Toppersnotes
Unleash the topper in you

# UGC-NET

## Paper - 2

NATIONAL TESTING AGENCY (NTA)

# ELECTRONIC SCIENCE

Paper 2 – Volume 7

SIERRA INNOVATIONS

# Index

## Unit – 8

# Unit – 8

# COMMUNICATION

## # Basic Information Theory:

* Communication system performance is limited by:
  - Available signal power
  - Background noise
  - Bandwidth limits
* Can we postulate an ideal system based on physical principles, against which we can assess actual systems?
* The role of a communicaiton system is to convey, from transmitter to receiver, a sequence of messages selected from a finite number of possible messages. Within a specified time interval, one of these messages is transmitted, during the next another (maybe the same one), and so on.
  - The messsages are predetermined and known by the receiver.
  - The message selected for transmission during a particular interval is not known by the receiver.
  - The receiver knows the (different) probabilities for the selection of each message by the transmitter.
* The job of the receiver is not to answer the question "What was the message?", but rather "Which one?".

## 1. Amount of Information:

Suppose the allowable messages (or symbols) are

$$m_1,\ m_2,\ \ldots$$

and each have probability of occurrence

$$p_1,\ p_2,\ \ldots$$

The transmitter selects message $k$ with probability $pk$. (The complete set of symbols $\{m_1,\ m_2,\ \ldots\}$ is called the **alphabet**.)

If the receiver correctly identifies the message then an **amount of information** $\mathrm{I}_k$ given by

$$\mathrm{I}_k = \log_2 \frac{1}{pk}$$

has been conveyed. $\mathrm{I}_k$ is dimensionless, but is measured in **bits**.

**Example:**

Given two equiprobable symbols $m_1$ and $m_2$. Then $p_1 = 1/2$, so correctly idefifying $m_1$ carries

$$I_1 = \log_2 2 = 1 \text{ bit}$$

of information. Similarly the correct transmission of $m_2$ carries 1 bit of information.

Sometimes **bit** is used as an abbreviation for the term **binary digit**. If in the above example $m_1$ is represented by a 0 and $m_2$ by a 1, then we see that one binary digit carries 1 bit of information. This is not always the case-use the term **binit** for binary digit when confusion can occur.

**Example:**

Suppose the binits 0 and 1 occur with probabilities 1/4 and 3/4 respectively. Then binit 0 carries $\log_2 4 = 2$ bits of information and binit 1 carries $\log_2 4/3 = 0.42$ bits.

The definition of information statisfies a number of useful criteria:

- It is intuitive: the occurrence of a highly probable event carries little information ($I_k = 0$ for $p_k = 1$).
- It is positive: information may not decrease upon receiving a message ($I_k \geq 0$ for $0 \leq p_k \leq 1$).
- We gain more information when a less probabe message is received ($I_k > I_l$ for $p_k < p_l$).
- Information is additive if the messages are independent:

$$I_{k,\ l} = \log_2 \frac{1}{p_k p_l} = \log_2 \frac{1}{p_k} + \log_2 \frac{1}{p_l} = I_k + I_l$$

## 2. Average Infomation: Entropy:

* Supoose we have M different independent messages (as before), and that a long sequence of L messages is generated. In the L message sequence, we expect $p_1$ L occurrences of $m_1$, $p_2$L of $m_2$, etc.

The total informationin the sequence is

$$I_{total} = p_1 L \log_2 \frac{1}{p_1} + p_2 L \log_2 \frac{1}{p_2} + \ldots$$

so the average information per message interval will be

$$H = \frac{I_{total}}{L} = p_1 \log_2 \frac{1}{p_1} + p_2 \log_2 \frac{1}{p_2} + \ldots = \sum_{k=1}^{M} p_k \log_2 \frac{1}{p_k}.$$

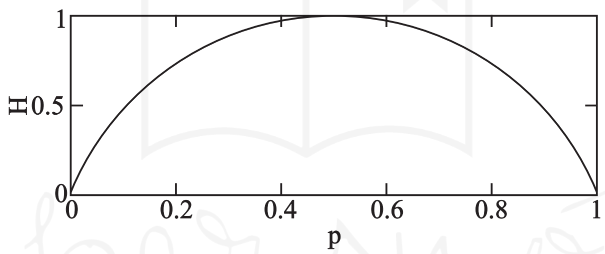
This average information is referred to as the **entropy**.

