

Name of Faculty: Sunny Kumar Jain

Designation: Assistant Professor

Department: Electronics & Comm.

Subject: Data communication & Computer Networks

Unit: 3<sup>rd</sup> / Part One

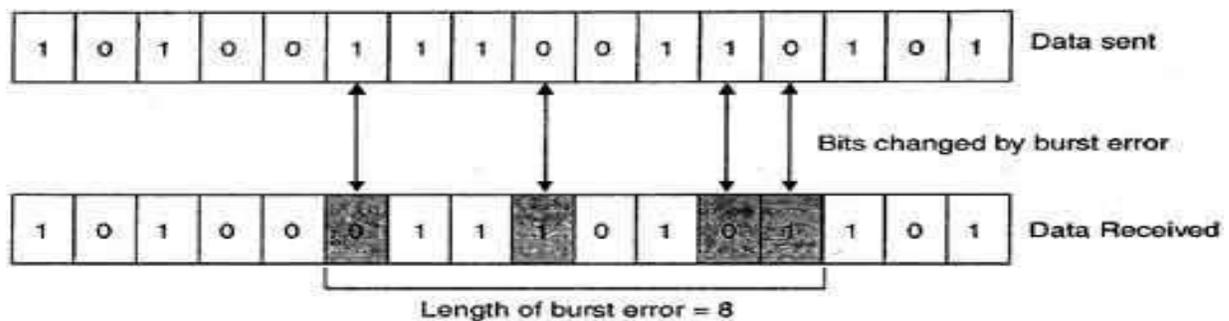
Topic: Error detection and correction: Types of errors –Detection – Vertical Redundancy Check (VRC) –Longitudinal Redundancy Check (LRC) –Cyclic Redundancy Check (CRC) –Check sum –Error Correction.

### Single Bit Error And Burst Error

- When data is being transmitted from one machine to another, it may be possible that data become corrupted on its way. Some of the bits may be altered, damaged or lost during transmission. Such a condition is known as error.
- The error may occur because of noise on line, attenuation and delay distortion. For reliable communication, it is important that errors are detected and corrected.

#### Burst Error:

- It means two or more bits in *data* unit are changed from 1 to 0 from 0 to 1 as shown in fig.



- In burst error, it is not necessary that only consecutive bits are changed. The length of burst error is measured from first changed bit to last changed bit.

As shown in fig. length of burst error is 8, although some bits are unchanged in between. Burst error is most likely to occur in a serial transmission. The noise occurring for a longer duration affects multiple bits. The number of bits affected depends on the data rate & duration of noise. For e.g. if data rate is 1 kbps, a noise of 1/100 second can affect 10 bits.

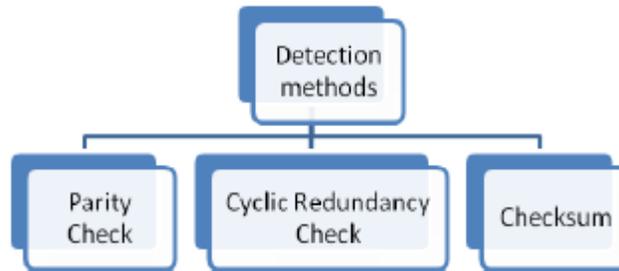
## **Error Detection Schemes**

- Error detection is most commonly realized using a suitable hash function (or checksum algorithm). A hash function adds a fixed-length tag to a message, which enables receivers to verify the delivered message by recomputing the tag and comparing it with the one provided.
- There exists a vast variety of different hash function designs. However, some are of particularly widespread use because of either their simplicity or their suitability for detecting certain kinds of errors (e.g., the cyclic redundancy check's performance in detecting burst errors).
- Random-error-correcting codes based on minimum distance coding can provide a suitable alternative to hash functions when a strict guarantee on the minimum number of errors to be detected is desired. Repetition codes, described below, are special cases of error-correcting codes: although rather inefficient, they find applications for both error correction and detection due to their simplicity.

