SC205 DISCRETE MATHEMATICS

BEAUTY OF ERROR CORRECTION

MATHS IS BEAUTIFUL

Instructor:
Prof. Manish K. Gupta
Prof.Manoj Raut
Prof. Prosenjit Kundu

Group Members:

1. 202201106 - Vraj Dobariya
2. 202201501 - Jish Chanchapara
3. 202201156 - Aman Mangukiya
4. 202201258 - Nishant Italiya
5. 202201041 - Dhriti Goenka
6. 202201162 - Smit Godhani

CONTENTS

1. Introduction:

Welcome to the fascinating world of error-correcting codes! In an increasingly interconnected digital landscape, where data transmission and storage play vital roles, the integrity and accuracy of information become paramount. This is where error-correcting codes come into play. Error-correcting codes are powerful algorithms designed to detect and correct errors that occur during data transmission or storage. By employing various mathematical techniques and encoding schemes, error-correcting codes ensure reliable and error-free communication, safeguarding data integrity and enabling seamless information exchange.
In this PDF we are mainly discussing hamming code which is a very fundamental method for error detecting and error correcting and also discussing various types of error correcting and detecting methods.

2. Where does error occur and How?

Where does the error occur and How? Before this question, we need to understand what is the error. When I was a student probably in 10th std, fortunately, one day for the first time, I heard the word 'ERROR', and our sir said that Error is the thing which is not correct in correctness means it is part of correctness which is incorrect. Error word is from the Latin word "Err" which means to astray or wander which again means away from the correct path or direction.

Error occurs in many places, for example, errors in our lives, in computers, in the production of products, in the transmission of data, CDs, DVDs,etc.

It covers almost every domain. Like a great man said that,

The Greatest Mistake is to Imagine that we never ERR.

>In our project we are going to discuss methods or algorithms which can be used in error detection and corrections in the data transmission or in the data which is already stored in some place for a long time and that data somehow contains error. And when we write data transmission we somehow mean it is a string of 0's and 1's and when transmitted, due to some error it can be very crucial to correct this error as when data is transmitted from space it is very important to make it error-free and cosmic rays tend to change bits of our data.
>To know more about how cosmic rays change our data bits, follow this link: How powerful the one BIT change is.
>The process of identifying and rectifying errors or mistakes in the transmission of data or information is known as error correction. It ensures the integrity of transmitted or stored data and prevents its corruption.
Errors occur due to various factors such as noise, interference, signal degradation, or hardware malfunction.
In the communication part, let's take a quick view of how data is transmitted, and at which stages there are chances of getting an error in our message. Follow this flow chart:

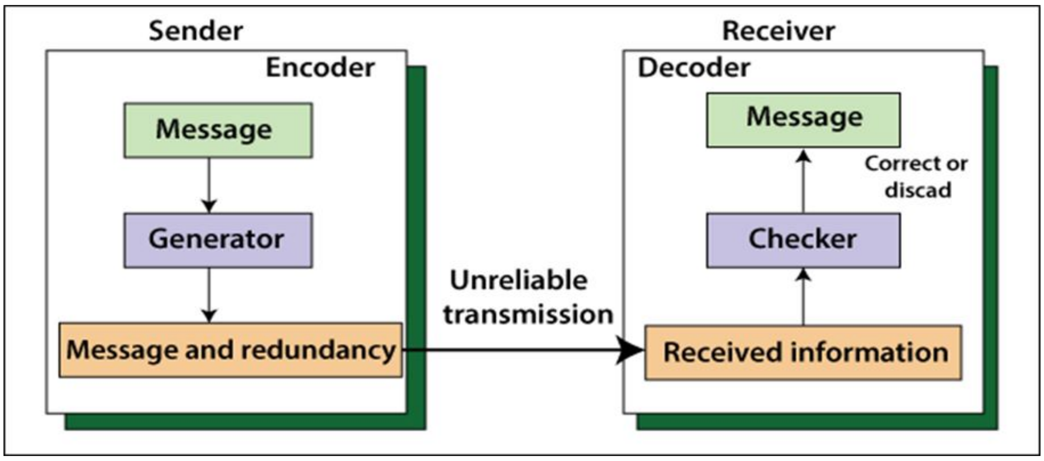


Fig 2.1 TRANSMISSION OF DATA

*** MORE INFORMATION ***

>The Source message is converted into a Binary string using Binary ENCODER that encodes according to particular algorithms that are set in it.
>Then the encoded message will be sent to the channel encoder(MODULATOR) and then it is sent to DEMODULATOR on the other side and DEMODULATOR sends it to DECODER.
> MODULATOR performs modulation, which is the process of imposing digital information onto a carrier signal.
> DEMODULATOR performs the inverse operation of MODULATOR, it attempts to associate channel symbols with noise corrupted received waveform.
>Then at last DECODER Decodes our encoded message according to particular algorithms that are set in it.

3. History of Error Correcting-Codes

The history of error correction dates back to the early days of communication and computing. Here are some key milestones: [4].

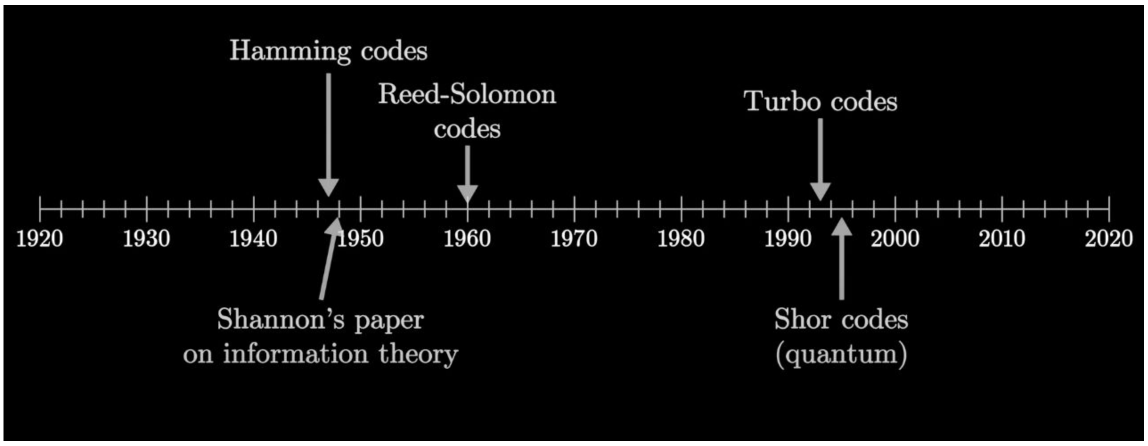


*Fig 3.1 Discovery of error correcting codes

1. Early Telegraphy: `This was the first-ever error-correcting method. The speciality of this method was that its encoding and decoding of messages were performed manually by telegraph operators. Back at that time, skilled telegraph operators were appointed for this error correction work with the help of Morse codes.`

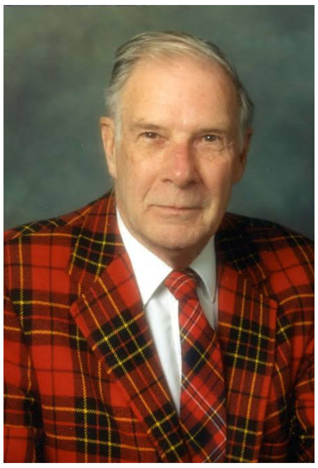


Figure 1: Fig 3.2 Richard Hamming

2. Hamming Codes: Hamming codes were invented by Richard Hamming in the 1950s when he was working in Bell Labs. During the years when he invented this code, digital computers had newly come and for that reason secure digital communication became important. So, if you find an error then it was important to correct that error. Thus, Hamming decided to work on that and finally, he came to the conclusion that we can add parity to the original message.

