1.5 ERROR DETECTING AND ERROR CORRECTING CODES

When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits leads to spurious data being received by the receiver and are called errors.

Error-correcting codes (ECC) are a sequence of numbers generated by specific algorithms for detecting and removing errors in data that has been transmitted over noisy channels. Error correcting codes ascertain the exact number of bits that has been corrupted and the location of the corrupted bits, within the limitations in algorithm.

ECCs can be broadly categorized into two types −

- Block codes − The message is divided into fixed-sized blocks of bits, to which redundant bits are added for error detection or correction.
- Convolutional codes − The message comprises of data streams of arbitrary length and parity symbols are generated by the sliding application of a Boolean function to the data stream.

Hamming Code

Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.

In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

Encoding a message by Hamming Code

The procedure used by the sender to encode the message encompasses the following steps –

- Step 1 − Calculation of the number of redundant bits.

- Step 2 − Positioning the redundant bits.
- Step 3 − Calculating the values of each redundant bit.

Once the redundant bits are embedded within the message, this is sent to the user.

Step 1 − Calculation of the number of redundant bits.

If the message contains *mm*number of data bits, *rr*number of redundant bits are added to it so that *mr* is able to indicate at least (*m* + *r*+ 1) different states. Here, (*m* + *r*) indicates location of an error in each of (*m* + *r*) bit positions and one additional state indicates no error. Since, *rr* bits can indicate $2^r r$ states, $2^r r$ must be at least equal to (*m* + *r* + 1). Thus the following equation should hold $2^r \geq m+r+1$

Step 2 − Positioning the redundant bits.

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc. They are referred in the rest of this text as r_1 (at position 1), r_2 (at position 2), r_3 (at position 4), r_4 (at position 8) and so on.

Step 3 − Calculating the values of each redundant bit.

The redundant bits are parity bits. A parity bit is an extra bit that makes the number of 1s either even or odd. The two types of parity are −

- Even Parity − Here the total number of bits in the message is made even.
- Odd Parity − Here the total number of bits in the message is made odd.

Each redundant bit, r_i, is calculated as the parity, generally even parity, based upon its bit position. It covers all bit positions whose binary representation includes a 1 in the i^{th} position except the position of r_i. Thus −

- r_1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)
- r_2 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)

- r_3 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)

Decoding a message in Hamming Code

Once the receiver gets an incoming message, it performs recalculations to detect errors and correct them. The steps for recalculation are −

- Step 1 − Calculation of the number of redundant bits.
- Step 2 − Positioning the redundant bits.
- Step 3 − Parity checking.
- Step 4 − Error detection and correction

Step 1 − Calculation of the number of redundant bits

Using the same formula as in encoding, the number of redundant bits are ascertained.

$2^r \geq m + r + 1$ where *m* is the number of data bits and *r* is the number of redundant bits.

Step 2 − Positioning the redundant bits

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

Step 3 − Parity checking

Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of c_1,c_2 ,c_3 ,c_4 etc. Thus

c_1 = parity(1, 3, 5, 7, 9, 11 and so on)

c_2 = parity(2, 3, 6, 7, 10, 11 and so on)

c_3 = parity(4-7, 12-15, 20-23 and so on)

Step 4 − Error detection and correction

The decimal equivalent of the parity bits binary values is calculated. If it is 0, there is no error. Otherwise, the decimal value gives the bit position which has error. For example, if $c_1c_2c_3c_4$ = 1001, it implies that the data bit at position 9, decimal equivalent of 1001, has error. The bit is flipped to get the correct message.

1.5.1 Parity code

The parity code is used for the purpose of detecting errors during the transmission of binary information. The parity code is a bit that is included with the binary data to be transmitted.