**Example:**

Consider the case of just two messages with probabilities $p$ and $(1 - p)$. This is called a **binary memoryless source**, since the successive symbols emitted are statistically independent.

The average information per message is

$$H = p \log_2 \frac{1}{p} + (1 - p)\log_2 \frac{1}{1-p}$$




The average information is maximised for $p = 1/2$, with a corresponding entropy of 1 bit/message.

In general, it can be proved that for $M$ messages, the entropy $H$ becomes maximum when all messages are equally likely. Then each message has probability $1/M$, and the entropy is

$$H_{max} = \sum_{k=1}^{M} \frac{1}{M} \log_2 M = \log_2 M$$

## 3. Information Rate:

* If the source of the messsages generates messages at the rate $r$ per second, then the **information rate** is defined to be

  $R = rH$ = average number of bits of information per second.

**Example:**

An analogue signal is band limited to $B$ Hz, sampled at the Nyquist rate, and the samples quantised to 4 levels. The quantisation levels $Q_1$, $Q_2$, $Q_3$ and $Q_4$ (messages) are assumed independent and occur with probabilities $p_1 = p_4 = 1/8$ and $p_2 = p_3 = 3/8$.

The average information per symbol at the source is

$$H = \sum_{k=1}^{4} p_k \log_2 \frac{1}{p_k}$$

$$= \frac{1}{8}\log_2 8 + \frac{3}{8}\log_2 \frac{8}{3} + \frac{3}{8}\log_2 \frac{8}{3} + \frac{1}{8}\log_2 8$$

$$= 1.8 \text{ bits/message.}$$

The information rate R is

$$R = rH = 2B(1.8) = 3.6 \text{ bits/s.}$$

Note that we could identify each message by a 2 digit binary code:

| Message | Probability | Binary Code |
|---|---|---|
| $Q_1$ | 1/8 | 0 0 |
| $Q_2$ | 3/8 | 0 1 |
| $Q_3$ | 3/8 | 1 0 |
| $Q_4$ | 1/8 | 1 1 |

If we transmit 2B messages per second, we will be transmitting 4B binits per second (since each message requires 2 binits).

Since each binit should be able to carry 1 bit of information, we should be able to transmit 4B bits of information using 4B binits. From the example we see that we are only transmitting 3.6 B bits. We are therefore not taking full advantage of the ability of binary PCM to convey information.

One remedy is to select different quantisation levels, such that each level is equally likely. Other methods involve the use of more complicated code assignments.

# # Special Instructional Objectives:

On completion of this lession, the student will be able to:

* Explain the need for error detection and correction
* State how simple parity check can be used to detect error
* Explain how two-dimensional parity check extends error detection capability
* State how checksum is used to detect error
* Explain how cyclic redundancy check works
* Explain how Hamming code is used to correct error