3.Reed-Solomon Codes: In the 1960s, Irving S. Reed and Gustave Solomon independently developed Reed-Solomon codes. This type of code is used in case of a burst error. Hamming code is not used when multiple errors occur. Thus, Reed-Solomon codes were invented.

Figure 2: Fig 3.3 Reed-Soloman

Even though many error correcting methods are invented, but all of them are interconnected and have some special ability. For example, turbo code is an extended version of Reed Solomon that has a speciality of brilliant error correction capability. After that shore codes came into the market of error detecting and correcting codes. There are some more error-correcting methods like BCH (Bose-Chaudhuri-H and LDPC (Low-Density Parity Check), and many more. But overall every method's motive is always to improve and secure data communication for digital networks.

4. Types of errors and codes:

4.1 Types of Errors:

=>Single Bit Error: **Single-bit errors are the most frequently occurring errors and our project will mainly focus on these errors.**

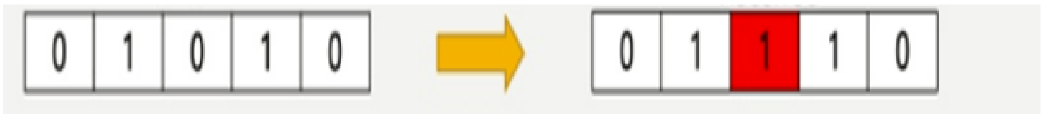


Fig 4.1 Single bit error

=>Burst Error: **Burst error refers to a consecutive sequence of bit errors that occur together in a data stream. Burst errors can be caused by phenomena like fading, interference, or noise bursts in the communication channel. Burst errors particularly require more complex encoding and decoding algorithms.**

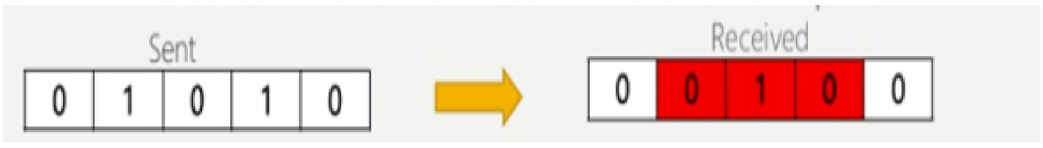


Fig 4.2 Burst error

=>Random Bit Errors: **Random bit errors occurs irregularly and independently throughout the transmitted data stream. These errors can be caused by noise, interference, or other random disturbances in the communication channel.**

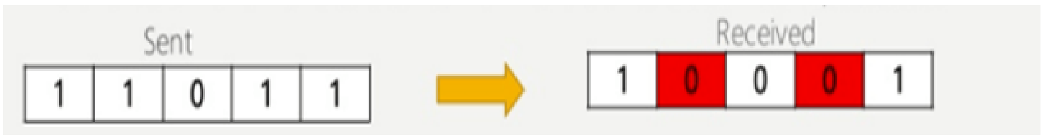


Fig 4.3 Random Bit errors

4.2 Types of Codes:

When we say Types of Codes, we mean the different types of arrangement of the message in which it is sent. Both ENCODER and DECODER should have algorithms with the same type of codes means it is strings of binary or matrix of binary or any other type.

1. Block Codes: Block codes divide the message into fixed-size blocks, and redundant bits are added to each block to provide error detection and correction capabilities. Examples of block codes include Hamming codes, Reed-Solomon codes, and BCH codes.
Below is an example of a Block Code. Encoding is done using parities which we will discuss later.[3]
>:

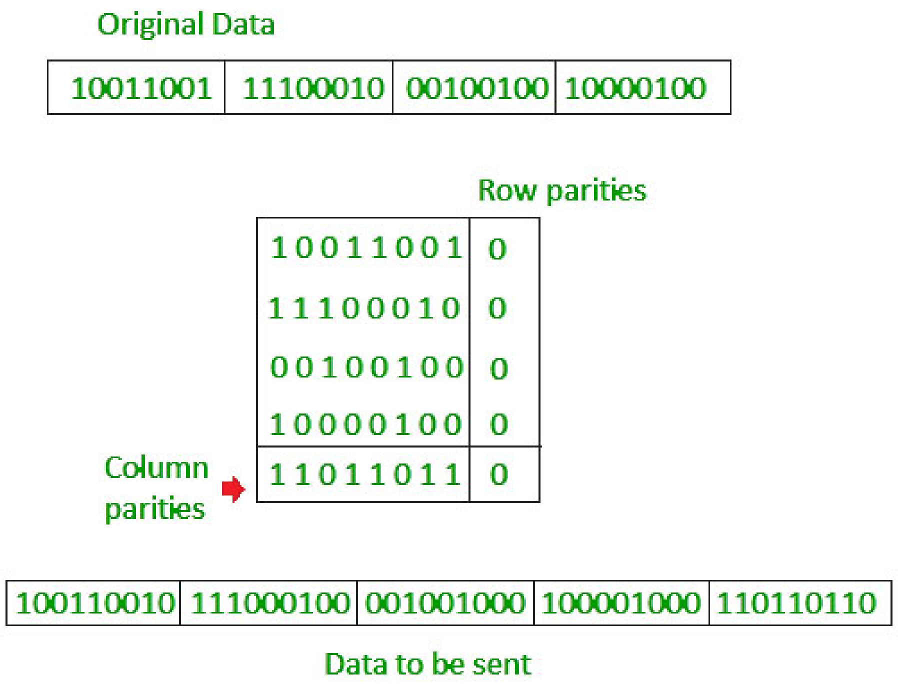


Fig 4.4 Block Codes

4.2.1 Types of Block Codes:

1.Hamming Codes: Hamming Codes are generally used to detect the error which is generated by the flipping of a maximum of two bits. It can correct one-bit errors and detect two-bit errors.

2. BCH Codes: BCH (Bose-Chaudhuri-Hocquenghem) codes are a class of error-correcting codes that can detect and correct multiple errors that are not corrected by hamming codes. They are used in applications such as satellite communications, storage systems, and wireless networks.

3. Turbo Codes: Turbo codes are more powerful error-correcting codes that are widely used in modern communication systems, such as 4G and 5G mobile networks. They provide excellent error correction performance and are based on the concept of iterative decoding. They also include the tree code's concept because it is an updated version for error correction.