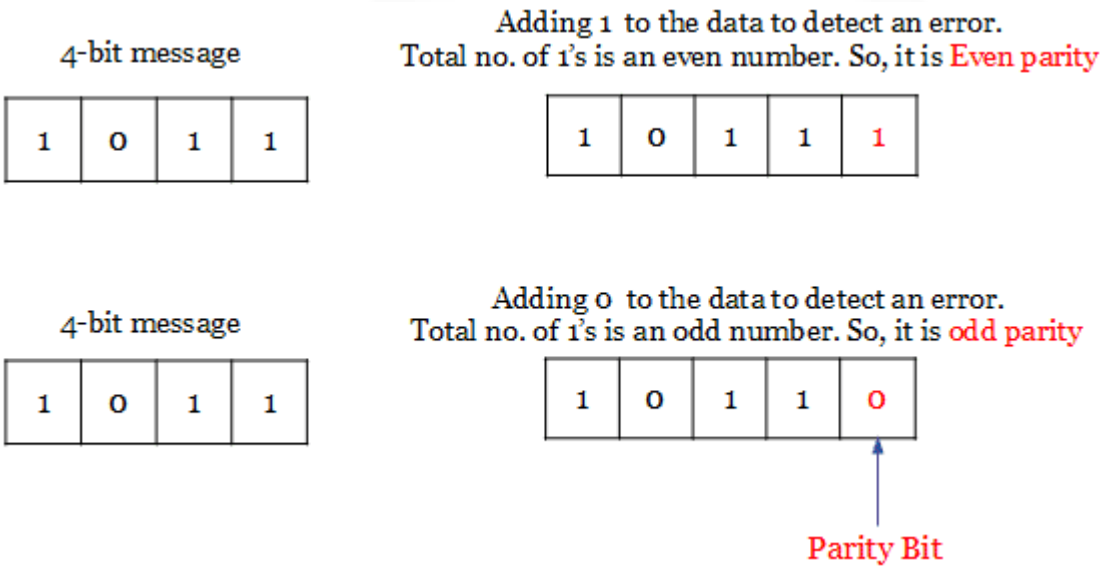
The inclusion of a parity bit will make the number of 1's either odd or even. Based on the number of 1's in the transmitted data, the parity code is of two types.

- Even parity code
- Odd parity code

In even parity, the added parity bit will make the total number of 1's an even number. If the added parity bit make the total number of 1's as odd number, such parity code is said to be odd parity code.

Let us consider the 4-bit message(1011) to be transmitted. Adding 1 to the message will make the total number of 1's in the message to be an even number. Hence it is called as even parity.

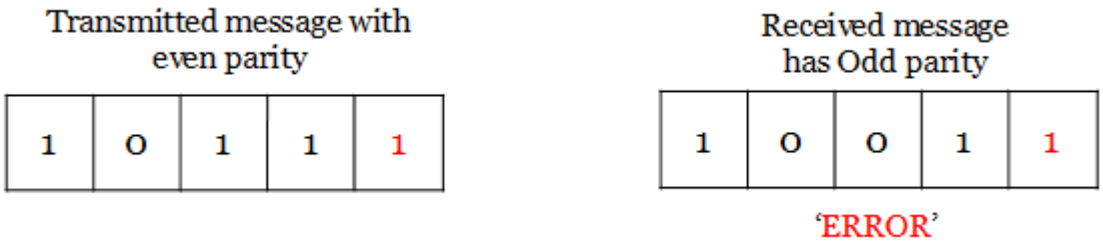
For the same message, adding 0 with the transmitted message will make the total number of 1's to be an odd number. Hence it is called as odd parity. It is shown in the example below.



The following table shows the some of the 4-bit messages to be transmitted along with the parity bits. The bits in red color are the parity bits.

4-bit message	Message with Odd parity	Message with Even Parity
0000	0000**1**	0000**0**
0001	0001**0**	0001**1**
0010	0010**0**	0010**1**
0011	0011**1**	0011**0**
0100	0100**0**	0100**1**
0101	0101**1**	0101**0**
0110	0110**1**	0110**0**
0111	0111**0**	0111**1**

