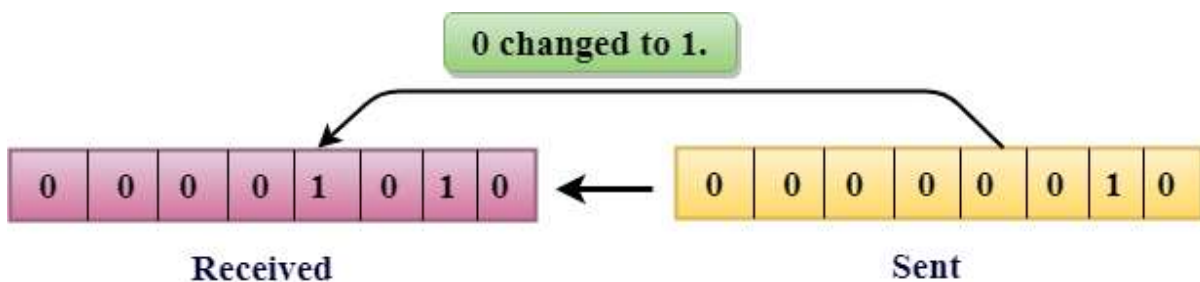
## Types of Errors



Errors can be classified into two categories:

- o Single-Bit Error
- o Burst Error

## Single-Bit Error:

The only one bit of a given data unit is changed from 1 to 0 or from 0 to 1.



In the above figure, the message which is sent is corrupted as single-bit, i.e., 0 bit is changed to 1.

**Single-Bit Error** does not appear more likely in Serial Data Transmission. For example, Sender sends the data at 10 Mbps, this means that the bit lasts only for 1 ?s and for a single-bit error to occurred, a noise must be more than 1 ?s.

Single-Bit Error mainly occurs in Parallel Data Transmission. For example, if eight wires are used to send the eight bits of a byte, if one of the wire is noisy, then single-bit is corrupted per byte.

## Burst Error:

The two or more bits are changed from 0 to 1 or from 1 to 0 is known as Burst Error.

The Burst Error is determined from the first corrupted bit to the last corrupted bit.
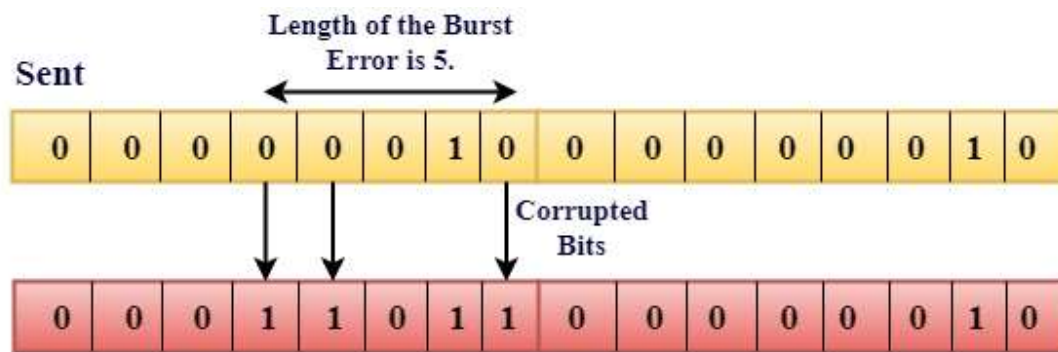


The duration of noise in Burst Error is more than the duration of noise in Single-Bit.

Burst Errors are most likely to occurr in Serial Data Transmission.

The number of affected bits depends on the duration of the noise and data rate.

---

# Error Detecting Techniques:

The most popular Error Detecting Techniques are:

- o   Single parity check
- o   Two-dimensional parity check
- o   Checksum
- o   Cyclic redundancy check
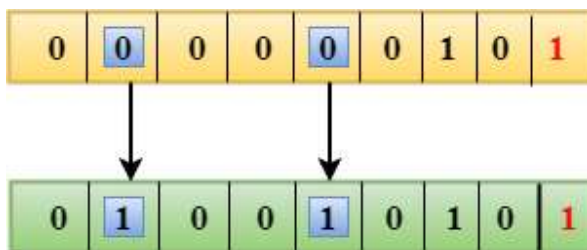
## Single Parity Check

- o   Single Parity checking is the simple mechanism and inexpensive to detect the errors.
- o   In this technique, a redundant bit is also known as a parity bit which is appended at the end of the data unit so that the number of 1s becomes even. Therefore, the total number of transmitted bits would be 9 bits.
- o   If the number of 1s bits is odd, then parity bit 1 is appended and if the number of 1s bits is even, then parity bit 0 is appended at the end of the data unit.
- o   At the receiving end, the parity bit is calculated from the received data bits and compared with the received parity bit.