2.Tree Codes: The major difference between block codes and tree codes is that tree codes don't take the block of code as an input for error correction and detection but take the data stream as the input. So, it is a very practical method for wireless communication systems or satellite transmission. This type of code is also known as Convolutional code.[1]

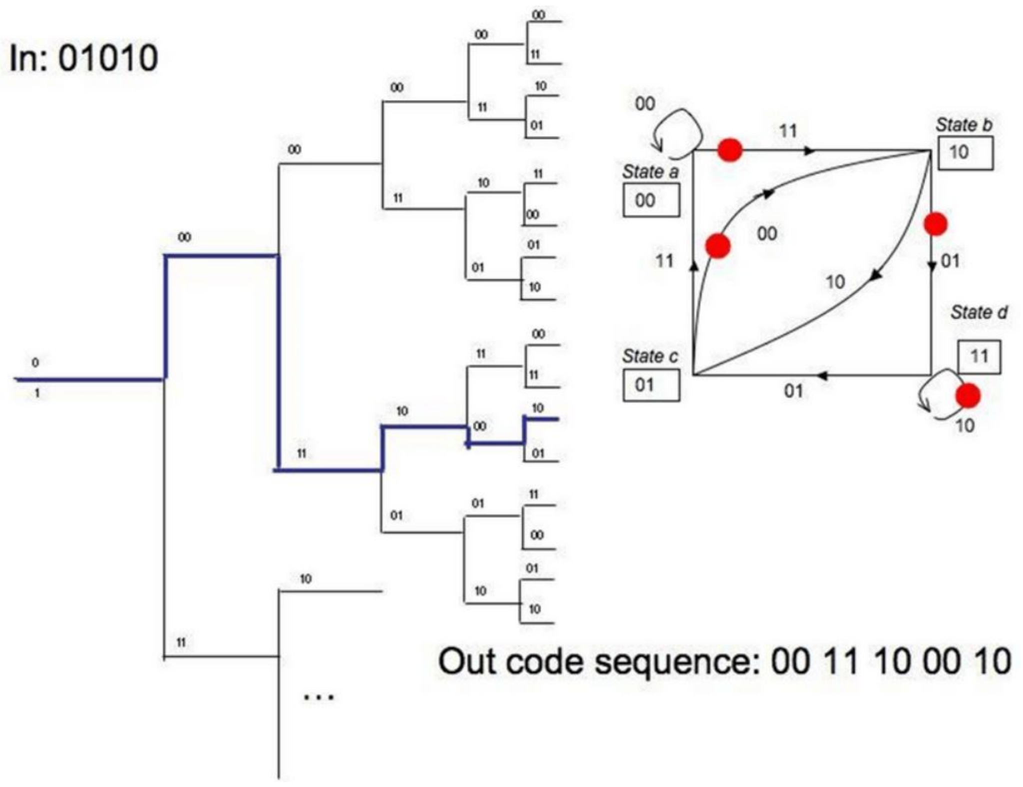


Fig 4.3 Convolution graphs (Tree codes)

4.2.2 Types of Tree Codes:

1. Convolutional codes: This method represents the tree code's function because this code also works on a stream of codes. Convolutional codes use shift registers to encode data, and the encoder transitions through different states as new data arrives. They are characterized by parameters like the constraint length and the generator polynomial.

2. Decoding of Turbo Codes: Turbo codes are more powerful in the decoding process and tree code's "turbo decoding algorithm" is used in it. The algorithm is included by an iterative decoding process that includes tree codes.

3. Low-Density Parity-Check This method is very simple to understand and it has minimal complexity at the decoding part. This makes this method more powerful. These codes are based on bipartite graphs and these graphs involve tree codes. Thus, it uses tree codes to make a powerful error correction machine.

There are many more error-correcting codes than these and we can also create our error-creating methods but "It should be optimal and more generalise so that all machines can apply that thing".

5. ERROR-DETECTING CODES:

5.1 Simple Parity Checks:

The easiest technique to encode a binary message to make it error-detecting is to count the number of 1s in the message and then append the last binary digit selected such that the entire message has an even number of 1s. As a result, the entire message is parity. Thus, we append an nth parity-check position to (n - 1) message locations. At the receiving end, the number of 1s is counted, and an odd number of 1s in the entire n places indicates that at least one mistake has happened.

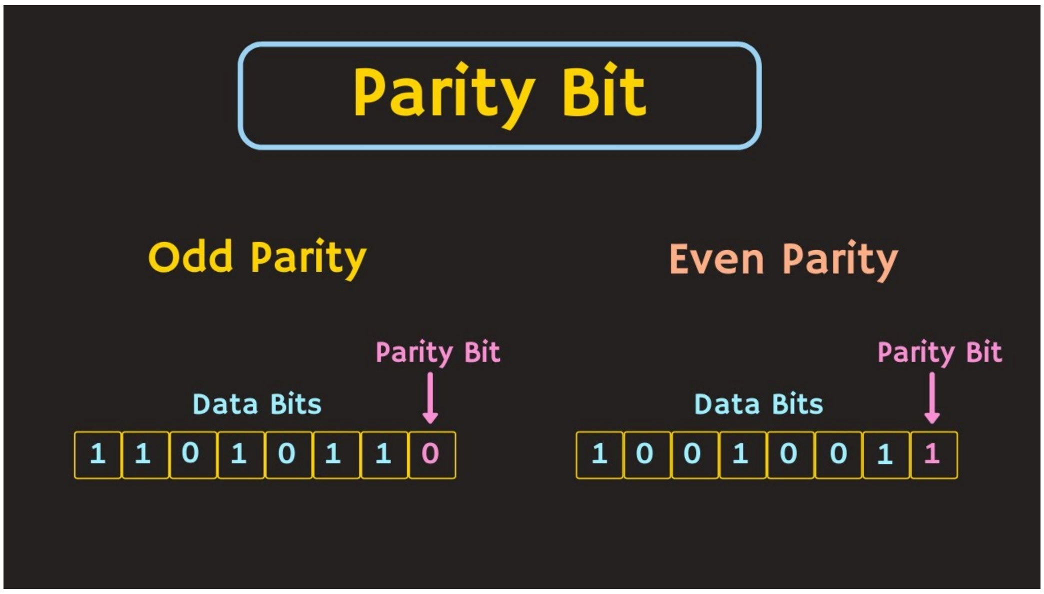
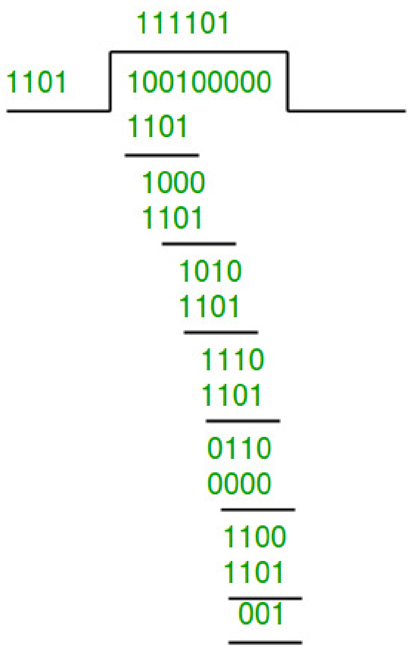


Fig 5.1 Simple Parity Checks

5.2 CRCS:

CRC stands for Cyclic Redundancy Check and is frequently used in communication systems such as Ethernet. A checksum is generated using a polynomial division technique and attached to the data in CRC. Using the incoming data and the generating polynomial, the receiving end does the same polynomial division. An error is discovered if the remainder is non-zero.



