Error Detection and Error Correction

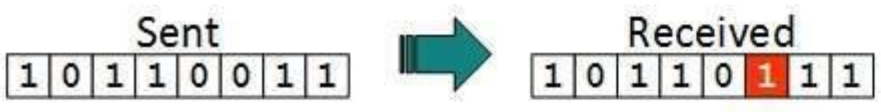
Error Detection:

Error
A condition when the receiver's information does not match with the sender's information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.

Types of Errors

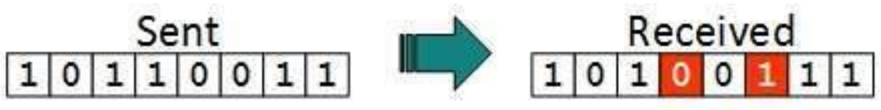
There may be three types of errors:

- **Single bit error**



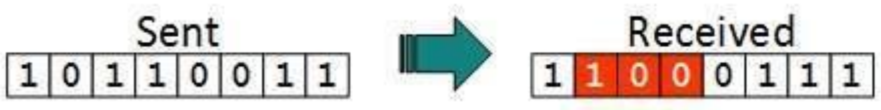
In a frame, there is only one bit, anywhere though, which is corrupt.

- **Multiple bits error**



Frame is received with more than one bits in corrupted state.

- **Burst error**



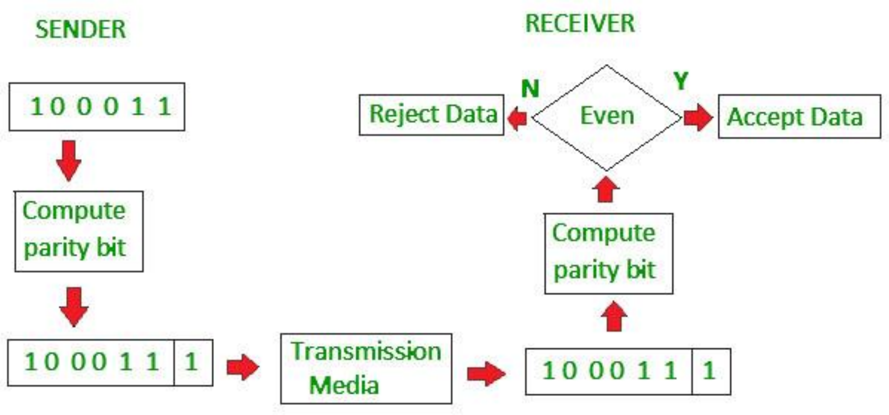
Frame contains more than1 consecutive bits corrupted.

Some popular techniques for error detection are:
1. Simple Parity check
2. Two-dimensional Parity check
3. Checksum
4. Cyclic redundancy check

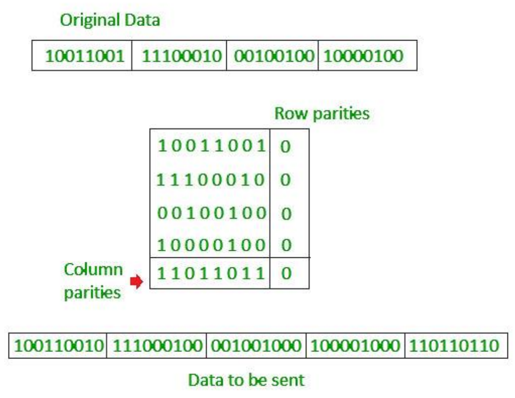
1. Simple Parity check
Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