## **Error Detection**

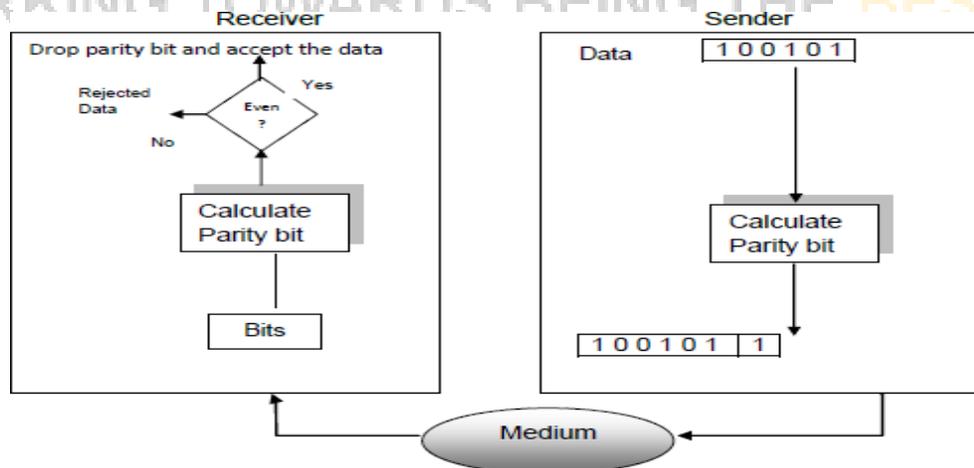
- Error detection is most commonly realized using a suitable hash function (or checksum algorithm). A hash function adds a fixed-length tag to a message, which enables receivers to verify the delivered message by recomputing the tag and comparing it with the one provided.
- There exists a vast variety of different hash function designs. However, some are of particularly widespread use because of either their simplicity or their suitability for detecting certain kinds of errors (e.g., the cyclic redundancy check's performance in detecting burst errors).

- Random-error-correcting codes based on minimum distance coding can provide a suitable alternative to hash functions when a strict guarantee on the minimum number of errors to be detected is desired. Repetition codes, described below, are special cases of error-correcting codes: although rather inefficient, they find applications for both error correction and detection due to their simplicity.



**Parity Check:**

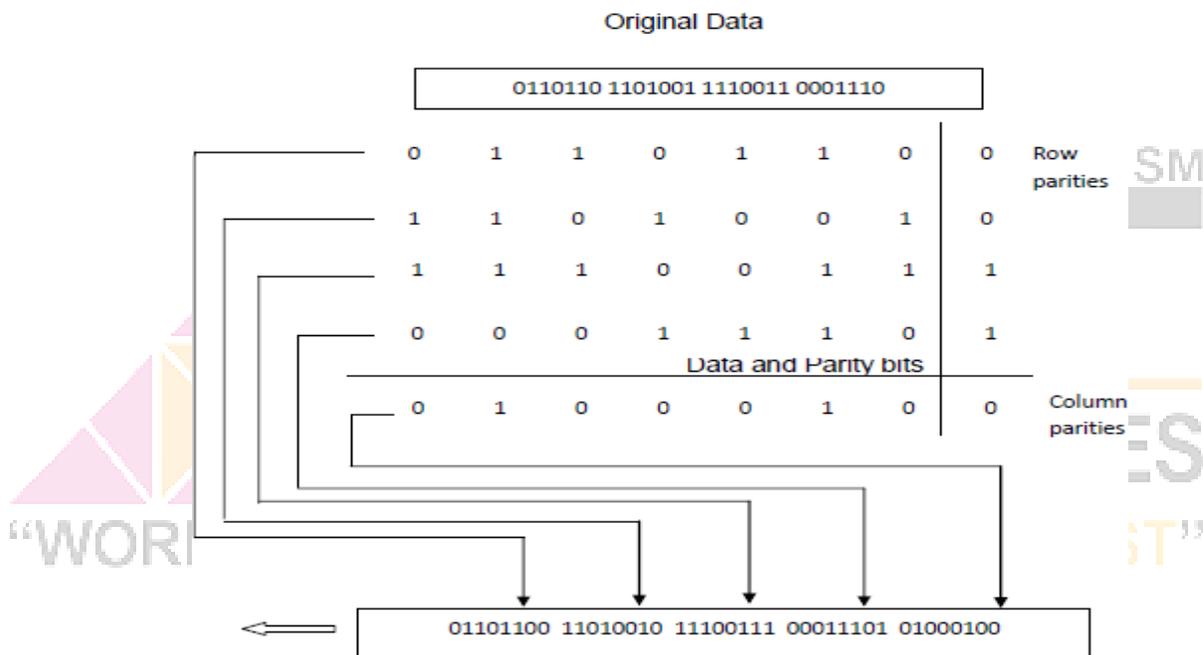
- In this technique, a redundant bit called a parity bit is added to every data unit so that the total number of 1's in the unit (including the parity bit) becomes even (or odd). Following Figure shows this concept when transmit the binary data unit 110101.