>In the term polynomial, we mean that the power of that term is the position of the bit that string has.
>Here we will choose a generator polynomial of any of your choice(not degree 0)
>Then we will multiply that with our message polynomial and that message will be sent to the receiver side.
>There decoder will have that same(obviously) generator polynomial and then the decoder will divide the received polynomial by the generator polynomial.
>After that it will check the remainder. If the remainder is not zero, then an error must have occurred.

[5]

REMEMBER, THESE CODES ONLY CHECK IF THERE IS AN ERROR OR NOT (DETECTION). THEY DO NOT CORRECT IT. FOR CORRECTING THE ERRORS, WE HAVE ERROR-CORRECTING CODES THAT PERFORM BOTH DETECTION AND CORRECTION OF ERROR

6. ERROR CORRECTING-CODES:

6.1 Triplication Codes:

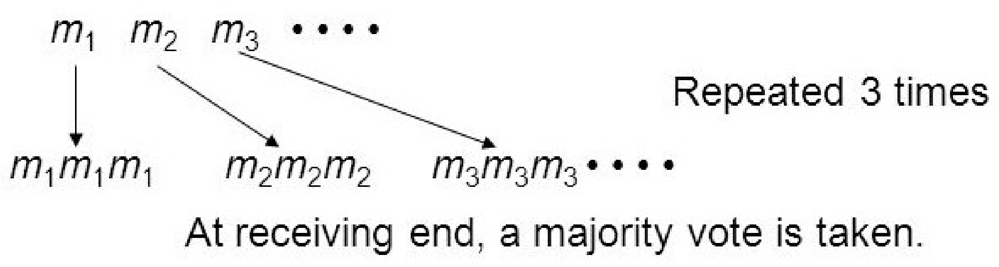

m_1 m_2 m_3 • • • •
Repeated 3 times
$m_1m_1m_1$ $m_2m_2m_2$ $m_3m_3m_3$ • • • •
At receiving end, a majority vote is taken.

Fig 6.1 Triplication codes for single error

-> Triplication codes work as their name suggests. It triplicates every digit 3 times and then sends. Then at the receiver end, the DECODER checks 3 digits at a time and counts. If the maximum is one then the message sent was one, otherwise, zero. Thus, this code works only if we are sure that there will be no more than one error and for more errors we need to add more replications. So, to check one error we need to add more digits and not a single digit. We need to add 2 digits every time we increase the number of errors. But in even digits, we will not be able to tell if an error is there or not.

->The drawback is that we are wasting our memory space by almost 66% and the program is only 33% memory efficient.

->Although it has a nice and simple algorithm and for more bits memory efficiency increases, but it increases by a very small amount. Thus, it has a drawback.

AT LEAST THIS MUCH ERRORS	MEMORY OCCUPIED	MEMORY WASTED
1	3 bits	66%
2	5 bits	60%
3	7 bits	57%
5	11 bits	54%
10	21 bits	52%
Infinity n->infinity	2*n + 1 bits	approx 50 %

Fig 6.2 Wastage of memory in Triplication codes

6.2 Rectangular Codes:

Another part of this theory is rectangular codes in which our message is put in a rectangle, 2D-array in which n-1 rows and m-1 columns are there and nth and mth column is encoded with parity bits according to their rows and columns.[2]

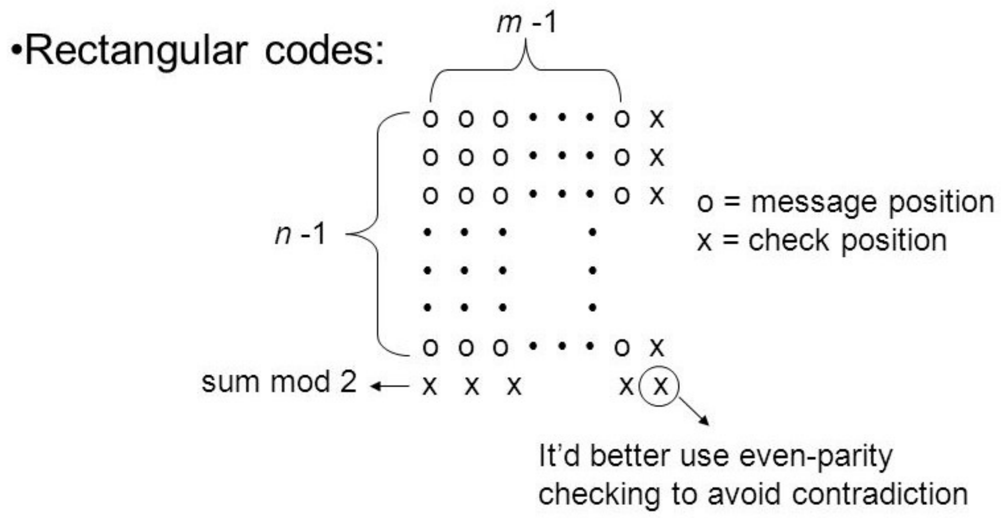


Fig 6.3 Rectangular Codes

7. HAMMING CODES:

-> Hamming code was invented by Richard W. Hamming in 1950 for error correction. There are two types of coding: (A) linear coding and (b) non-linear coding. Hamming code belongs to linear coding. Linear codes do not have a very complex encoding and decoding system but in non-linear coding system, the encoding and decoding is very complex.

-> Hamming codes are useful in single or double-bit errors only. They correct only one-bit errors and detect up to two-bit errors. It uses basic parities to do the same. It includes encoding and decoding, two main functions to secure or correct the data.[2]

1. Linear Hamming Method::

7.1 Basic Notation:

The hamming code is generally represented as (n,k) code; where n is the number of total bits that include parity and data bits and k is the number of data(information) bits.
Relation between n and k is ::

$$\boxed{2^k >= n+k+1}$$

By that formula, we can calculate the number of parity bits required for a fixed number of bits. For example, for a (7,4) code we need 3 parity bits and for a (11,7) code we need 4 parity bits. These numbers can be verified by the above formula.

7.2 Encoding of Hamming codes :

The hamming code's linear encoding method is very simple. We can divide this process into three steps,
Step 1: Calculation of the number of parity bits.
Step 2: Position of the parity bits.

Step 3: Calculating the value of each parity bit.

Step-1:
In the above formula of n and k, we can calculate how many parity bits you have to choose by r = n-k. For example (7,4) there are three parity bits. So, if you have 4 bits of information code then you have to attach 3 more bits (parity bit) at a particular place so it can correct the error.

