**UNIT – 3**

**Data Link Layer**

**Data Encoding: Introduction, Block coding, cyclic codes, checksum, framing, Noiseless channels, noisy channels, Asynchronous and Synchronous transmission, Full and Half duplex**

**Encoding schemes: BCA (NRZ, Bipolar AMI, B8ZS, HDB3, ASK, FSK, PSK, PCM, AM, FM, PM),**

**Data Link Control: Flow control: Stop and Wait, Sliding window**

**Error detection: Parity Check, CRC. Error control: Stop and Wait ARQ, Go back-N ARQ, Selective-Reject ARQ, Briefidea of HDLC and other Data Link control protocols.**

**PYQ (2023)**

1. What do you mean by a noisy channel?
2. Define a frame.
3. Differentiate between asynchronous and synchronous transmission.
4. Write short note on :-
5. CRC
6. Sliding Window
7. Part**-C**: Draw and explain HDLC and Stop and Wait ARQ.

**PYQ (2022)**

1. How Manchester and Differential Manchester encoding solve the problem of synchronization ?
2. Draw Digital encoding schemes ?
3. List some error detection techniques. How many errors can an "n bit" CRC detect? If a message 11001001 is to be transmitted using the CRC polynomial X³ + 1 to protect it from errors. Calculate the message to be transmitted.

**PYQ (2021)**

1. What is Encoding ?
2. Asynchornous vs Synchronous transmission
3. Go back-N ARQ
4. Sliding Window Protocol
5. Draw Digital encoding schemes ?

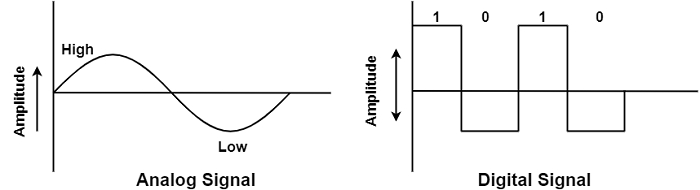
**Data Encoding**

Data encoding is the process of converting data, such as binary sequences, characters, or symbols, into a specific format that is optimized for efficient transmission, storage, or processing. It uses patterns of signals, like electrical or radio waves, to represent the data (like 1s and 0s) so that it can be understood by computers or communication systems. Encoding also helps detect and fix errors during transmission.

**Encoding Techniques-**

The data encoding technique is divided into the following types :-

1. **Analog data to Analog signals** − The modulation techniques such as Amplitude Modulation (AM), Frequency Modulation (FM) and Phase Modulation (PM) of analog signals, fall under this category.
2. **Analog data to Digital signals** − This process can be termed as digitization, which is done by Pulse Code Modulation (PCM). Hence, it is nothing but digital modulation. As we have already discussed, sampling and quantization are the important factors in this. Delta Modulation gives a better output than PCM.
3. **Digital data to Analog signals** − The modulation techniques such as Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), Phase Shift Keying (PSK), etc., fall under this category.
4. **Digital data to Digital signals** − There are several ways to map digital data to digital signals. e.g. NRZ, Bipolar AMI, Manchester & Differential Manchester encoding, etc.



**Block Coding**

Block coding is a method used to encode data into a specific format. The purpose of block coding is to **add redundant information to the data**, which can be **used to detect and correct errors that may occur during transmission or storage**. Block coding is often used in conjunction with **error correction codes (ECCs)** to provide a more robust way of transmitting and storing data.

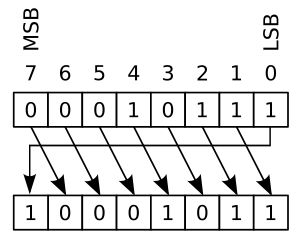
*Types of Block Codes* *:-*

1. **Parity Check Codes** - In this technique, an extra parity bit is included with each block of data. The calculation of the parity bit is done as per the number of 1s in the block of data. However, the parity check codes can detect only 1-bit errors, also they cannot correct them.
2. **Hamming Codes** - Hamming codes are relatively advanced codes than parity check codes used for block coding. These codes are able to detect as well as correct 1-bit errors. This method adds additional redundant bits to each data block to create a specific code-word. The positions of the redundant bits in the code-word allow for detection and correction of errors in the data.
3. **Reed-Solomon Codes** - Reed-Solomon codes are highly advanced codes used for block coding where robust error detection and correction is desired. These codes have ability to detect and correct multi-bit errors in a data block. The operation of Reed-Solomon Codes is based on the combination of parity checks and polynomial mathematics, where parity check detects errors in the data block, while the error locator polynomials correct them.
4. **BCH (Bose-Chaudhuri-Hocquenghem) Codes** - BCH codes are a type of block code that can correct a specific number of errors. They are commonly used in digital communication systems to ensure the accuracy of transmitted data. BCH codes are mainly used where error correction is required multiple times.