# Introduction:

* Environmental interference and physical defects in the communication medium can cause random bit errors during data transmission. Error coding is a method of detecting and correcting these errors to ensoure information is trasferred intact from its source to its destination. Error coding is used to fault tolerant computing in computer memory, magnetic and optical data storage media, satellite and deep space commuications, network communications, cellular telephone networks and almost any other form of digital data communicaiton. Error coding uses mathematical formulas to encode data bits at the source into longer bit words for transmission. The "code word" can then be decoded at the destination to retrieve the information. The extra bits in the code word provide redundancy that, according to the coding scheme used, will allow the destination to use the decoding process to determine if the communication medium introduced errors and in some cases correct them so that the data need not e retransmitted. Different error coding schemes are chosen depending on the types of errors expected, the communication medium's expected error rate and whether or not data retransmission is possible. Faster processors and better communications technology make more complex coding schemes, with better error detecting and correcting capacbilities, possible for smaller embedded systems, allowing for more robust communications. However, tradeoffs between bandwidth and coding overhead, coding complexity and allowable coding delay between transmissions, must be considered for each application.
* Even if we know what type of errors can occur, we can't simple recognize them. We can do this simply by comparing this copy received with another copy of intended transmission. In this mechanism the source data block is send twice. The receiver comapres them with the help of a comparator and if thoe two blocks differ, a request for re-transmission is made. To achieve forward error correction, three sets of the same data block are sent and majority decision selects the correct block. These methods are very inefficient and increase the traffic two or three times. Fortunately there are more efficient error detection and correction codes. There are two basic strategies for dealing with errors. One way si to include engouhg redundant information (extra bits are introduced into the data sream at the trnasmitter on a regular and logical basis) along with each block of data sent to enable the receiver to deduce what the transmitted character must have been. The other way is to include only enough redundancy to allow the receiver to deduce that eror has occurred, but not which error haas occurred and the receiver asks for a retransmission. The former strategy uses **Error-Correcting Codes** and latter uses **Error-detecting Codes**.
* To understand how errors can be handled, it is necessary to look closely at what error really is. Normally, a frame consists of m-data bits (i.e., message bits) and r-redundant bits (or check bits). Let the total number of bits be n (m + r). An n-bit unit containing data and check-bits is often referred to as an **n-bit codeword**.

* Given any two code-words, say 10010101 and 11010100, it is possible to determine how many corresponding bits differ, just EXCLUSIVE OR the two code-words and count the number of 1's in the result. The number of bits position in which code words differ is called the **Hamming distance**. If two code words are a Hamming distance d-apart, it will require d signal-bit errors to convert one code word to other. The error detecting and correcting properties depends on its Hamming distance.
  - To detecte d errors, you nedd a distance (d + 1) code because with such a code thre is no way that d-signal bit errors can change a valid code word into aother valid code word. Whenever receiver sees an invalid code word, it can tell that a transmission errr has occured.
  - Similarly, to correct d errors, you need a distance 2d+1 code because that way the legal code words are so far apart that even with d changes, the original codeword is still closer than any other code-word, so it can be uniquely determined.

First, various types of errors have been introduced in Sec. 3.2.2 followed by different error detecting codes in Sec. 3.2.3. Finally, error correctng codes have been introduced in Sec. 3.2.4.

## Types of Errors:

* These interferences can change the timing and shape of the signal. If the signal is carryiing binary encoded data, such changes can alter the meaning of the data. These errors can be divided into two types: Single-bit error and Burst error.

### Single-bit Error:

* The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1 as shown in figure.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | Sent |

Single bit change (1 is changed to 0)

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | Received |

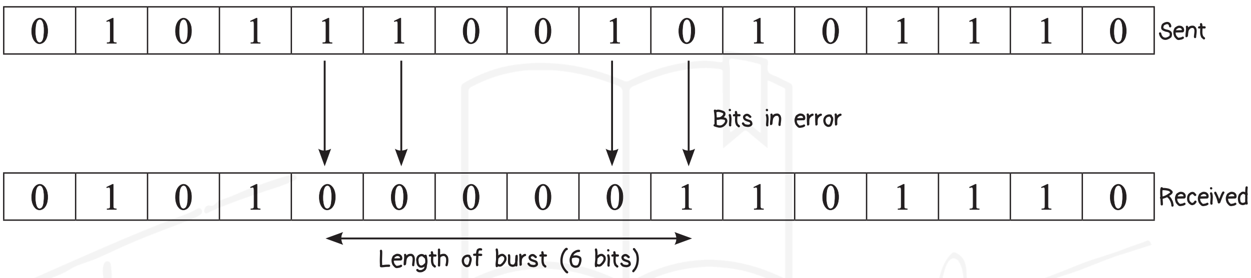
Single bit error

* Single bit errors are least likely type of errors in serial data transmission. To see why, imagine a sender sends data at 10 Mbps. This means that each bit lasts only for 0.1 $\mu$s (micro-second). For a single bit error to occur noise must have duration

of only 0.1 μs (micro-second), which is very rare. However, a single-bit error can happen if we are having a parallel data transmission. For example, if 06 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

## Burst Error:

* The term burst error means that two or more bits in the data unit have changed from 0 to 1 or vice-versa. Note that burst error doesn't necessary means that error occurs in consecutive bits. The length of the burst error is measured from the first corrupted bit to the last corrupted bit. Some bit in between may not be corrupted.