- Simple parity check can detect all single-bit errors. It can also detect burst errors as long as the total number of bits changed is odd. This method cannot detect errors where the total number of bits changed is even.

### Two-Dimensional Parity Check:

- A better approach is the two dimensional parity checks. In this method, a block of bits is organized in a table (rows and columns). First we calculate the parity bit for each data unit. Then we organize them into a table. We then calculate the parity bit for each column and create a new row of 8 bits. Consider the following example; we have four data units to send. They are organized in the tabular form as shown below.



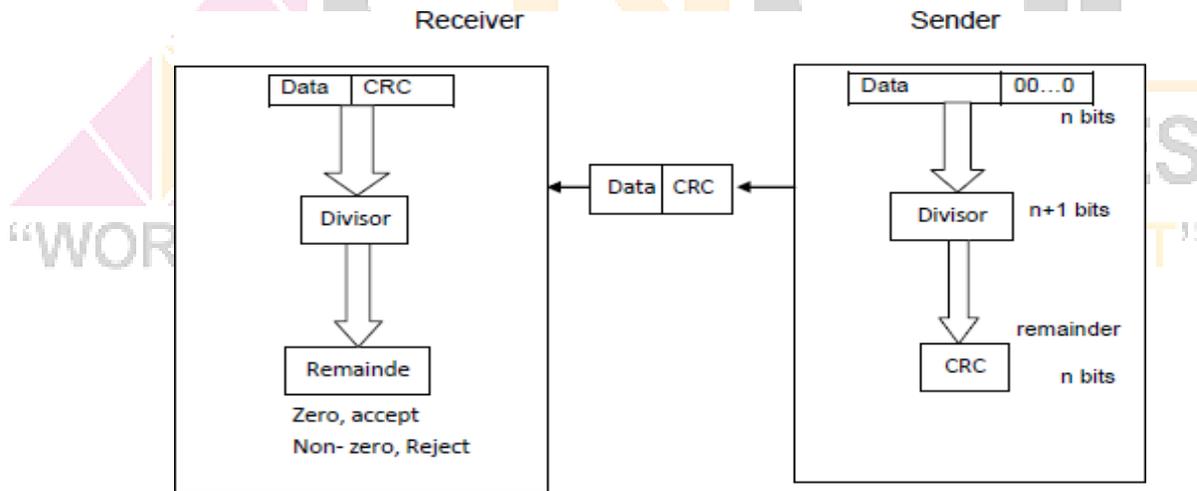
- We then calculate the parity bit for each column and create a new row of 8 bits; they are the parity bits for the whole block. Note that the first parity bit in the fifth row is calculated based on all first bits: the second parity bit is calculated based on all second bits: and so on. We then attach the 8 parity bits to the original data and send them to the receiver. Two-dimensional parity check increases the likelihood of detecting burst errors. A burst error of more than n bits is also detected by this method with a very high probability.