o   This technique generates the total number of 1s even, so it is known as even-parity checking.
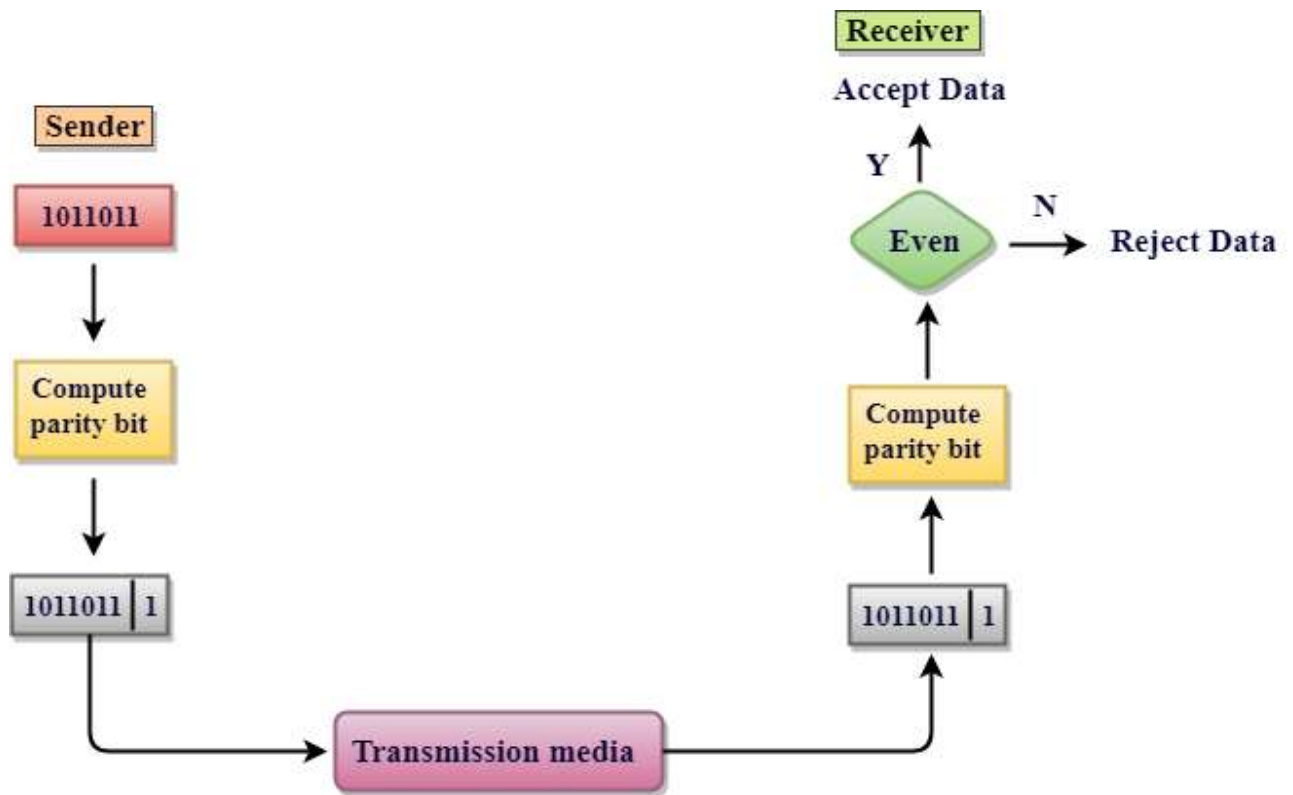


