HOMEWORK-1

1. Distinguish between UART, USART, RS-232 and RS-485, and also mention the physical layer specifications for each (voltage levels, timing, frequencies, wiring conventions etc.

Ans. UART (Universal Asynchronous Receiver/Transmitter) and USART (Universal Synchronous Asynchronous Receiver/Transmitter) are communication interfaces used for serial communication between devices. RS-232 and RS-485, on the other hand, are specific standards that define the electrical characteristics and physical connectors for serial communication.

UART (Universal Asynchronous Receiver/Transmitter):

- Functionality: UART is a hardware component used for asynchronous serial communication, meaning data is transmitted without the need for a shared clock signal between devices.

- Voltage Levels: UART uses voltage levels typically around 0V for logical 0 and 5V for logical 1 in most cases, but this can vary based on the specific implementation.

- Timing: In UART communication, bits are sent sequentially at a specific baud rate (bits per second). Both the transmitting and receiving devices need to agree on the same baud rate for successful communication.

- Wiring: UART communication uses two wires for communication: one for transmitting (TX) and one for receiving (Rx).

USART: (Universal Synchronous Asynchronous Receiver/Transmitter):

- Functionality: USART can operate both in synchronous and asynchronous modes, giving it more flexibility than UART. In synchronous mode, devices share a common clock signal.

- <u>Voltage Levels:</u> Similar to UART, USART voltage levels depend on the specific implementation but are often around 0V for logical 0 and 5V for logical 1.

- <u>Timing:</u> In asynchronous mode, USART operates similar to UART, with bits sent sequentially at a specific baud rate. In synchronous mode, devices use a shared clock signal.

- <u>Wiring:</u> USART communication also uses two wires for communication: one for transmitting (TX) and one for receiving (RX).

RS-232 (Recommended Standard 232):

- <u>Functionality:</u> RS-232 is a standard for serial communication between devices. It specifies voltage levels, signal timing and connector pinouts.

- <u>Voltage Levels:</u> RS-232 typically uses voltage levels ranging from -15V to +15V for logical 0 and logical 1.

- <u>Timing:</u> RS-232 specifies various parameters, including baud rate, data bits, stop bits, and parity, to ensure proper timing and synchronization between devices.

- <u>Wiring:</u> RS-232 communication uses multiple wires, including TX, RX, ground (GND), and sometimes additional control signals like RTS (Request to Send) and CTS (Clear to Send).

RS-485 (Recommended Standard 485):

- <u>Functionality:</u> RS-485 is a standard for serial communication in a network. It allows for multiple devices to be connected ~~to~~ on the same bus.

- Voltage Levels: RS-485 uses differential signaling, where the voltage difference between two wires represents the logical state. Common voltage levels include +5V for logical 1 and -5V for logical 0.

- Timing: RS-485 supports higher data rates and longer communication distances compared to RS-232. It also supports multi-point Communication, allowing multiple devices to communicate on the same bus.

- Wiring: RS-485 uses two twisted-pair wires for communication: one for transmitting (A/B) and one for receiving (A/B). It also requires termination resistors at both ends of the communication line to prevent signal reflections.

Physical Layer Specifications:

Characteristic	UART	USART	RS-232	RS-485
Voltage levels.	0-5V	0-5V	-3 to +3V	-7 to +12V
Timing	Asynchronous	Synchronous or Asynchronous	Asynchronous	Asynchronous.
Frequencies	Upto 25 Mbps	Up to 25 Mbps	Up to 20 Mbps	Up to 20 Mbps.
Wiring conventions	Single wire	Single wire	Single wire or differential pair	Differential pair.

2. What is flow control in serial communication?

Flow control in serial communication refers to the management of data transmission between devices to prevent data loss or corruption.

In serial communication, data is sent bit by bit over a single channel, which can lead to issues if the sender and receiver operate at different speeds. Flow control mechanisms are implemented to ensure that the sender does not overwhelm the receiver with data, allowing both devices to communicate effectively.

There are two main types of flow control: hardware flow control and software flow control.

1. Hardware Flow Control:

- RTS / CTS (Request to Send / Clear to Send): In RTS/CTS flow control, the sender uses the RTS signal to ask the receiver if it is ready to receive data. The receiver responds with the CTS signal to indicate that it is prepared to accept data. This mechanism prevents data overrun at the receiver's end.

- DTR / DSR (Data Terminal Ready / Data Set Ready): DTR/DSR flow control works similarly to RTS/CTS but uses different signals for communication. DTR is sent by the sender to indicate it is ready, and DSR is sent by the receiver to confirm it is prepared to receive data.