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

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

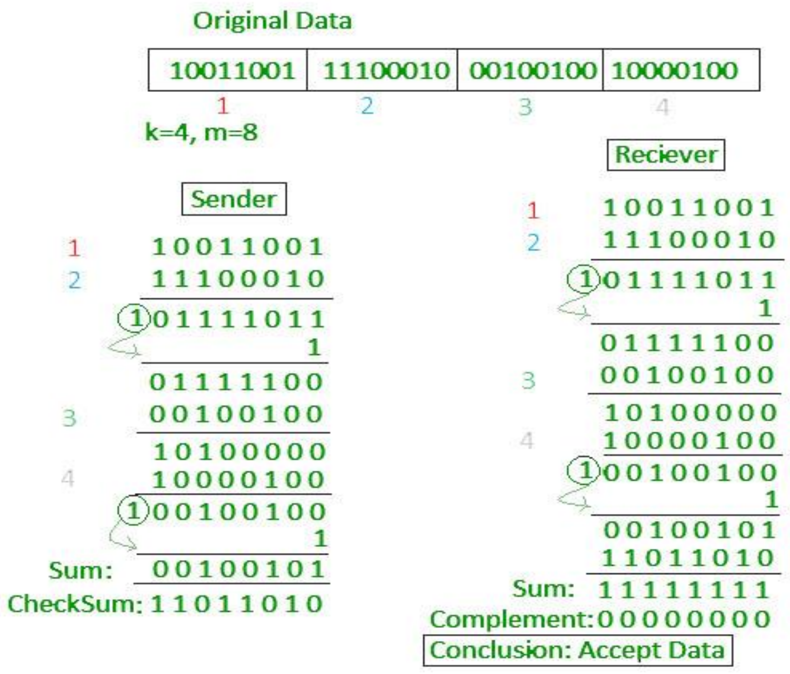


2. Two-dimensional Parity check
Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns, then both are sent along with the data. At the receiving end these are compared with the parity bits calculated on the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



3. Checksum

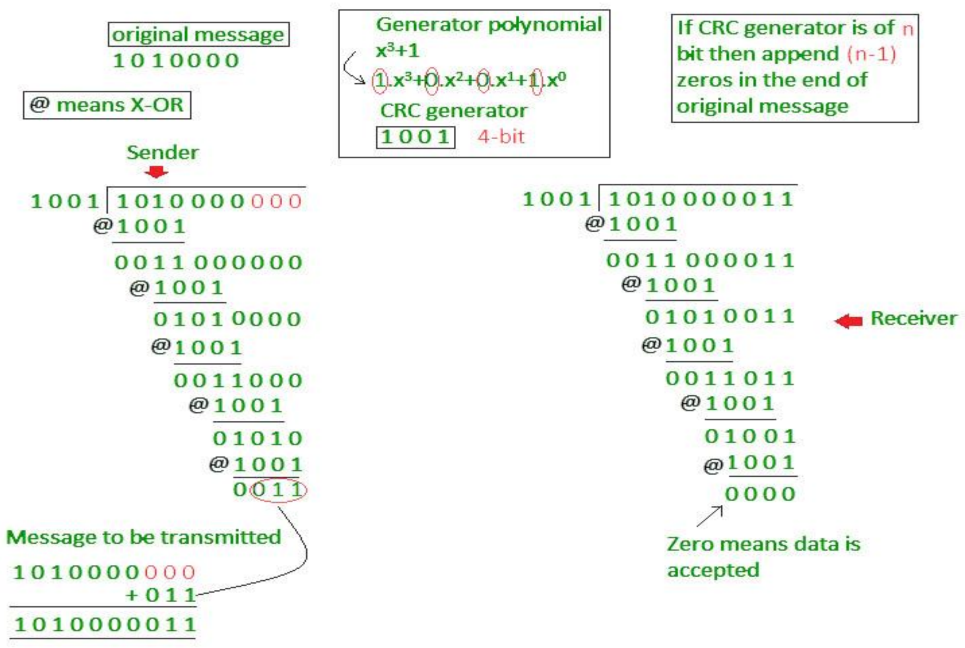
- In checksum error detection scheme, the data is divided into k segments each of m bits.
- In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.
- At the receiver's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