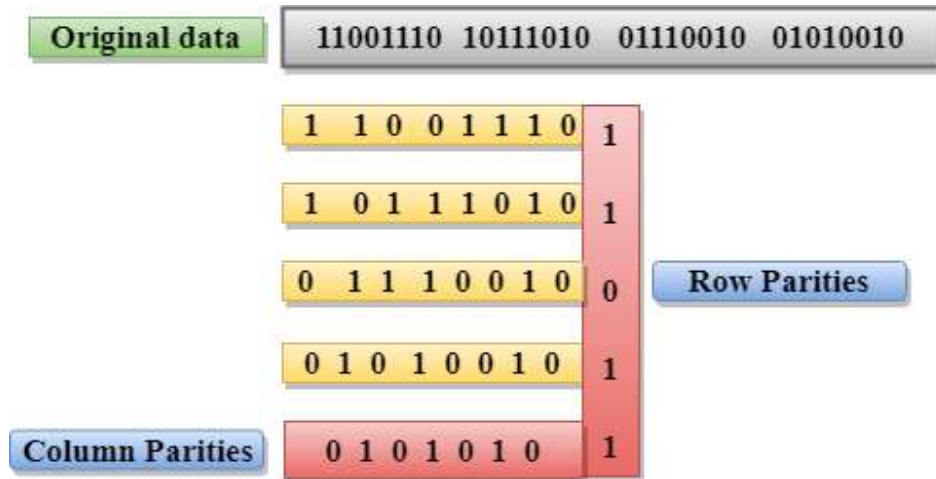## Drawbacks Of Single Parity Checking

o   It can only detect single-bit errors which are very rare.

o   If two bits are interchanged, then it cannot detect the errors.



## Two-Dimensional Parity Check

o   Performance can be improved by using **Two-Dimensional Parity Check** which organizes the data in the form of a table.

o   Parity check bits are computed for each row, which is equivalent to the single-parity check.

o   In Two-Dimensional Parity check, a block of bits is divided into rows, and the redundant row of bits is added to the whole block.

- At the receiving end, the parity bits are compared with the parity bits computed from the received data.

| Original data | 11001110 10111010 01110010 01010010 |

| 1 1 0 0 1 1 1 0 | 1 |
| 1 0 1 1 1 0 1 0 | 1 |
| 0 1 1 1 0 0 1 0 | 0 | Row Parities |
| 0 1 0 1 0 0 1 0 | 1 |

Column Parities | 0 1 0 1 0 1 0 | 1 |

## Drawbacks Of 2D Parity Check

- If two bits in one data unit are corrupted and two bits exactly the same position in another data unit are also corrupted, then 2D Parity checker will not be able to detect the error.
- This technique cannot be used to detect the 4-bit errors or more in some cases.
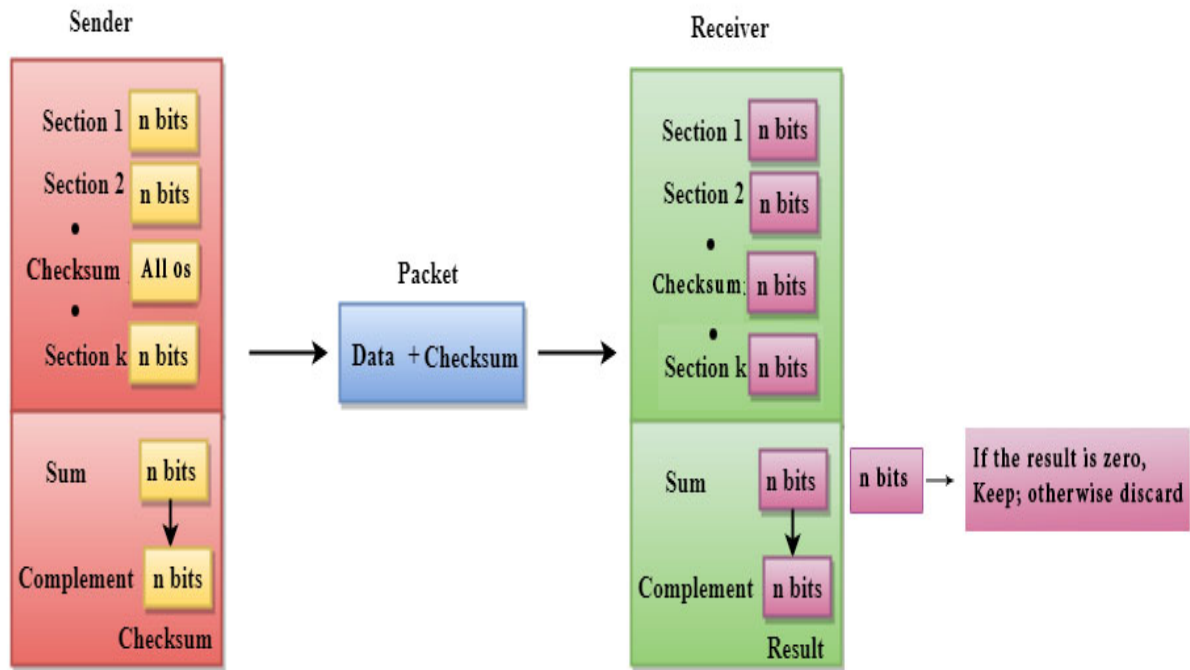
## Checksum

A Checksum is an error detection technique based on the concept of redundancy.