**Cyclic Codes**

A cyclic code is a block code, where the **circular shifts of each codeword gives another word that belongs to the code**. They are **error-correcting codes** that have **algebraic properties** that are convenient for efficient error detection and correction.

**Some examples of cyclic codes include**: BCH codes, Hamming codes, Golay codes, RS codes, and LDPC codes.

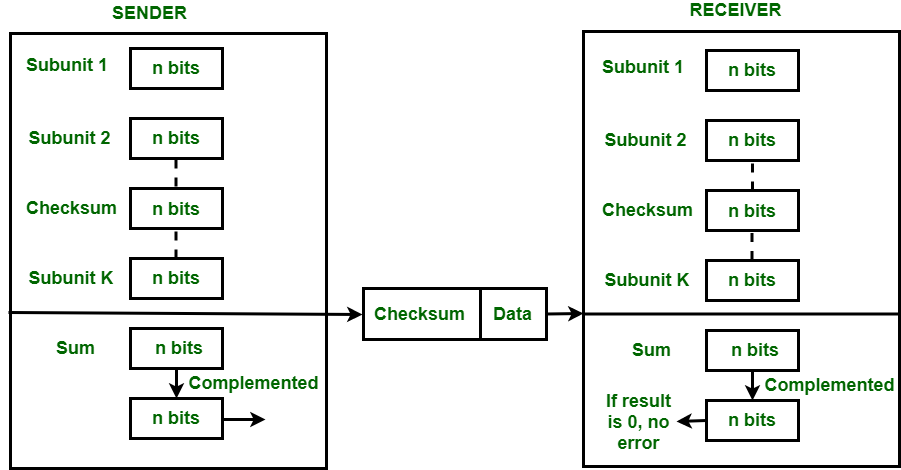


**Checksum**

**Checksum** is the **error detection method** used by upper layer protocols and is considered to be more reliable than LRC, VRC and CRC. This method makes the use of **Checksum Generator** on Sender side and **Checksum Checker**on Receiver side.

At the Sender side, the data is divided into equal subunits of n bit length by the checksum generator. This bit is generally of 16-bit length. These subunits are then added together using one’s complement method. The resultant bit is then **complemented**. This **complemented sum** which is called **checksum** is appended to the end of original data unit and is then transmitted to receiver.

The Receiver after receiving data + checksum passes it to checksum checker. Checksum checker divides this data unit into various subunits of equal length and adds all these subunits. These subunits also contain checksum as one of the subunits. The resultant bit is then complemented. **If the complemented result is zero, it means the data is error-free. If the result is non-zero it means the data contains an error and Receiver rejects it.**

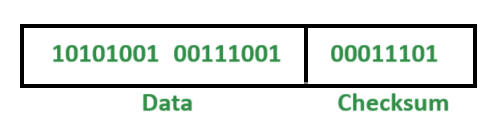


**Example –**If the data unit to be transmitted is 10101001 00111001, the following procedure is used at Sender side and Receiver side.

**Sender Side:**

10101001 subunit 1   
00111001 subunit 2   
11100010 sum (using 1s complement)   
**00011101**  checksum (complement of sum)

**Data transmitted to Receiver is:**



**Receiver Side :**

10101001 subunit 1   
00111001 subunit 2   
00011101 checksum   
11111111 sum  
**00000000** sum's complement

**Result is zero, it means no error.**

**Advantages -** The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

**Disadvantages -** The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

**Example –** If the data transmitted along with checksum is 10101001 00111001 00011101. But the data received at destination is **0**0101001 **1**0111001 00011101.