Burst Error

* Burst errors are mostly likely to happen in serial transmission. The duration of the noise is noramlly longer than the duration of a single bit, which means that the noise affects data; if affects a set of bits as shown in figure. The number of bits affected depends on the data rate and duration of noise.

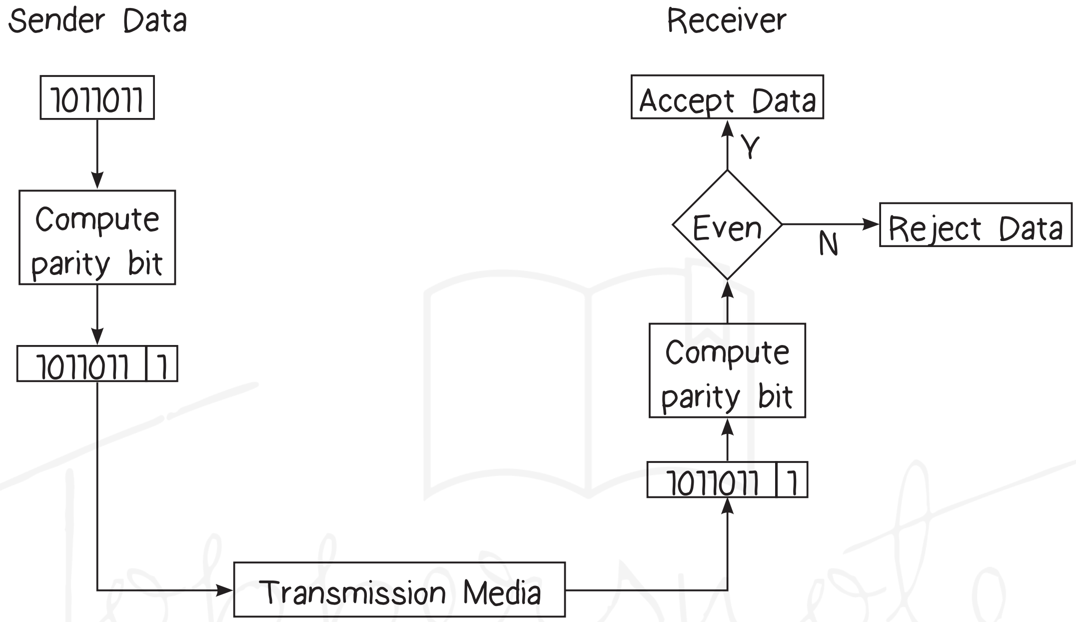
# Error Detecting Codes:

* Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are:
  - Simple Parity Check
  - Two-dimensional Parity Check
  - Checksum
  - Cyclic Redundancy Check

## Simple Parity Checking or One-dimension Parity Check:

* The most common and least expensive mechanism for error-detection is the simple parity check. In this technique. a redundant but called **parity bit**, is appended to every data unit so that the number of 1s is the unit (including the parity becomes even).
* 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 an odd number of 1's (ON bits) and 0 is added if it contains an even number of 1's. At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit, as shown in figure. This scheme makes the total number of 1's even, that is why it is called even parity checking. Considering a 4-bit word, different combinations of the data words and the corresponding code words are given in Table.