**It is divided into two parts:**

### Checksum Generator

A Checksum is generated at the sending side. Checksum generator subdivides the data into equal segments of n bits each, and all these segments are added together by using one's complement arithmetic. The sum is complemented and appended to the original data, known as checksum field. The extended data is transmitted across the network.

Suppose L is the total sum of the data segments, then the checksum would be ?L

1. The Sender follows the given steps:
2. The block unit is divided into k sections, and each of n bits.
3. All the k sections are added together by using one's complement to get the sum.
4. The sum is complemented and it becomes the checksum field.
5. The original data and checksum field are sent across the network.

### Checksum Checker

A Checksum is verified at the receiving side. The receiver subdivides the incoming data into equal segments of n bits each, and all these segments are added together, and then this sum is complemented. If the complement of the sum is zero, then the data is accepted otherwise data is rejected.

1. The Receiver follows the given steps:
2. The block unit is divided into k sections and each of n bits.
3. All the k sections are added together by using one's complement algorithm to get the sum.
4. The sum is complemented.
5. If the result of the sum is zero, then the data is accepted otherwise the data is discarded.

## Cyclic Redundancy Check (CRC)

CRC is a redundancy error technique used to determine the error.

**Following are the steps used in CRC for error detection:**

- In CRC technique, a string of n 0s is appended to the data unit, and this n number is less than the number of bits in a predetermined number, known as division which is n+1 bits.
- Secondly, the newly extended data is divided by a divisor using a process is known as binary division. The remainder generated from this division is known as CRC remainder.
- Thirdly, the CRC remainder replaces the appended 0s at the end of the original data. This newly generated unit is sent to the receiver.
- The receiver receives the data followed by the CRC remainder. The receiver will treat this whole unit as a single unit, and it is divided by the same divisor that was used to find the CRC remainder.

If the resultant of this division is zero which means that it has no error, and the data is accepted.