2. Software Flow Control:

- XON / XOFF: XON (transmit on) and XOFF (transmit OFF) are control characters (Ctrl-Q) and Ctrl-S, respectively) used in software flow control. When the receiver's buffer is nearing full, it sends an XOFF character to the sender, indicating that it should stop transmitting data. When the buffer has ~~more~~ room for more data, the receiver sends an XON character to resume data transmission.

The choice of flow control mechanism depends on the devices and protocols being used. Effective flow control ensures that data is transmitted accurately and prevents data loss or corruption due to speed mismatches between sender and receiver.

Examples of serial communication scenarios where flow control is important:

- A computer sending a large file to a printer.
- A modem sending data over a slow phone line.
- A microcontroller sending data to a sensor.
- A device sending data to a server over a wireless network.

In general, hardware flow control is more reliable than software flow control, but it requires additional wires in the serial cable. Software flow control is simpler to implement, but it can be less reliable, especially in noisy environments.

3. Normally a DB-9 connector is used for serial communication. What does DB-9 stand for? What are each of the pins used for? Why are 9 pins needed when serial only needs two lines (TX and RX)?

Ans. DB-9 stands for D-Subminiature, 9-pin connector. It is a type of electrical connector that is commonly used for serial communication. It has a trapezoidal shape and nine pins arranged in two rows, with five pins in the top row and four pins in the bottom row.

Despite serial communication typically requiring only two lines (transmit, TX, and receive, RX), the DB-9 connector includes additional pins for various purposes, such as grounding, handshaking, and other serial communication features.

The description of the pins and their functions in a standard DB-9 serial connector are given below:

1. CD (Carrier Detect): Used to indicate the presence of a connection.
2. RXD (Receive Data): This is where data is received.
3. TXD (Transmit Data): This is where data is transmitted.
4. DTR (Data Terminal Ready): Indicates that the device is ready to establish a connection.
5. GND (Ground): Provides the common ground reference between two devices.
6. DSR (Data Set Ready): Indicates that the device on the other end of the connection is ready to communicate.
7. RTS (Request to send): Indicates the sender is ready to transmit data.
8. CTS (Clear to send): Indicates the receiver is ready to accept data.
9. RI (Ring Indicator): Used to indicate an incoming ring signal.

Although only two lines (TXD and RXD) are essential for basic serial communication, the additional pins (DTR, DSR, RTS, CTS, CD, GND, and RI) are used for more advanced features like hardware flow control, modem communication, and status signalling between devices.

The use of control signals in serial communication helps to improve the reliability and efficiency of data transmission.

In addition to serial communication, DB-9 connectors. are also sometimes used for other purposes, such as connecting ~~to~~ a modem to computer or connecting a GPS receiver to a computer.

4. A GPS receiver chip communicates via UART at 115200 baud. When powered "ON", it sends out a string of data that contains the module status as well as the current GPS location and GPS derived time. (an example of the string is given here: https://www.gpsworld.com/what-exactly-is-gps-nmea-data).

Write pseudocode for a program that:
- reads incoming serial data from the module.
- When a GLL string is received (see string definition here on page 11 (https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual-Rev2.1-Dec07.pdf)
if checksum is valid and status is valid, the the time portion of the string should be sent out over the UART TX pin to another device.

Ans. Consider a sample message:

$GPGGA, 181908.00, 3404.7041778, N, 07044.3966270, W, 4, 13, 1.00, 495.144, M, 29.200, M, 0.10, 0000*40

Observations:
- All parameters sent are comma seperated.
- All parameters can be saved as character strings.
- An array of character strings can be used.
- Checksum can be converted to char form and compared.

<u>Pseudocode</u>:

- Configure UART for transmission and reception on interrupt, with 115200 baud and FIFO enabled.
- Refer https://www.gpsworld.com/what-exactly-is-gps-nmea-data)

Define parameters as necessary, and create a structure containing the same.

- char GPS_pos.
- float time
- float lat_num
- char lat_dir
- float lon_num
- char lon_dir
- int q_ind
- int num_sat
- float hdop
- float alt
- char alt_unit
- float geoidal_sep
- char geoidal_sep-unit
- float age_corr
- int corr_stat_id
- int checksum.

- Define data structure as follows:

[GPS_pos time lat_num lat_dir lon_num lon_dir q_ind num_sat hdop alt alt_unit geoidal_sep-unit age_corr corr_stat_id checksum]

- On UART reception,
    - Keep receiving data as long as "," is not received.
    - If "," is received,
        - Store data received till now in 1st ~~row~~ element of data structure
        - Start storing succeeding data in next element of data structure.
        - Continue till reception is complete.

- Perform checksum on each element of the structure, and add all to form final checksum.

- Compare with received checksum.

    If calculated checksum == checksum
    - Send time on TX pin of UART.