Even-parity checking scheme

Possible 4-bit data words and corresponding code words

| Decimal value | Data Block | Parity bit | Code word |
|---|---|---|---|
| 0 | 0000 | 1 | 00000 |
| 1 | 0001 | 1 | 00011 |
| 2 | 0010 | 1 | 00101 |
| 3 | 0011 | 0 | 00110 |
| 4 | 0100 | 1 | 01001 |
| 5 | 0101 | 0 | 01010 |
| 6 | 0110 | 0 | 01100 |
| 7 | 0111 | 1 | 01111 |
| 8 | 1000 | 1 | 10001 |
| 9 | 1001 | 0 | 10010 |
| 10 | 1010 | 0 | 10100 |
| 11 | 1011 | 1 | 10111 |
| 12 | 1100 | 0 | 11000 |

| | | | |
|---|---|---|---|
| 13 | 1101 | 1 | 11011 |
| 14 | 1110 | 1 | 11101 |
| 15 | 1111 | 0 | 11110 |

* Note that for the sake of simplicity, we are discussing here the even-parity checking, where the number of 1's should be an even number. It is also possible to use old-parity checking, where the number of 1's should be odd.

## Performance:

* An observation of the table reveals that to move from one code word to aother, at least two data bits should be changed. Hence these set of code words are said to have a minimum distance (hamming distance) of 2, which means that a receiver that has knowledge of the code word set can detect all single bit errors in each code word. However, if two errors occur in the code word, it becomes another valid member of the set and the decoder will see only another valid code word and know nothing of the error. Thus erros in more than one bit cannot be detected. In fact it can be shown that a single parity check code can detected only odd number of errors in a code word.

## Two-dimension Parity Check:

* Performance can be improved by using two-dimensional parity check, which organizes the block of bits in the form of a table. Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check btis 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. This is illustrated in figure.

Original data: 10110011 : 10101011 : 01011010 : 11010101

| Data | Row parities |
|---|---|
| 10110011 | 1 |
| 10101011 | 1 |
| 01011010 | 0 |
| 11010101 | 1 |
| Column parities: 10010111 | 1 |

101100111 : 101010111 : 010110100 : 110101011 : 100101111

Data to be sent

Two-dimension Parity Checking

## Performance:

- Two-Dimension Parity Checkiong increases the likelihood of detecting burst errors. As we have shown in figure that a 2-D parity check of *n* bits can detect a burst error of *n* bits. A burst error of more than *n* bits is also detected by 2-D partity check with a high-probability. There is, however, one pattewrn of error that remains elusive. If two bits ins one data unit are damaged and two bits in exactly same position in another data unit are also damaged, the 2-D Parity check checker will not detect an error. For example, if two data changed, making the data units as 01001110 and 001011101 the error be detected by 2-D parity check.

## 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 as shown in figure. At the receiver's end, all received segments are added using 1's complement to get the sum. The sum is complemented. If the result is zero, the received data is accepted, otherwise discarded, as shown in figure.

## Performance:

- The checksum detects all errors involving an odd number of bits. It also detects most errors involving even number of bits.

**Example:**

k = 4, m = 8

```
           10110011
           10101011
           --------
         ┌-01011110
         └→       1
           --------
           01011111
           01011010
           --------
           10111001
           11010101
           --------
         ┌-10001110
         └→       1
           --------
     Sum:  10001111
           --------
Checksum:  01110000
```

(a)

**Example:**

Received data
k = 4, m = 8

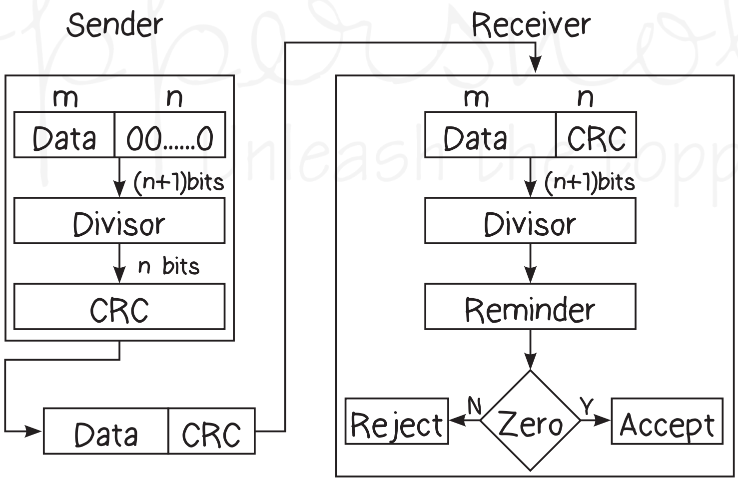
```
             10110011
             10101011
             --------
           ┌-01011110
           └→       1
             --------
             01011111
             01011010
             --------
             10111001
             11010101
             --------
           ┌-10001110
           └→       1
             --------
             10001111
             01110000
             --------
       Sum:  11111111
Complement = 00000000
Conclusion = Accept data
```