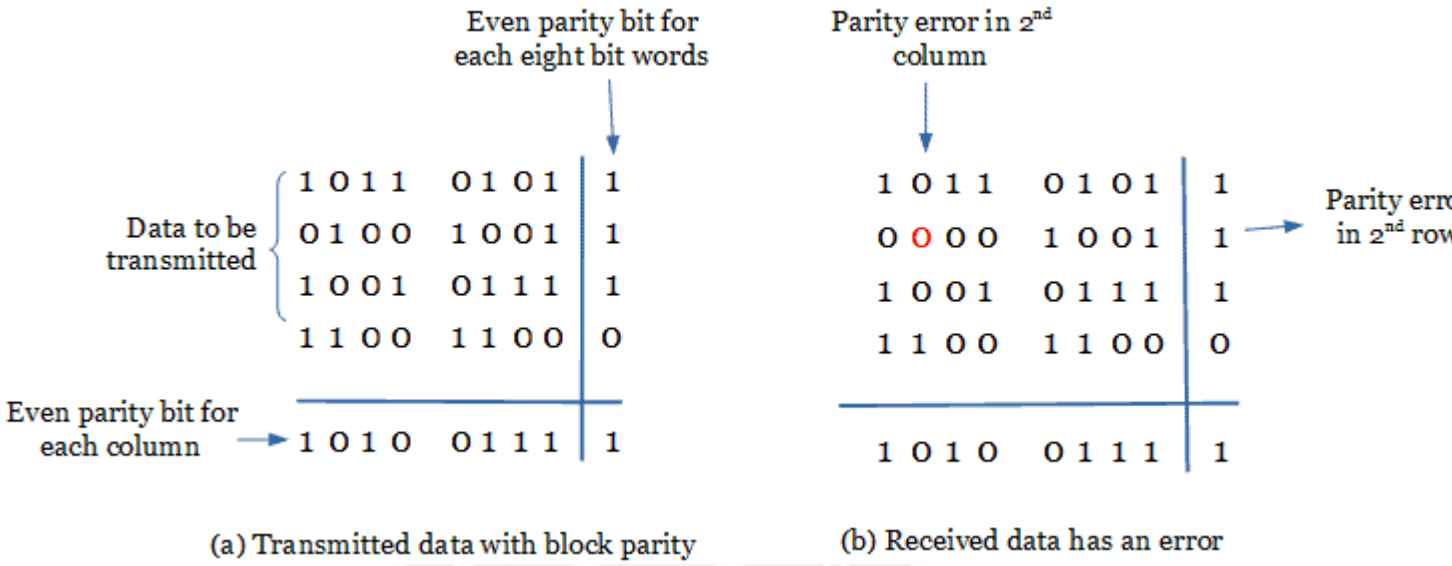
On the receiver side, if the received data is other than the sent data, then it is an error. If the sent date is even parity code and the received data is odd parity, then there is an error.



So, both even and odd parity codes are used only for the detection of error and not for the correction in the transmitted data. Even parity is commonly used and it has almost become a convention.

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

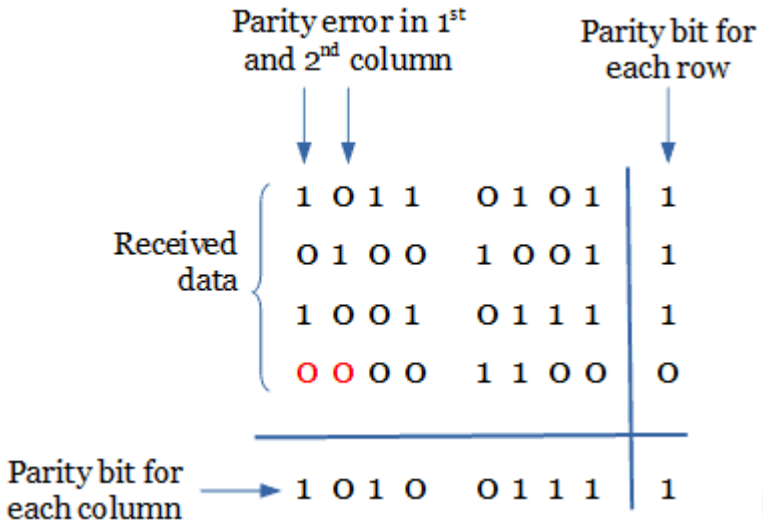
As shown above, even parity is given to each eight-bit word and for each column. In this case, the parity bit is called block parity. The block of data is transmitted from the transmitter side.

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.



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.