## Cyclic Redundancy Check (CRC)

- Most powerful of the redundancy checking techniques is the cyclic redundancy check (CRC). This method is based on the binary division. In CRC, the desired sequence of redundant bits are generated and is appended to the end of data unit. It is also called as CRC remainder. So that the resulting data unit becomes exactly divisible by a predetermined binary number.

At its destination, the incoming data unit is divided by the same number. If at this step there is no remainder then 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. The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor; the remainder is the CRC. To be valid, a CRC must have two qualities: It must have exactly one less bit than the divisor, and appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.

The following figure shows the process:



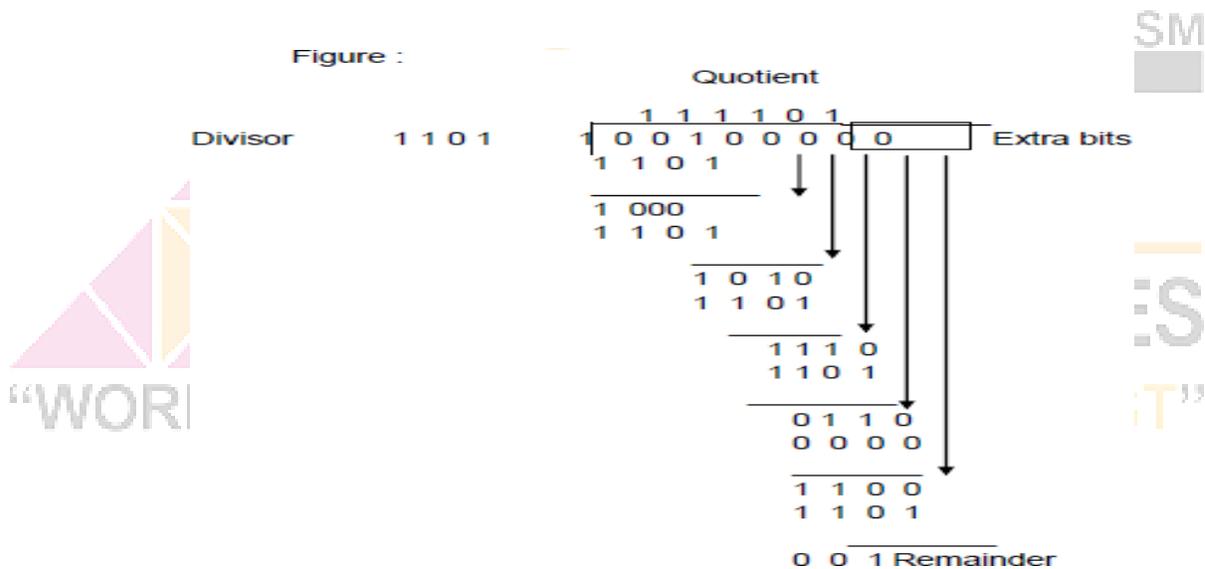
Step1: A string of 0's is appended to the data unit. It is n bits long. The number n is 1 less if-number of bits in the predetermined divisor which is n + 1 bits.

Step 2: The newly generated data unit is divided by the divisor, using a process called as binary division. The remainder resulting from this division is the CRC.

Step 3: the CRC of n bits derived in step 2 replaces the appended 0's at the data unit. Note that the CRC may consist of all 0's.

The data unit arrives at the receiver data first, followed by the CRC. The receiver treats the whole string as a unit and divides it by the same divisor that was used the CRC remainder. If the string arrives without error, the CRC checker yields a remainder of zero, the data unit passes. If the string has been changed in transit, the division yields zero remainder and the data unit does not pass.