If the resultant of this division is not zero which means that the data consists of an error. Therefore, the data is discarded.
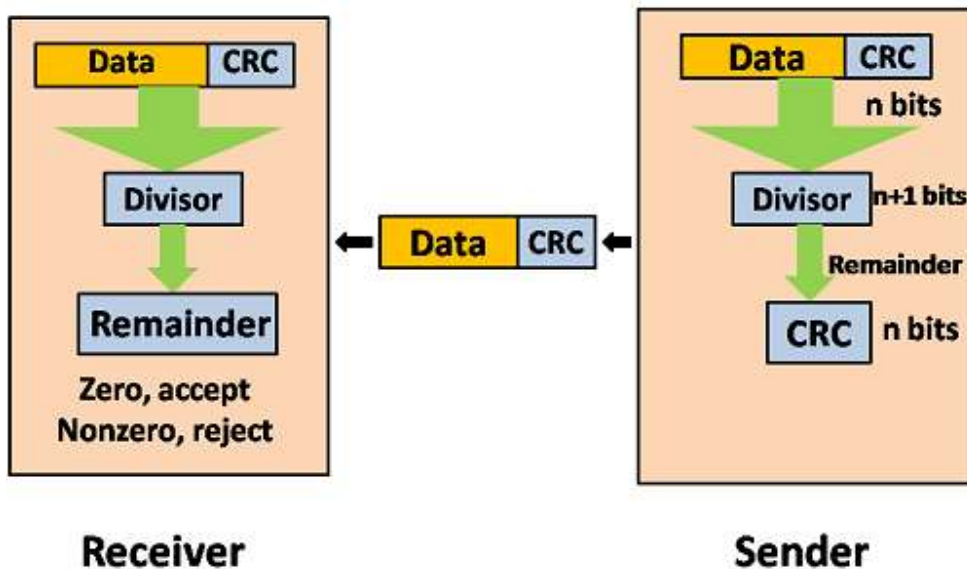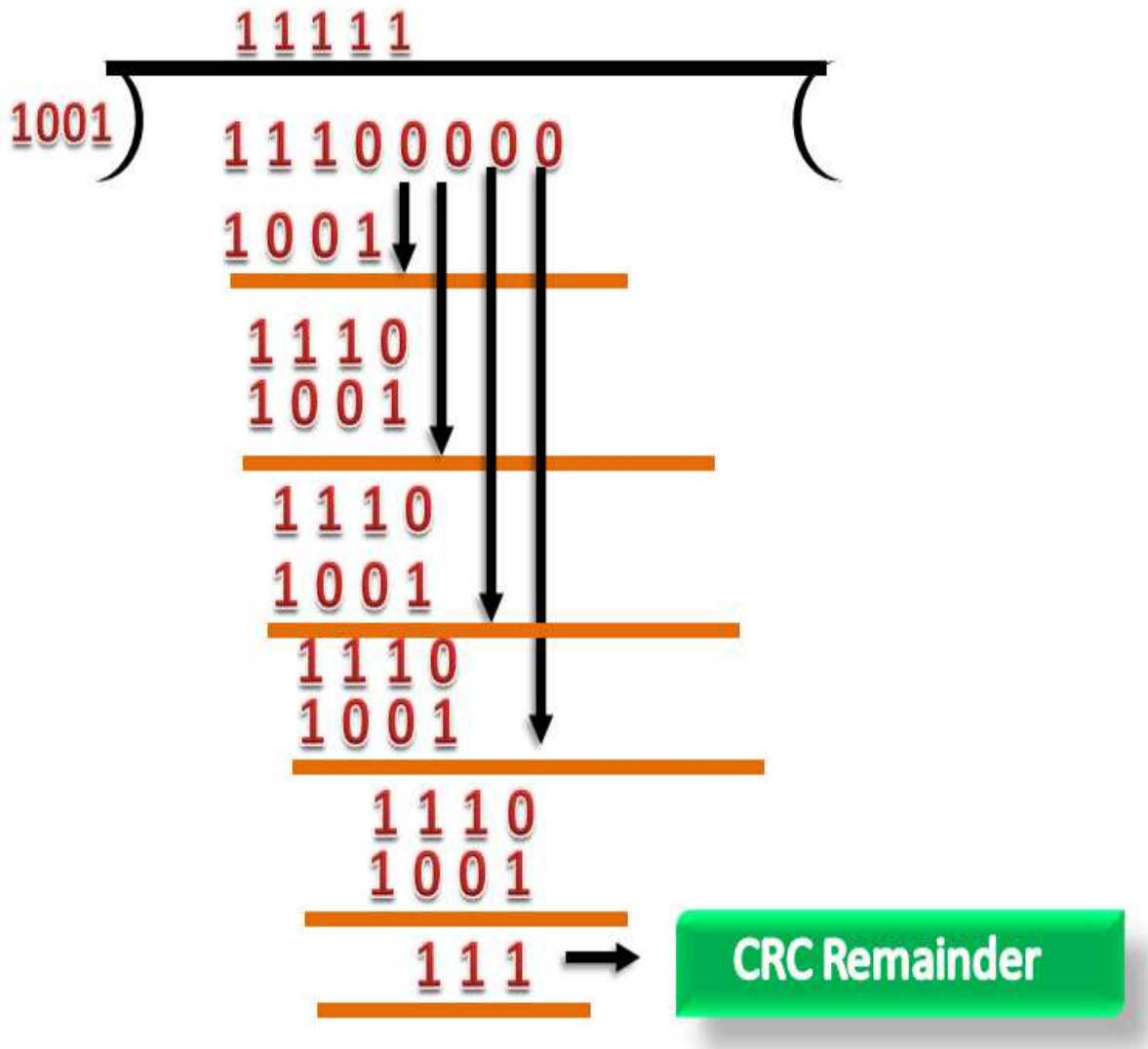


Let's understand this concept through an example:

**Suppose the original data is 11100 and divisor is 1001.**

## CRC Generator

- A CRC generator uses a modulo-2 division. Firstly, three zeroes are appended at the end of the data as the length of the divisor is 4 and we know that the length of the string 0s to be appended is always one less than the length of the divisor.
- Now, the string becomes 11100000, and the resultant string is divided by the divisor 1001.
- The remainder generated from the binary division is known as CRC remainder. The generated value of the CRC remainder is 111.

- o CRC remainder replaces the appended string of 0s at the end of the data unit, and the final string would be 11100111 which are sent across the network.

```
                1 1 1 1 1
         _____
1001 )   1 1 1 0 0 0 0 0
         1 0 0 1
         _____
             1 1 1 0
             1 0 0 1
         _____
                 1 1 1 0
                 1 0 0 1
         _____
                   1 1 1 0
                   1 0 0 1
         _____
                     1 1 1 0
                     1 0 0 1
         _____
                       1 1 1  →  CRC Remainder
         _____
```