Step-2:-
In this section we arrange the parity bit at some decided place such that at the time of the decoding process, we do some mathematical operations and find error bits. Hamming did all arrangements and then he concluded by saying that all parity bits will be placed at the power of two, like 1,2,4,8.....and more. For example

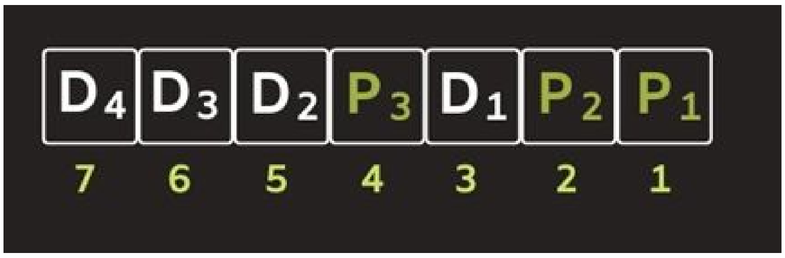


Fig 7.1 Positioning of parity bits (7,4)

Just like shown in the image, we have to start calculating the position of any message from the end. and parity will always be placed at position like 1,2,4,8,16.

Step-3:
Now, we decide the value of parity according to parity types. We have two types of parity: 1) Even parity and 2)Odd parity. First, you have to count the number of 1s in the binary code and then you can apply any of the two, odd parity or even parity. In even parity, if the number of 1s is odd, the parity bit is 1, and if the number of 1s if even, the parity bit is 0. The odd parity method works vice-versa. We generally use the even parity method in hamming code.
Now, to decide which information bit to use for a particular parity bit, we follow

a simple rule. Each parity bit covers all bit positions whose binary representation includes a 1 in the i_{th} position except the position of r_i.
For example, for a (7,4) code the parity bits are calculated like::

$$
\begin{aligned}
P1 &= D7 \oplus D5 \oplus D3 \\
P2 &= D7 \oplus D6 \oplus D3 \\
P4 &= D7 \oplus D6 \oplus D5
\end{aligned}
$$

Fig 7.2 Formula for parity value of (7,4) code

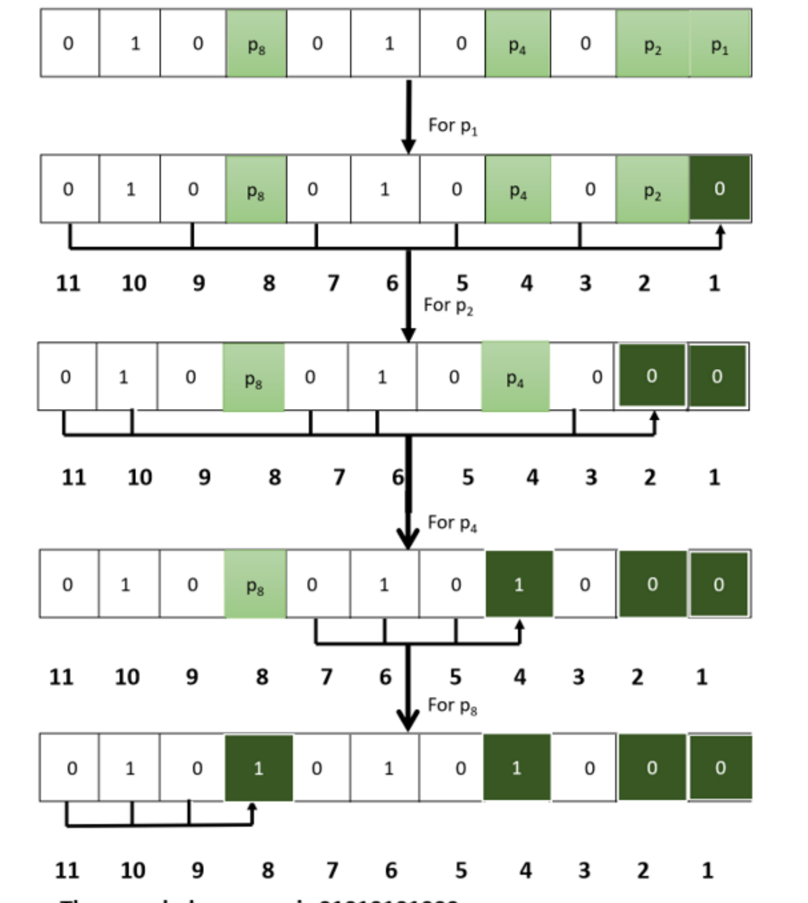
Let's take one example

We have a message of length seven(7). So, now we have a (11,7) hamming code that has information "0100100". The encoded message will also include 4 parity bits; making its total length 11.
To get parity values we use the formula::

$$
\begin{aligned}
P1 &= D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 \\
p2 &= D3 \oplus D6 \oplus D10 \oplus D11 \oplus D7 \\
p3 &= D5 \oplus D6 \oplus D7 \\
p4 &= D9 \oplus D10 \oplus D11
\end{aligned}
$$

Fig 7.3 Formula for parity value of (11,7) code

Now, we can follow the below steps for encoding



The encoded message is 01010101000.

Fig 7.4 Encoding process

The above image explains to you about each and every EXOR operation that happened for each parity.
After that we can say that the final encoded message is :

0	1	0	1	0	1	0	1	0	0	0

Fig 7.5 Final Encoded Message

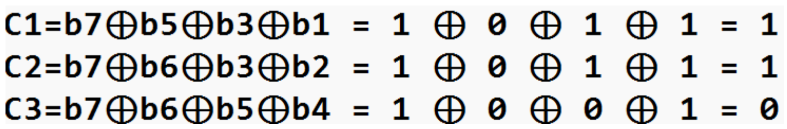
7.3 Decoding of hamming codes

In the decoding, you have to do two main steps: first- identify if there is any error or not, and second locate the error-bit and correct it. Hamming code can maximum detect a maximum of two errors and correct one error. Thus, with the use of parity, we can find one error. Finding the check bits follows the same mathematical algorithm as finding the parity bits. You can see the check bits for a (7,4) hamming code in the below image:

$$
\begin{aligned}
C1 &= D7 \oplus D5 \oplus D3 \oplus D1 \\
C2 &= D7 \oplus D6 \oplus D3 \oplus D2 \\
C3 &= D7 \oplus D6 \oplus D5 \oplus D4
\end{aligned}
$$

Fig 7.6 Formula for decoding

From the above expressions you can check whether there is any error or not. If there is an error, the check bits give us the index of error and if no error is present the result is 0. For example
Our encoded code is "1001111" so for this::



```
C1=b7⊕b5⊕b3⊕b1 = 1 ⊕ 0 ⊕ 1 ⊕ 1 = 1
C2=b7⊕b6⊕b3⊕b2 = 1 ⊕ 0 ⊕ 1 ⊕ 1 = 1
C3=b7⊕b6⊕b5⊕b4 = 1 ⊕ 0 ⊕ 0 ⊕ 1 = 0
```