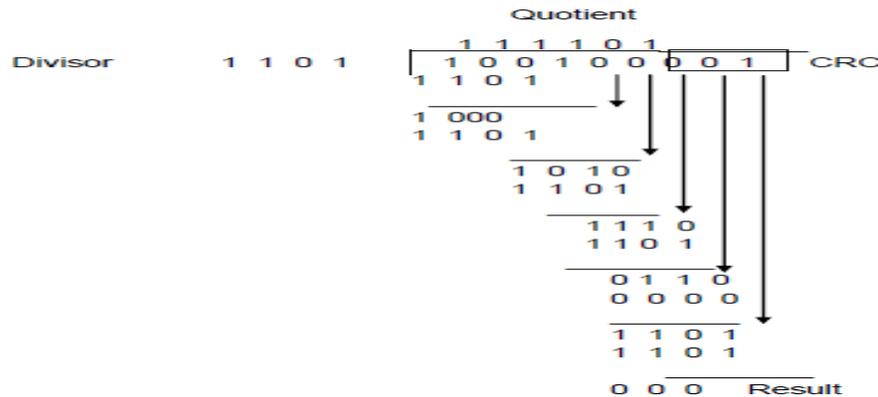
Following figure shows the process of generating CRC remainder:



A CRC checker functions does exactly as the generator does. After receiving the data appended with the CRC, it does the samemodulo-2 division. If the remainder is all 0's, the CRC is dropped and the data is accepted: otherwise, the received stream of bits is discarded and data is resent.

Following Figure shows the same process of division in the receiver.

Figure:



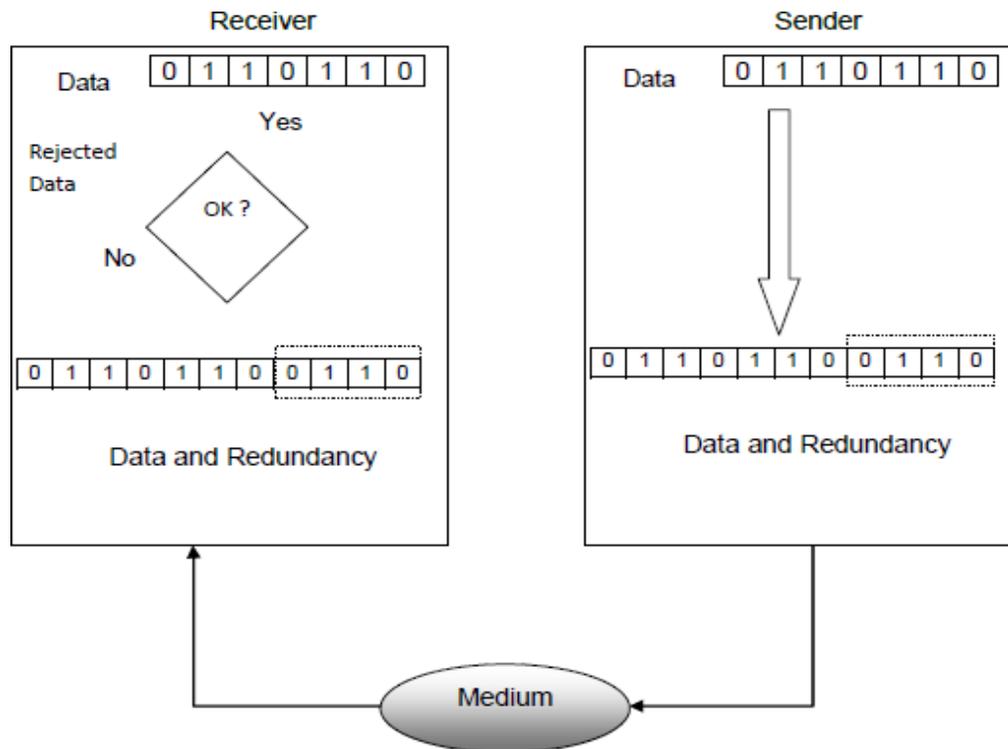
Performance: CRC is a very effective error detection method. If the divisor is chosen according to the previously mentioned rules, 1.CRC can detect all burst errors that affect an odd number of bits. 2.CRC can detect all burst errors of length less than or equal to the degree of the polynomial 3.CRC can detect, with a very high probability, burst errors of length greater than the degree of the polynomial.