## CRC Checker

- o The functionality of the CRC checker is similar to the CRC generator.
- o When the string 11100111 is received at the receiving end, then CRC checker performs the modulo-2 division.
- o A string is divided by the same divisor, i.e., 1001.
- o In this case, CRC checker generates the remainder of zero. Therefore, the data is accepted.

```
        1 1 1 1 1
1 0 0 1 ) 1 1 1 0 0 1 1 1
          1 0 0 1
          ──────────
          1 1 1 0
          1 0 0 1
          ──────────
          1 1 1 1
          1 0 0 1
          ──────────
          1 1 0 1
          1 0 0 1
          ──────────
          1 0 0 1
          1 0 0 1
          ──────────
          0 0 0 0  →  Remainder is 0
          ──────────
```
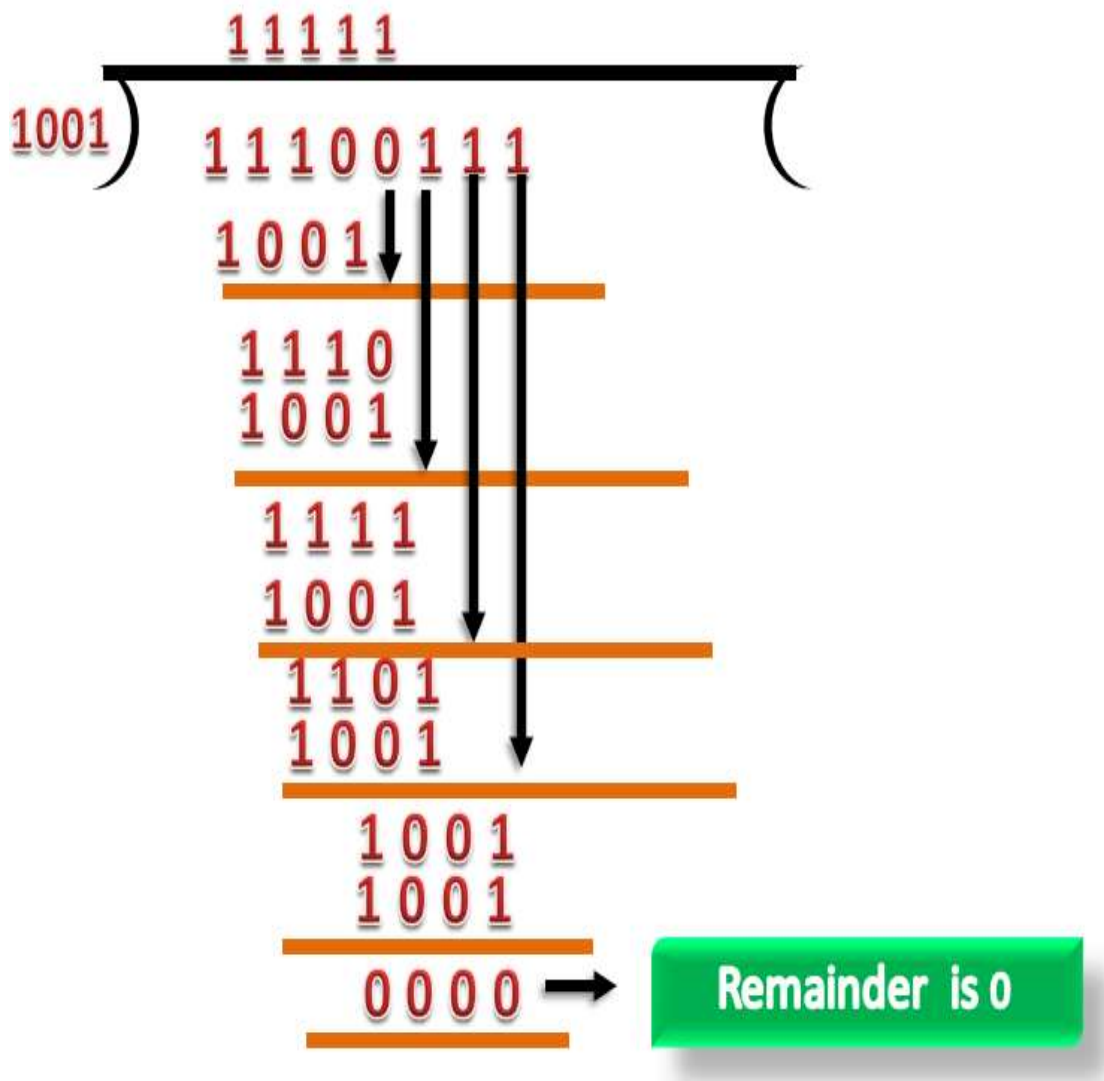
# Error Correction

Error Correction codes are used to detect and correct the errors when data is transmitted from the sender to the receiver.

Error Correction can be handled in two ways:

- o **Backward error correction:** Once the error is discovered, the receiver requests the sender to retransmit the entire data unit.
- o **Forward error correction:** In this case, the receiver uses the error-correcting code which automatically corrects the errors.