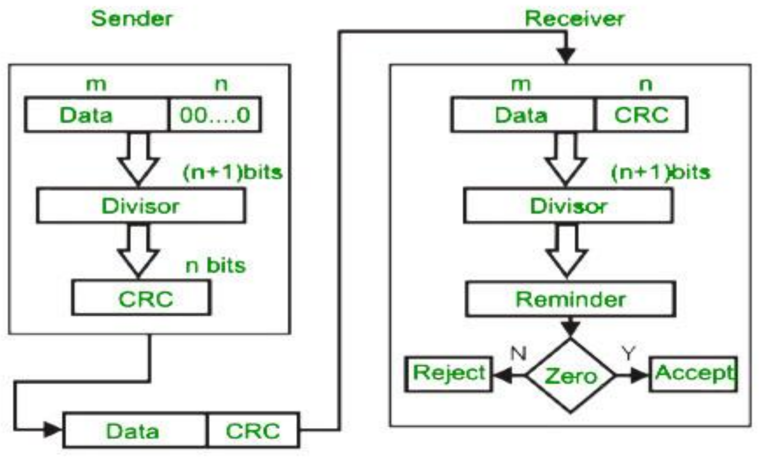
```
Original Data
| 10011001 | 11100010 | 00100100 | 10000100 |
      1          2          3          4
k=4, m=8

        Sender                              Reciever

1       10011001                   1        10011001
2       11100010                   2        11100010
      ①01111011                           ①01111011
               1                                   1
        01111100                            01111100
3       00100100                   3        00100100
        10100000                            10100000
4       10000100                   4        10000100
      ①00100100                           ①00100100
               1                                   1
Sum:    00100101                            00100101
CheckSum: 11011010                          11011010
                                   Sum:     11111111
                                   Complement:00000000
                                   Conclusion: Accept Data
```

4. Cyclic redundancy check (CRC)

- Unlike 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 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.



-



Error Correction Techniques:

Error Correction

In the digital world, error correction can be done in two ways:

- **Backward Error Correction** When the receiver detects an error in the data received, it requests back the sender to retransmit the data unit.
- **Forward Error Correction** When the receiver detects some error in the data received, it executes error-correcting code, which helps it to auto-recover and to correct some kinds of errors.

The first one, Backward Error Correction, is simple and can only be efficiently used where retransmitting is not expensive. For example, fiber optics. But in case of wireless transmission retransmitting may cost too much. In the latter case, Forward Error Correction is used.

To correct the error in data frame, the receiver must know exactly which bit in the frame is corrupted. To locate the bit in error, redundant bits are used as parity bits for error detection.For example, we take ASCII words (7 bits data), then there could be 8 kind of information we need: first seven bits to tell us which bit is error and one more bit to tell that there is no error.

For m data bits, r redundant bits are used. r bits can provide 2r combinations of information. In m+r bit codeword, there is possibility that the r bits themselves may get corrupted. So the number of r bits used must inform about m+r bit locations plus no-error information, i.e. m+r+1.

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

1. Hamming Code:

Parity bits: A bit that is added to the original binary data to make sure the total number of 1s is even or odd (in case of even or odd parity respectively).

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

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

To produce d+r, an information of ‘d’ bits is added to the redundant bits ‘r’.

Each (d+r) digit’s position is assigned a decimal value.

The ‘r’ bits are assigned to locations 1, 2,….2k-1.

The parity bits are recalculated at the receiving end. The position of an error is determined by the decimal value of the parity bits.

Example: If the data to be transmitted is 1011001

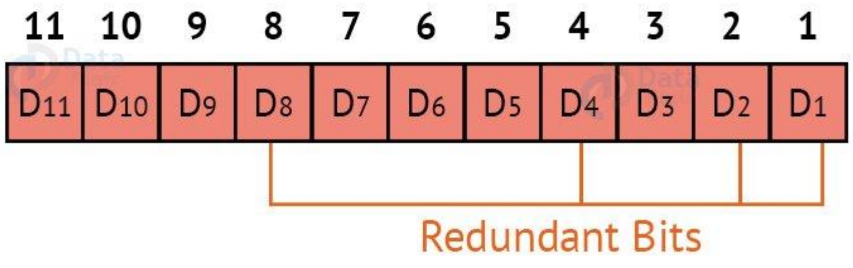
Number of data bits = 7

Thus, number of redundancy bits = 4

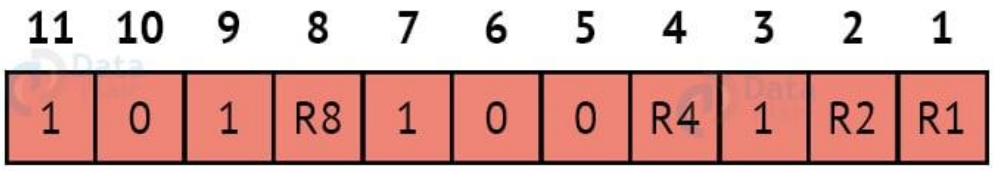
Total bits = 7+4 = 11

Redundant bits are always placed at positions that correspond to the power of 2, so the redundant bits will be placed at positions: 1,2,4 and 8.