(b)

(a) Sender's end for the calculation of the checksum, (b) Receiving end for checking the checksum

## Cyclic Redundancy Checks (CRC):

- This Cyclic Redundancy Check is the most powerful and easy to implement technique. Unlike checksum scheme, whis 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 therefroe accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected. The generalized technique can be explained as follows.
- If a k bit message is to be transmitted, the transmitter generates an r-bit seuqnece, known as Frame Check Sequence (FCS) so that the (k + r) bits are actually being by r zeros, by a predetermined number. This number, which is (r + 1) bit in length, can also be considered as the coefficients of a polynomial, called Generator Polynomial. The remainder of this division process generates the r-bit FCS. On receiving the packet, the receiver divides the (k + r) bit frame by the same predtermined number and if it produces no remaineder, it can be assumed that no error has occurred during the transmission. Operations at both the sender and receiver and are shown in figure.




Basic scheme for Cyclic Redundancy Checking

- This mathematical operation performed is illustrated in figure by dividing a sample 4-bit number by the coeefficient of the generator polynomial $x^2 + x + 1$, which is 1011, using the modulo-2 arithmetic. Modulo-2 arithmetic is a binary addition process without any carry over, which is just the Exclusive-OR operation. Consider the case where $k$ = 1101. Hence we have to divide 1101000 (i.e. $k$ appended by 3 zeros) by 1011, which produces the remainder $r$ = 001, so that the bit frame $(k + r)$ = 1101001 is actually being jtransmitted through the communication channel. At the receiving end, if the received number, i.e., 1101001 is divided by the same generator polynomial 1011 to get the remainder as 000, it can be assumed that the data is free of errors.

```
              1111  <---- k
        ___________
  1011 | 1101000
         1011
         ----------
          1100
          1011
         ----------
           1110
           1011
         ----------
            1010
            1011
         ----------
             001  <---- r
```

Cyclic Redundancy Checks (CRC)

* The transmitter can generate the CRC by using a feedback shift register circuit. The same circuit can also be used at the receiving end to check whether any error has occurred. All the values can be expressed as polynomials of a dummy variable $X$. For example, for $P = 11001$ the corresponding polynomial is $X^4 + X^3 + 1$. A polynomial is selected to have at least the following properties:
  - It should not be divisible by $X$.
  - It should not be divisible by $(X + 1)$.
* The first condition guarantees that all burst errors of a length equal to the degree of polyomial are detected. The second condition guarantees that all burst errors affecting an odd number of bits are detected.
* CRC process can be expressed as $X^nM(X)/P(X) = Q(X) + R(X)/P(X)$
* Commonly used divisor polynomials are:
  - CRC-16 = X16 + X15 + X2 + 1
  - CRC-CCITT = X16 + X12 + X5 + 1
  - CRC-32 = $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + 1$

## Performance:

* CRC is a very effective error detection technique. If the divisor is chosen according to the previously mentioned rules, its performance can be summarized as follows:
  - CRC can detect all single-bit errors
  - CRC can detect all double-bit errors (three 1's)

- CRC can detect any odd number of errors $(X + 1)$
- CRC can detect all burst errors of less than the degree of the polynomial.
- CRC detects most of the larger burst errors with a high probability.
- For example CRC-12 detects $99.97\%$ of errors with a length 12 or more.