A single additional bit can detect the error, but cannot correct it.

For correcting the errors, one has to know the exact position of the error. For example, if we want to calculate a single-bit error, the error correction code will determine which one of seven bits is in error. To achieve this, we have to add some additional redundant bits.

Suppose r is the number of redundant bits and d is the total number of the data bits. The number of redundant bits r can be calculated by using the formula:

```
2ʳ>=d+r+1
```

The value of r is calculated by using the above formula. For example, if the value of d is 4, then the possible smallest value that satisfies the above relation would be 3.

To determine the position of the bit which is in error, a technique developed by R.W Hamming is Hamming code which can be applied to any length of the data unit and uses the relationship between data units and redundant units.

---

# Hamming Code

**Parity bits:** The bit which is appended to the original data of binary bits so that the total number of 1s is even or odd.

**Even parity:** To check for even parity, if the total number of 1s is even, then the value of the parity bit is 0. If the total number of 1s occurrences is odd, then the value of the parity bit is 1.

**Odd Parity:** To check for odd parity, if the total number of 1s is even, then the value of parity bit is 1. If the total number of 1s is odd, then the value of parity bit is 0.

## Algorithm of Hamming code:

- o   An information of 'd' bits are added to the redundant bits 'r' to form d+r.
- o   The location of each of the (d+r) digits is assigned a decimal value.
- o   The 'r' bits are placed in the positions $1,2,.....2^{k-1}$.
- o   At the receiving end, the parity bits are recalculated. The decimal value of the parity bits determines the position of an error.

# Relationship b/w Error position & binary number.

| Error Position | Binary Number |
|:---:|:---:|
| 0 | 000 |
| 1 | 001 |
| 2 | 010 |
| 3 | 011 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |

Let's understand the concept of Hamming code through an example:

Suppose the original data is 1010 which is to be sent.

**Total number of data bits 'd'** = 4

**Number of redundant bits r :** $2^r >= d+r+1$

$$2^r >= 4+r+1$$

Therefore, the value of r is 3 that satisfies the above relation.

**Total number of bits = d+r = 4+3 = 7;**

# Determining the position of the redundant bits

The number of redundant bits is 3. The three bits are represented by r1, r2, r4. The position of the redundant bits is calculated with corresponds to the raised power of 2. Therefore, their corresponding positions are **1, $2^1$, $2^2$**.

1. The position of r1 = 1
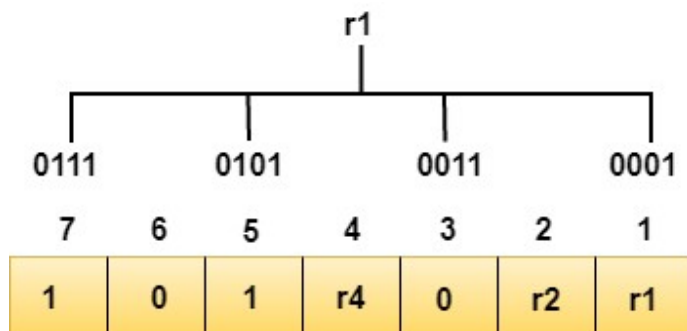2. The position of r2 = 2
3. The position of r4 = 4

Representation of Data on the addition of parity bits:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|:---:|:---:|:---:|:---:|:---:|:---:|:---:|
| 1 | 0 | 1 | r4 | 0 | r2 | r1 |

# Determining the Parity bits
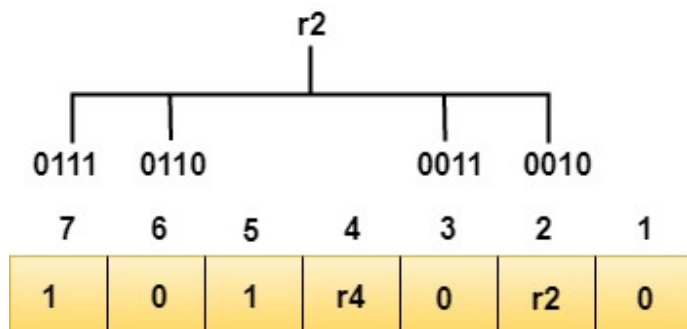
## Determining the r1 bit

The r1 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the first position.

We observe from the above figure that the bit positions that include 1 in the first position are 1, 3, 5, 7. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r1 is **even, therefore, the value of the r1 bit is 0**.

## Determining r2 bit

The r2 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the second position.