### Checksum

A checksum is fixed length data that is the result of performing certain operations on the data to be sent from sender to the receiver. The sender runs the appropriate checksum algorithm to compute the checksum of the data, appends it as a field in the packet that contains the data to be sent, as well as various headers. When the receiver receives the data, the receiver runs the same checksum algorithm to compute a fresh checksum. The receiver compares this freshly computed checksum with the checksum that was computed by the sender. If the two checksum matches, the receiver of the data is assured that the data has not changed during the transit.

### Redundancy

- redundancy In order to detect and correct the errors in the data communication we add some extra bits to the original data. These extra bits are nothing but the redundant bits which will be removed by the receiver after receiving the data. Their presence allows the receiver to detect or correct corrupted bits. Instead of repeating the entire data stream, a short group of bits may be attached to the entire data stream. This technique is called redundancy because the extra bits are redundant to the information: they are discarded as soon as the accuracy of the transmission has been determined.



- There are different techniques used for transmission error detection and correction.

**Error Correction methods:-**

**Hamming Code:**

The Hamming code can be applied to data units of any length and uses the relationship between data and redundancy bits discussed above. For example, a 7-bit ASCII code requires 4 redundancy bits that can be added to the end of the data unit or interspersed with the original data bits. In following Figure, these bits are placed in positions 1, 2, 4, and 8 (the positions in an 11-bit sequence that are powers of 2). For clarity in the examples below, we refer to these bits as r1, r2, r4, and r8.

11 10 9 8 7 6 5 4 3 2 1

d	d	d	r8	d	d	d	r4	d	r2	r1
---	---	---	----	---	---	---	----	---	----	----

In the Hamming code, each r bit is the parity bit for one combination of data bits, is shown below:

- r1 : bits 1,3,5,7,9,11
- r2 : bits 2,3,6,7,10,11
- r3 : bits 4,5,6,7
- r4 : bits 8,9,10,11

r1 will be assigned to these bits position:

11 9 7 5 3

d	d	d	r8	d	d	d	r4	d	r2	r1
---	---	---	----	---	---	---	----	---	----	----

1

r2 will be assigned to these bits position:

11 10 7 6 3 2

d	d	d	r8	d	d	d	r4	d	r2	r1
---	---	---	----	---	---	---	----	---	----	----

r4 will be assigned to these bits position:

7 6 5 4

d	d	d	r8	d	d	d	r4	d	r2	r1
---	---	---	----	---	---	---	----	---	----	----

r8 will be assigned to these bits position:

11 10 9 8

d	d	d	r8	d	d	d	r4	d	r2	r1
---	---	---	----	---	---	---	----	---	----	----

Now suppose we have 1001101 data to be sent, then the redundant bits are calculated by the following method:

Data is: 1 0 0 1 1 0 1

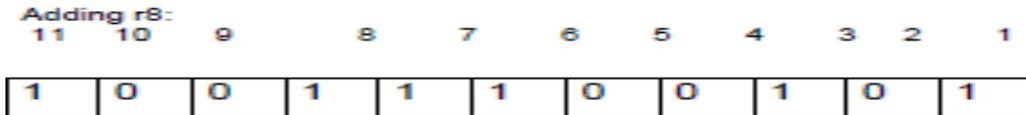
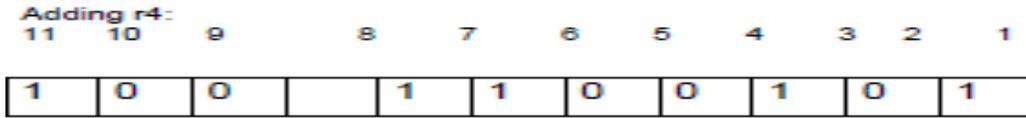
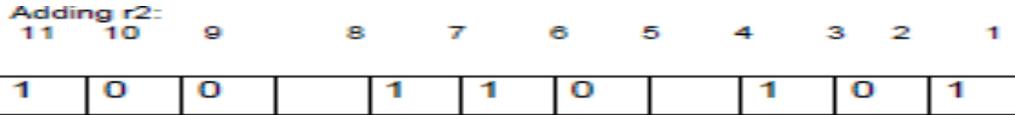
11 10 9 8 7 6 5 4 3 2 1