## Error Correcting Codes:

- The techniques that we have discussed so far can detect errors, but do not correct them. **Error Correction** can be handled in two ways.
  - One is when an error is discovered; tghe receiver can have the sender retransmit the entire data unit. This is known as **backward error correction**.
  - In the other, receiver can use an error-correcting code, which automatically corrects certain errors. This is known as **forward error correction**.
- In theory it is possible to correct any number of errors atomically. Error-correcting codes are more sophisticated than error detecting codes and require more redundant bits. The number of bits required to correct multiple-bit or burst error is so high that in most of the cases it is inefficient to do so. For this reason, most error correction is limited to one, two or at the most three-bit errors.

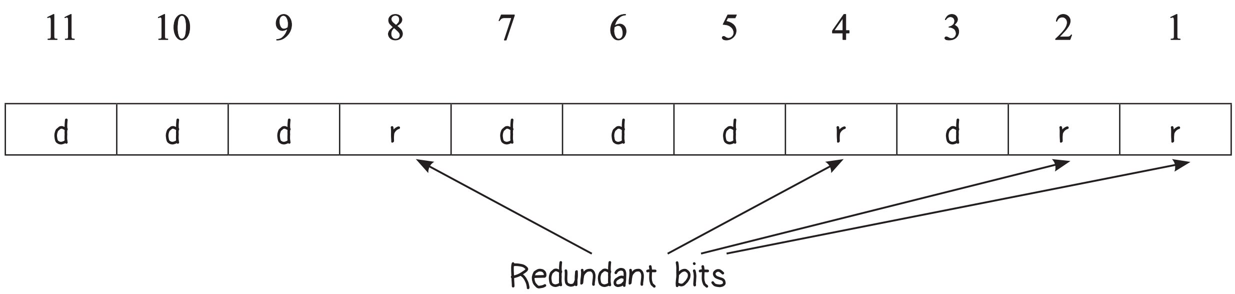
### Single-bit Error Correction:

- Concept of error-correction can be easily understood by examining the simplest case of single-bit errors. As we have already seen that a single-bit error can be detected by addition of a parity bit (VRC) with the data, which needed to be send. A single additional bit can detect error, but it's not sufficient engough to correct that error too. For correcting an error one has to know the exact position of error, i.e., exactly which bit is in error (to locate the invalid bits). For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in errorr. To this, we have to add some additional redundant bits.
- To calculate the numbers of redundant bits $(r)$ required to correct $d$ data bits, let us find out the relationship between the two. So we have $(d + r)$ as the total number of bits, which are to be transmitted; then $r$ must be able to indicate at least $d + r + 1$ different values. Of these, one value means no error, and remaining $d + r$ values indicate error location of error in each of $d + r$ locations. So, $d + r + 1$ states must be distinguishable by $r$ bits and $r$ bits can indicates $2^r$, states, Hence, $2^r$ must be greater than $d + r + 1$.

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

- The value of $r$ must be determined by putting in the value of $d$ in the relation. For example, if $d$ is 7, then the smallest value of $r$ that statisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits $(d + r = 7 + 4 = 11)$.

* Now let us examine how we can manipulate these bits to discover which bit is in error. A technique developed by R. W. Hamming provides a partical solution. The solution or coding scheme he developed is commonly known as Hamming Code. Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits as discussed.