Redundant bits will be placed here:



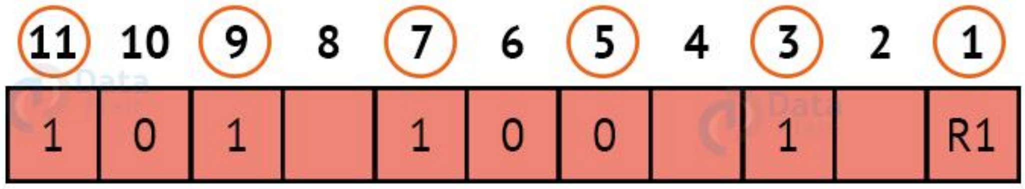
Thus now, all the 11 bits will look like this:



Here, R1, R2, R4 and R8 are the redundant bits.

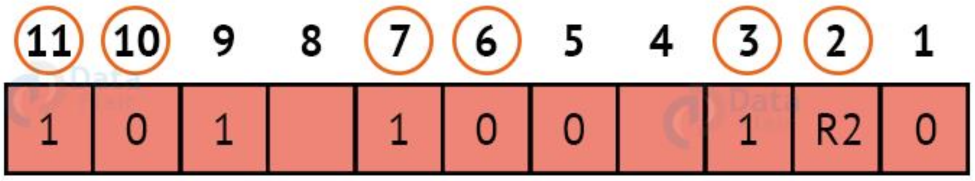
Determining the parity bits:

R1:



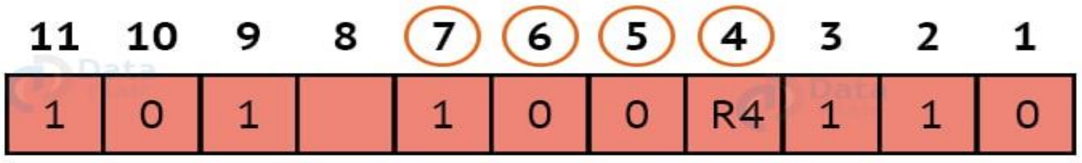
We look at bits 1,3,5,7,9,11 to calculate R1. In this case, because the number of 1s in these bits together is even, we make the R1 bit equal to 0 to maintain even parity.

R2:



We look at bits 2,3,6,7,10,11 to calculate R2. In this case, because the number of 1s in these bits together is odd, we make the R2 bit equal to 1 to maintain even parity.

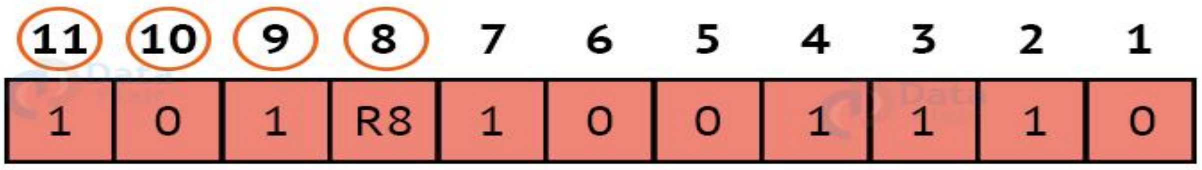
R4:



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

We look at bits 4,5,6,7 to calculate R4. In this case, because the number of 1s in these bits together is odd, we make the R4 bit equal to 1 to maintain even parity.

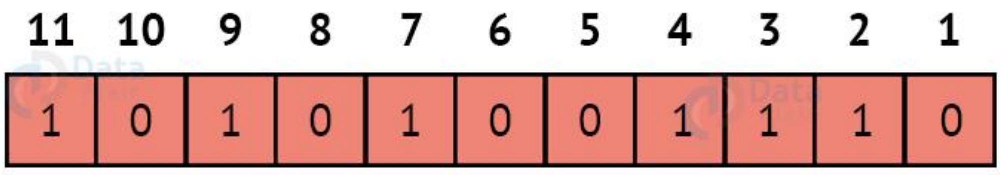
R8:



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

We look at bits 8,9,10,11 to calculate R8. In this case, because the number of 1s in these bits together is even, we make the R8 bit equal to 0 to maintain even parity.

Thus, the final block of data which is transferred looks like this:



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