Fig 7.7 "110" = 6th position from back

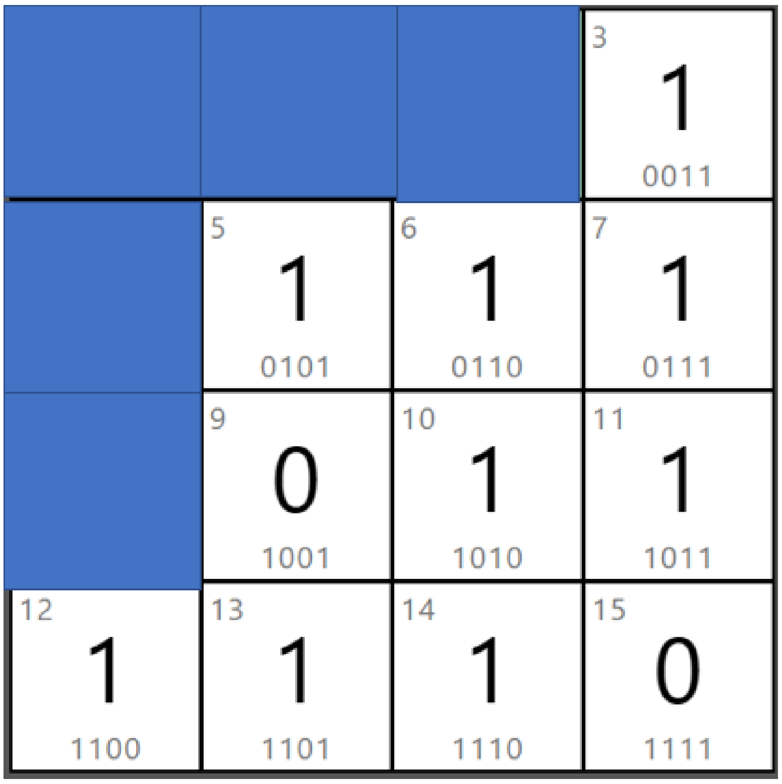
Thus, from this we can say that "110" is the place where the error has occurred. Therefore, the actual code is "1101111" which was encoded by the machine. That's how data decoding works and it is very useful for the detection and correction of single-bit error.

2. Hamming code with the help of table(matrix) ::

This method is only a 2-D version of linear hamming code. Its basic function is similar to the linear codes. Follow the below pictures carefully and you will understand the process:

Binary: 11110111110

In this type of code we have to calculate parity first. So, we are arranging the information bits in a 4x4 matrix in such a way that they do not occupy the positions for parity bits(power of two). After arranging ::

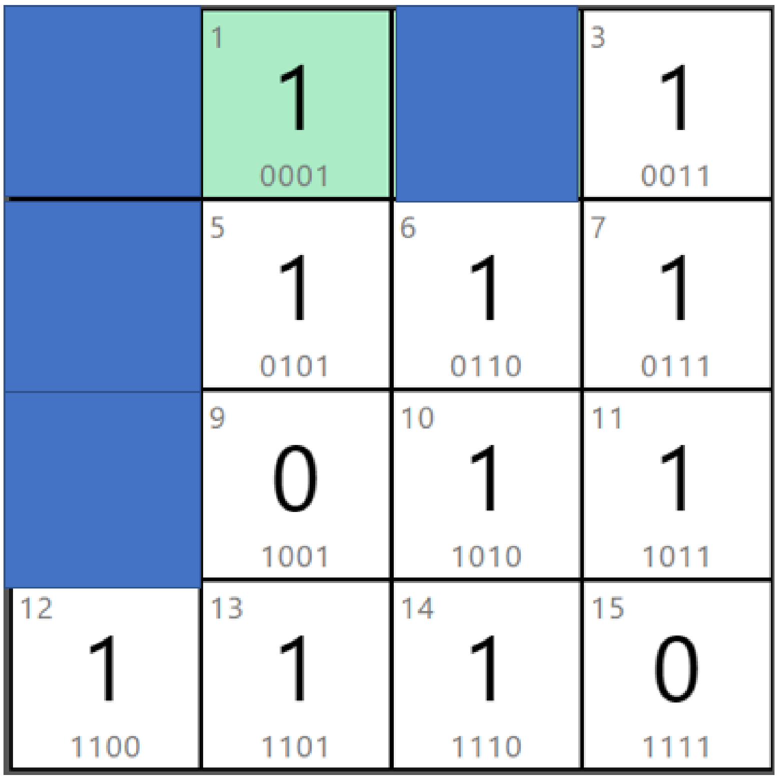


In this matrix we have two choices for choosing the value of parity. You have to decide whether you want odd parity or even parity. For now, we are working on even parity. In that, we have to count 1 in the block of bits and choose parity accordingly. For example, the number of 1s is odd so parity is 1. Thus p_1 became 1.

In the above image we can see that there is a shaded blue part. Those are parity spots and we have to decide the value for them. So you have to go two columns and two rows at a time for deciding parity. We compute four local parities and one global parity.

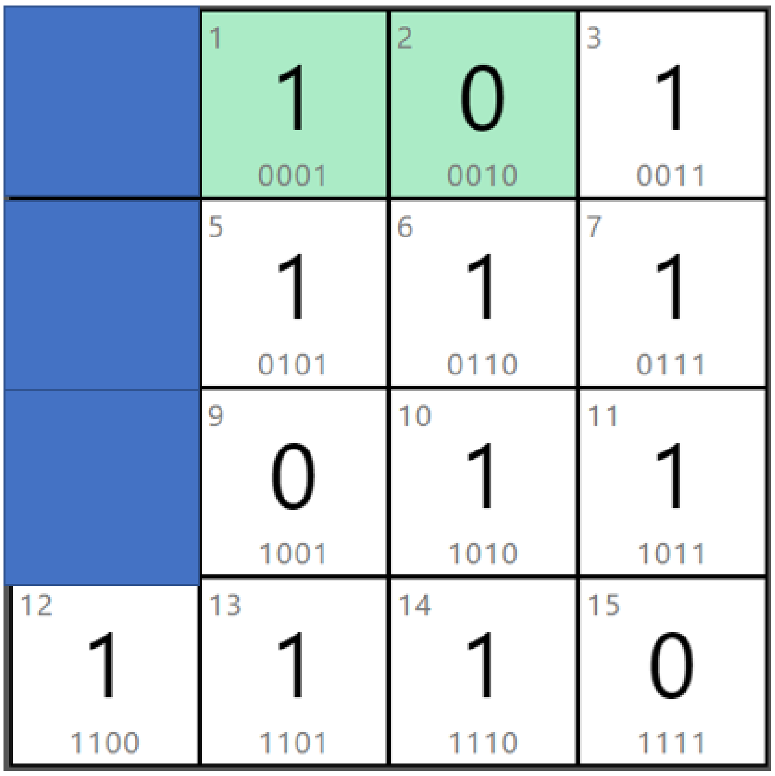
STEP 1 - second and fourth columns

->Here, we are selecting parity value 1 as we are following the even parity method.