**Receiver Side:**

**0**0101001 1st bit of subunit 1 is damaged  
**1**0111001 1st bit of subunit 2 is damaged  
00011101 checksum  
11111111 sum  
00000000 Ok 1's complement

**Although data is corrupted, the error is undetected.**

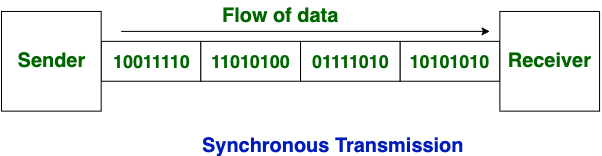
**Framing**

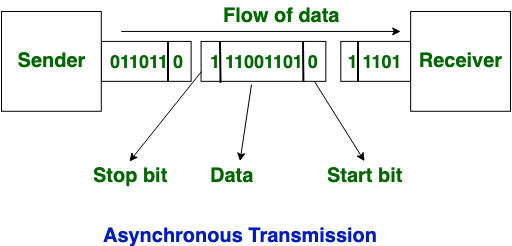
**Noisy channels**

**Noiseless channels**

**Asynchronous VS Synchronous Transmission**

| **Basis** | **Synchronous Transmission** | **Asynchronous Transmission** |
| --- | --- | --- |
| **Data Format** | Data is sent in blocks or frames, making it more efficient for large data transfers. | Data is sent in bytes or characters, ideal for smaller data transmissions. |
| **Speed** | Faster due to continuous data flow, suitable for real-time communication like video calls. | Slower as each byte requires additional start and stop bits, making it less efficient for real-time uses. |
| **Cost** | More expensive due to the need for precise clock synchronization and advanced hardware. | More economical, as it requires simpler hardware without the need for synchronization. |
| **Time Interval** | Constant time interval for data transmission, ensuring a steady flow of information. | Irregular and random time intervals between bytes, making it less predictable. |
| **User Response** | Users must wait until the entire transmission is complete before getting a response. | Users receive responses without waiting for the whole transmission, improving responsiveness. |
| **Gap Between Data** | No gap between data blocks, ensuring a smooth and continuous data flow. | Gaps between transmitted bytes create pauses in the communication flow. |
| **Transmission Line Usage** | Efficient use of transmission lines, minimizing idle time during communication. | Inefficient use, as the line remains idle during gaps between character transmissions. |
| **Start and Stop Bits** | No start and stop bits are used, which reduces overhead and increases efficiency. | Start and stop bits are required for each byte, adding extra overhead. |
| **Clock Synchronization** | Requires precise clock synchronization, making sure both sender and receiver are in sync. | No need for synchronized clocks, as the parity bit handles byte information. |
| **Error Detection and Correction** | Errors are detected and corrected in real-time, improving accuracy during transmission. | Errors are only detected and corrected after the data has been completely received. |
| **Latency** | Low latency because data is transmitted in real-time with no gaps. | Higher latency due to the gaps between bytes and processing delays. |
| **Examples** | Telephonic conversations, Video conferencing, Online gaming where real-time data transfer is needed. | Email, File transfer, and online forms where instant response is not crucial. |
| **Transmission Mode** | Often used in Full-Duplex, where data is sent and received at the same time. | Usually used in Half-Duplex, where data is sent one way at a time with pauses. |





**Full VS Half Duplex**

| **Parameters** | **Simplex** | **Half Duplex** | **Full Duplex** |
| --- | --- | --- | --- |
| **The direction of communication** | Simplex mode is a uni-directional communication. | Half Duplex mode is a two-way directional communication but one at a time. | Full Duplex mode is a two-way directional communication simultaneously. |
| **Sender and Receiver** | In simplex mode, Sender can send the data but that sender can’t receive the data. | In Half Duplex mode, Sender can send the data and also can receive the data but one at a time. | In Full Duplex mode, Sender can send the data and also can receive the data simultaneously. |
| **Channel usage** | Usage of one channel for the transmission of data. | Usage of one channel for the transmission of data. | Usage of two channels for the transmission of data. |
| **Performance** | The simplex mode provides less performance than half duplex and full duplex. | The Half Duplex mode provides less performance than full duplex. | Full Duplex provides better performance than simplex and half duplex mode. |
| **Bandwidth Utilization** | Simplex utilizes the maximum of a single bandwidth. | The Half-Duplex involves lesser utilization of single bandwidth at the time of transmission. | The Full-Duplex doubles the utilization of transmission bandwidth. |
| **Suitable for** | It is suitable for those transmissions when there is requirement of full bandwidth for delivering data. | It is suitable for those transmissions when there is requirement of sending data in both directions, but not at the same time. | It is suitable for those transmissions when there is requirement of sending and receiving data simultaneously in both directions. |
| **Examples** | Example of simplex mode are: Keyboard and monitor. | Example of half duplex mode is: Walkie-Talkies. | Example of full duplex mode is: Telephone. |