We observe from the above figure that the bit positions that include 1 in the second position are **2, 3, 6, 7**. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r2 is **odd, therefore, the value of the r2 bit is 1**.

## Determining r4 bit

The r4 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the third position.
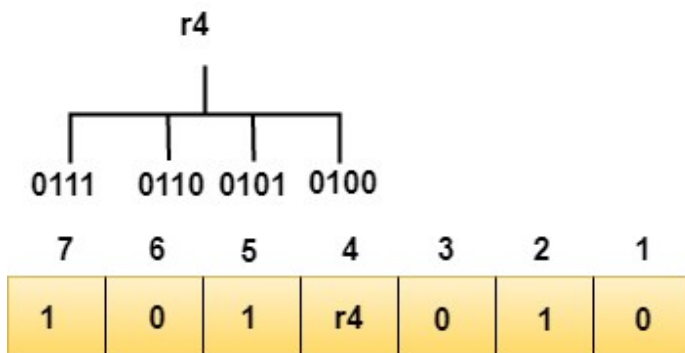
We observe from the above figure that the bit positions that include 1 in the third position are **4, 5, 6, 7**. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r4 is **even, therefore, the value of the r4 bit is 0**.
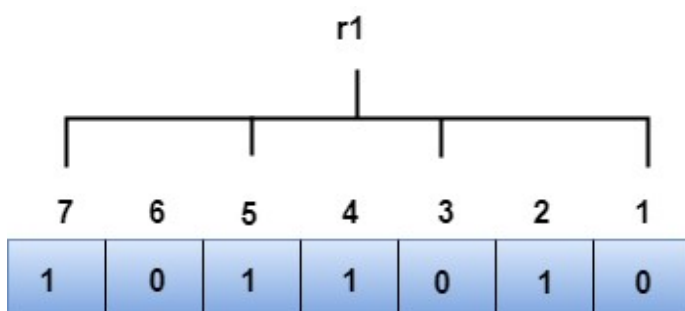
**Data transferred is given below:**



Suppose the 4th bit is changed from 0 to 1 at the receiving end, then parity bits are recalculated.
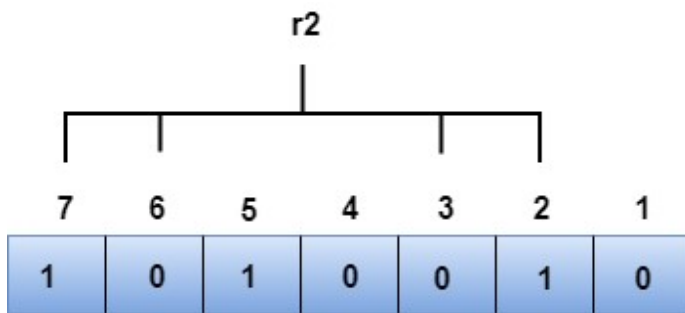
---

# R1 bit

The bit positions of the r1 bit are 1,3,5,7



We observe from the above figure that the binary representation of r1 is 1100. Now, we perform the even-parity check, the total number of 1s appearing in the r1 bit is an even number. Therefore, the value of r1 is 0.
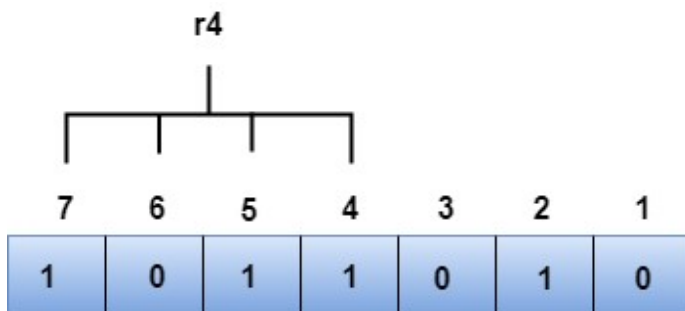
## R2 bit

The bit positions of r2 bit are 2,3,6,7.

We observe from the above figure that the binary representation of r2 is 1001. Now, we perform the even-parity check, the total number of 1s appearing in the r2 bit is an even number. Therefore, the value of r2 is 0.

## R4 bit

The bit positions of r4 bit are 4,5,6,7.



We observe from the above figure that the binary representation of r4 is 1011. Now, we perform the even-parity check, the total number of 1s appearing in the r4 bit is an odd number. Therefore, the value of r4 is 1.

- o *The binary representation of redundant bits, i.e., r4r2r1 is 100, and its corresponding decimal value is 4. Therefore, the error occurs in a 4th bit position. The bit value must be changed from 1 to 0 to correct the error.*

    \#