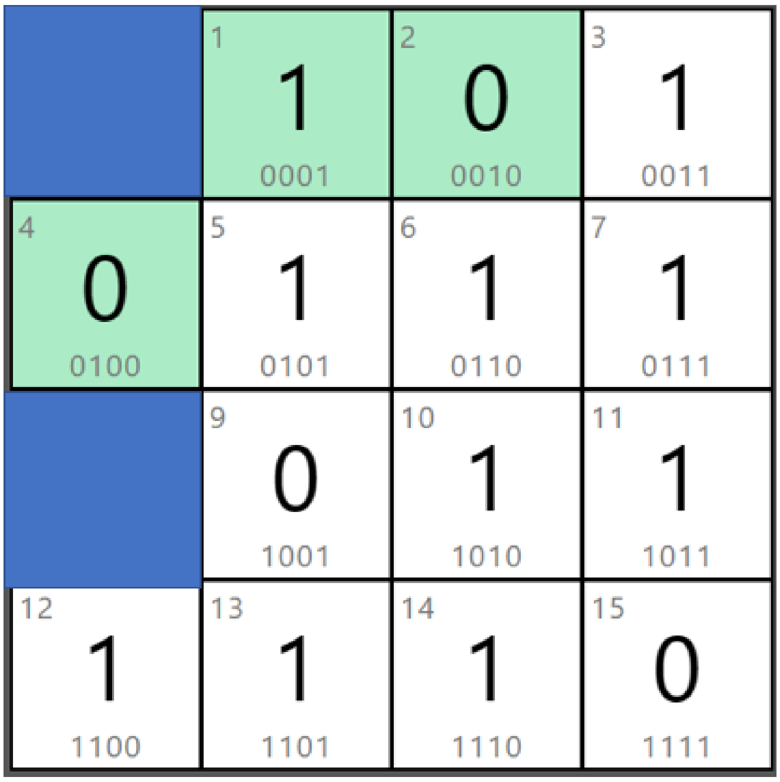
STEP 2 - third and fourth columns

-Now, the parity value is 0 from the 3^{rd} and 4^{th} column, following the even parity method..



STEP 3 - second and fourth row

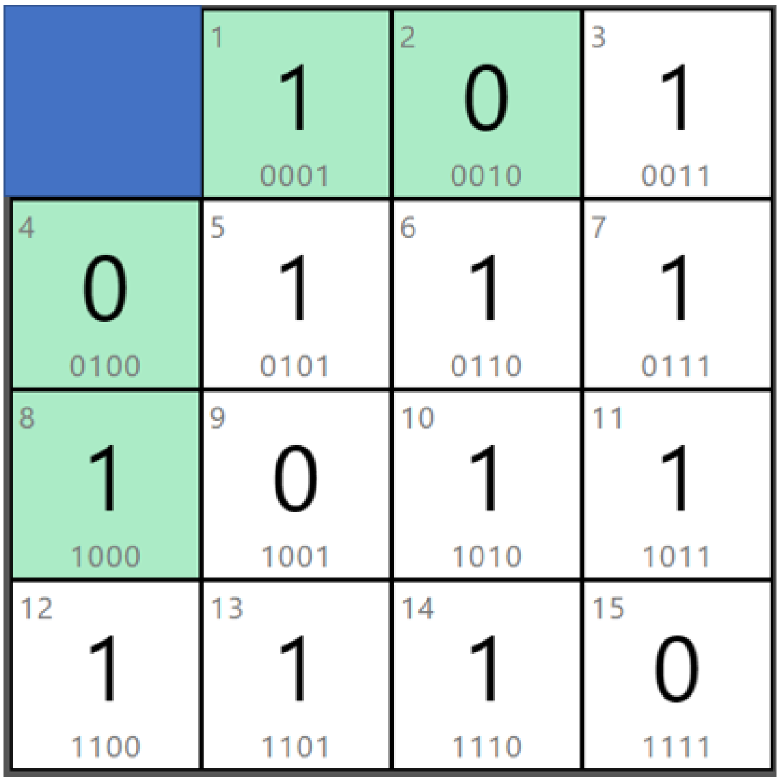
->Now, the parity value is 0 from the 2^{nd} and 4^{th} row.



🟦	1 1 0001	2 0 0010	3 1 0011
4 0 0100	5 1 0101	6 1 0110	7 1 0111
🟦	9 0 1001	10 1 1010	11 1 1011
12 1 1100	13 1 1101	14 1 1110	15 0 1111

STEP 4 - third and fourth row

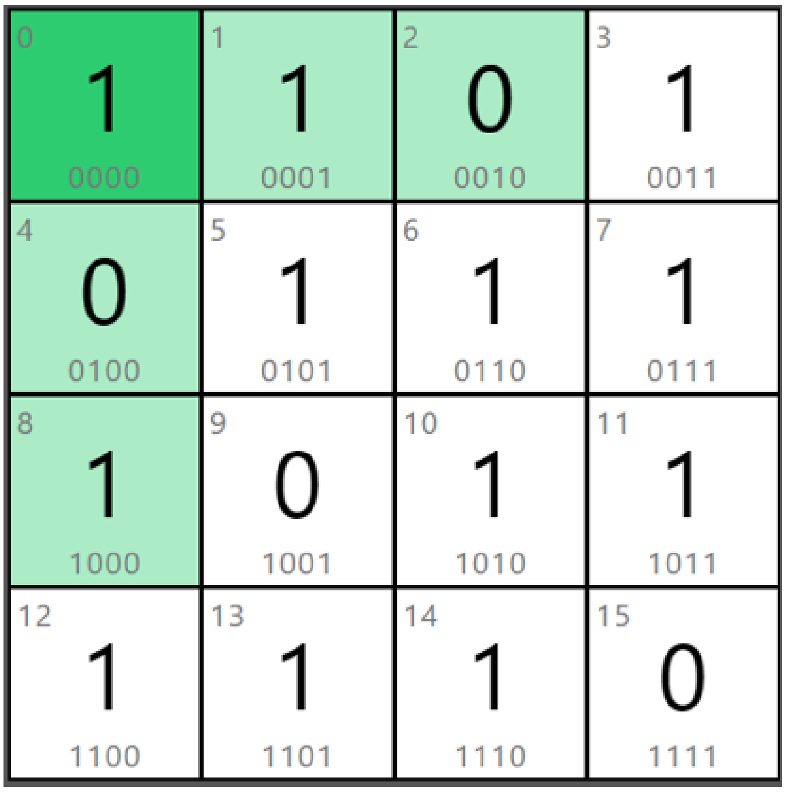
->Now, the parity value is 1 from the 3^{rd} and 4^{th} row.