Positions of redundancy bits in hamming code

7 6 5 4 3 2 1

$d_4$ $d_3$ $d_2$ $r_4$ $d_1$ $r_2$ $r_1$

$r_1$ → 1, 3, 5, 7

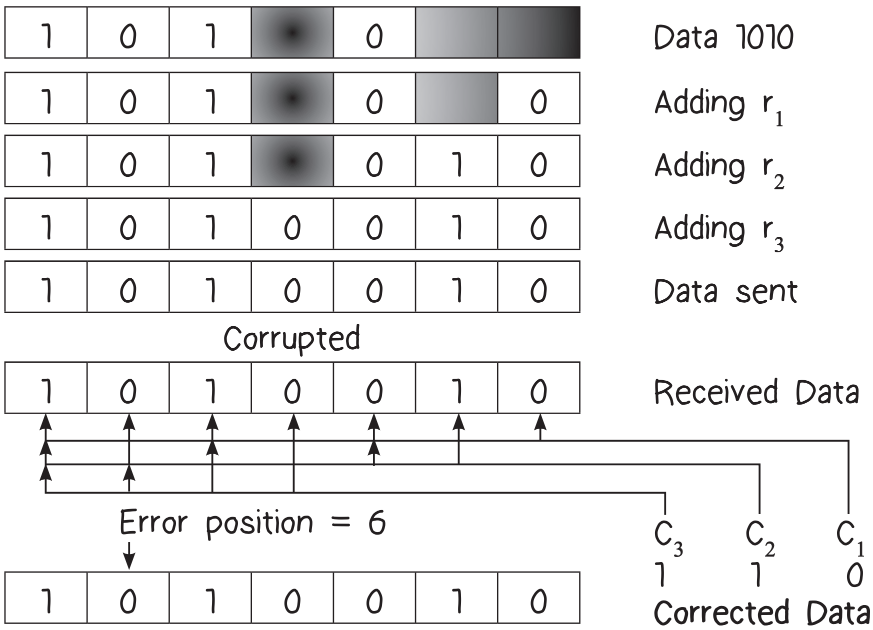
$r_2$ → 2, 3, 6, 7

$r_4$ → 4, 5, 6, 7

7 6 5 4 3 2 1

$d_4$ $d_3$ $d_2$ $r_4$ $d_1$ $r_2$ $r_1$

| Error position | Position number c3 c2 c1 |
|---|---|
| 0 (no error) | 0 0 0 |
| 1 | 0 0 1 |
| 2 | 0 1 0 |
| 3 | 0 1 1 |
| 4 | 1 0 0 |
| 5 | 1 0 1 |
| 6 | 1 1 0 |
| 7 | 1 1 1 |




Use of Hamming code for error correction for a 4-bit data

* Basic approach for error detection by using Hamming code is as follows:
  - To each group of m information bits $k$ parity bits are added to form $(m + k)$ bit code as shown in figure.
  - Location of each of the $(m + k)$ digits is assigned a decimal value.
  - The $k$ parity bits are placed in positions 1, 2, ..., $2^{k-1}$ positions. –K parity checks are performed on selected digits of each codeword.
  - At the receiving end the parity bits are recalculated. The decimal value of the $k$ parity bits provides the bit-position in error, if any.
* Figure shows how hamming code is used for correction for 4-bit numbers $(d_4d_3d_2d_1)$ with the help of three redundant bits $(r_3r_2r_1)$. For the example data 1010, first $r_1$ (0) is calculated considering the parity of the bit positions, 1, 3, 4 and 7. Then the parity bits $r_2$ is calculated considering bit positions 2, 3, 6 and 7. Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown. If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found out by calcualting $r_3r_2r_1$ at the receiving end. For example, if the received code word is 1110010, the recalculated value of $r_3r_2r_1$, which indicates that bit position in error is 6, the decimal value of 110.

**Example:**

Let us consider an example for 5-bit data. Here 4 parity bits are required. Assume that during transmission bit 5 has been changed from 1 to 0 as shown in figure. The receiver receives the code word and recalculates the four new parity bits using the same set of bits used by the sender plus the relevant parity (r) bit for each set (as shown in figure). Then it assembles the new parity values into a binary number in order of r positions (r8, r4, r2, r1).

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | | 1 | 0 | 1 | | 1 | | |

Data to be send

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

Data to be send along with redundant bits

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Data Received

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | | 1 | 0 | 0 | | 1 | | |

Data Received Minus Parity Bits

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | | | | 1 | | 0 | 1 |

Parity bits recalculated

Calculations;

Parity recalculated (r8, r4, r2, r1) = 01012 = 510

Hence, bit 5th is in error i.e. d5 is in error.

So, correct code-word which was transmitted is:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | | 1 | 0 | 1 | | 1 | | |

Use of Hamming code for error correction for a 5-bit data