1	0	0		1	1	0		1		
---	---	---	--	---	---	---	--	---	--	--

Adding r1:

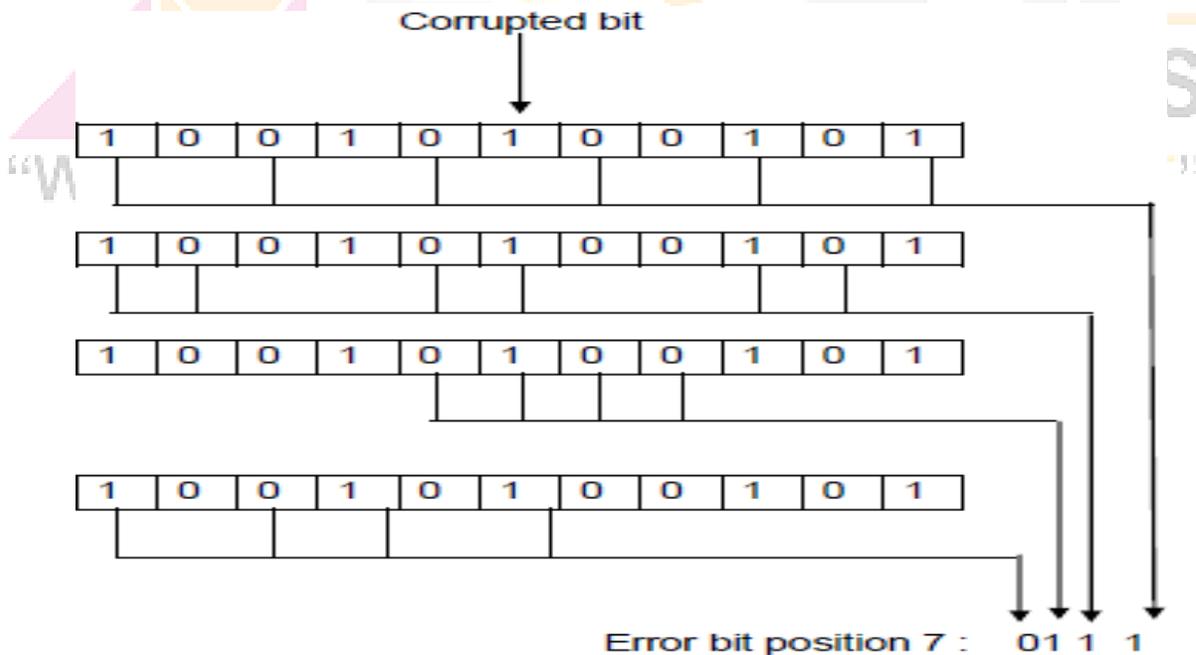
11 10 9 8 7 6 5 4 3 2 1

1	0	0		1	1	0		1		1



Code: 10011100101

- Now imagine that by the time the above transmission is received, the number 7 bit has been changed from 1 to 0. The receiver takes the transmission and recalculates 4 new parity bits, using the same sets of bits used by the sender plus the relevant parity r bit for each set (see following Fig.). Then it assembles the new parity values into a binary number in order of r position (r8 r4, r2, r1). In our example, this step gives us the binary number 0111 (7 in decimal), which is the precise location of the bit in error.



- Once the bit is identified, the receiver can reverse its value and correct the error. The beauty of the technique is that it can easily be implemented in hardware and the code is corrected before the receiver knows about it.