	1 1 0001	2 0 0010	3 1 0011
4 0 0100	5 1 0101	6 1 0110	7 1 0111
8 1 1000	9 0 1001	10 1 1010	11 1 1011
12 1 1100	13 1 1101	14 1 1110	15 0 1111

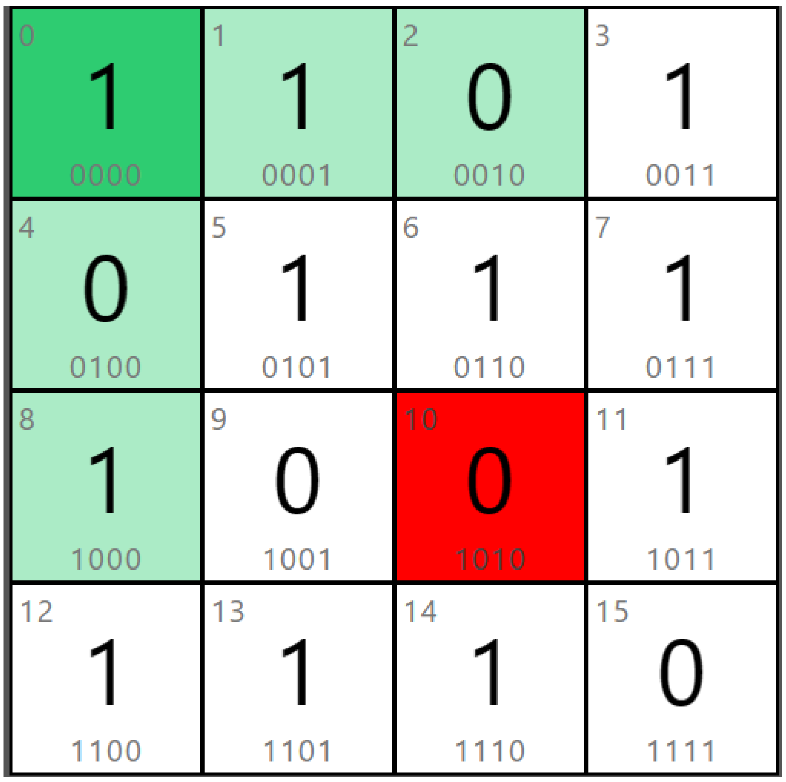
STEP 5 - Global parity

->For finding the global parity, find the parity bit according to all the remaining 15 bits. In this example, we are considering the global parity bit as 1 following the even parity method.



0 1 0000	1 1 0001	2 0 0010	3 1 0011
4 0 0100	5 1 0101	6 1 0110	7 1 0111
8 1 1000	9 0 1001	10 1 1010	11 1 1011
12 1 1100	13 1 1101	14 1 1110	15 0 1111

This binary table is encoded successfully and it is transmitted. Now suppose, that there is one error due to noise and cosmic-ray in communication technology as marked red in the figure::



0 1 0000	1 1 0001	2 0 0010	3 1 0011
4 0 0100	5 1 0101	6 1 0110	7 1 0111
8 1 1000	9 0 1001	10 0 1010	11 1 1011
12 1 1100	13 1 1101	14 1 1110	15 0 1111

Hamming codes can also find the error. It follows the below mentioned techniques:

1-) Logical-Methods::

In this method, we go column by column and row by row. So, you check the second and fourth columns. Does it have an odd parity? In our example "NO" then C_0=0;

After that you check the third and fourth columns. Does it have odd parity? In our example "YES". Thus, C_1=1.

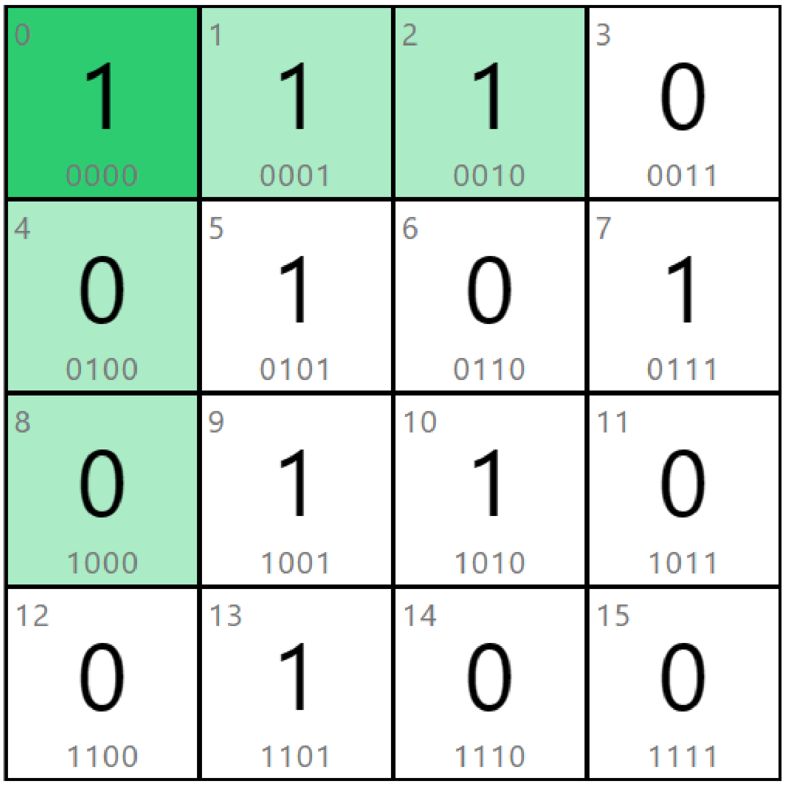
Similarly, you check the second and fourth rows. Do they have odd parity? In our example "NO"; so, C_2=0

Finally, check the third and fourth rows. Does it have odd parity? In our example "YES", thus C_3=1.
Now arrange C_0,C_1,C_2 and C_3 as [C3C2C1C0]. In our example, we can say that it is [1010] which means that the 10^{th} box i.e. (3,3) element in the matrix has an error. You can correct the error and get your original message.

2-) EXOR method::
You have to get that one-bit location which has one and after collecting all indexes as shown in the image you will find either zero or any non-zero number, If you get zero, then it means there is no error and if you get any non-zero number then that number shows you the index containing the bit which has an error. So here by example, you get indexes (0000,0001,0011,0101,0110,0111,1000,1011,1100,1101,1110) and exor all of them. You will get (1010) which shows the position of the error.

Two-bit error::

If you have a two-bit error, in that case, you will find that for our example, the total number of 1s remains the same. So you might think there is no error. But in that case, you also have to check for columns by column and row by row. You will find that there is no match with even the parity of that row or column. In that case, you can say that it has a two-bit error. Let's take one example in which you have to find the number of errors::



0 1 0000	1 1 0001	2 1 0010	3 0 0011
4 0 0100	5 1 0101	6 0 0110	7 1 0111
8 0 1000	9 1 1001	10 1 1010	11 0 1011
12 0 1100	13 1 1101	14 0 1110	15 0 1111

-> In the above image you see that the total no of 1s is even, so you think it has no error But if you look at the second and fourth columns, third and fourth columns, second and fourth row and third and fourth row, you will find that number of 1s is odd. This is contrary to the fact

that there are even numbers of 1s in total. This indicates that there is a Two-bit error.
->Drawback of this theorem is that you can't find the second-bit error location. You can only find one-bit error location.
->Due to this drawback of Hamming Code, Reed Solomon was invented which can correct multiple errors.

References

[1] Gil Cohen, Bernhard Haeupler, and Leonard J Schulman. Explicit binary tree codes with polylogarithmic size alphabet. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 535–544, 2018.

[2] Richard W Hamming. *Coding and information theory*. Prentice-Hall, Inc., 1986.

[3] Douglas J Muder. Minimal trellises for block codes. *IEEE Transactions on Information Theory*, 34(5):1049–1053, 1988.

[4] IS Reed. A brief history of the development of error correcting codes. *Computers & Mathematics with Applications*, 39(11):89–93, 2000.

[5] Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1998.