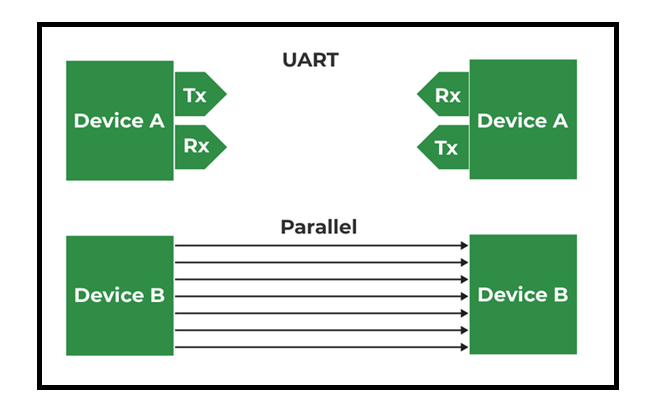
**UART (Universal Asynchronous Receiver-Transmitter)**

A Universal Asynchronous Receiver-Transmitter (UART) is a protocol for asynchronous serial communication in which the data format and transmission speeds are configurable . It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel . The electric signaling levels are handled by a driver circuit external to the UART. Common signal levels are RS-232, RS-485, and raw TTL for short debugging links . A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port . One or more UART peripherals are commonly integrated in microcontroller chips . Specialized UARTs are used for automobiles, smart cards, and SIMs . A related device, the universal synchronous and asynchronous receiver-transmitter (USART) also supports synchronous operation . In OSI level terms, UART lives on layer 2, the Data link layer . Transmitting and receiving serial data A UART contains those following components: a clock generator, usually a multiple of the bit rate to allow sampling in the middle of a bit period input and output shift registers, along with the transmit/receive or FIFO buffers transmit/receive control read/write control logic . The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels, which may be standardized voltage levels, current levels, or other signals . Communication may be 3 modes: simplex (in one direction only, with no provision for the receiving device to send information back to the transmitting device) full duplex (both devices send and receive at the same time) half duplex (devices take turns transmitting and receiving) . For UART to work the following settings need to be the same on both the transmitting and receiving side: Voltage level, Baud Rate, Parity bit, Data bits size, Stop bits size, and Flow Control . UART is mostly used in IoT security.

**UART Basics**

UART is a Universal Asynchronous Receiver Transmitter protocol that is used for serial communication. Two wires are established here in which only one wire is used for transmission whereas the second wire is used for reception. Data format and transmission speeds can be configured here. So, before starting with the communication define the data format and transmission speed. Data format and transmission speed for communication will be defined here and we do not have a clock over here that’s why it is referred to as asynchronous communication with UART protocol. Here we will see how this protocol is designed physically.



Here, DEVICE A that is having transmitter PIN and a receiver pin; DEVICE B is having a receiver and transmission pin. The Transmitter of DEVICE A is one that should be connected with the receiver pin of DEVICE B and the transmitter pin of DEVICE B should be connected with the receiver pin of DEVICE A we just need to connect two wires for communication.

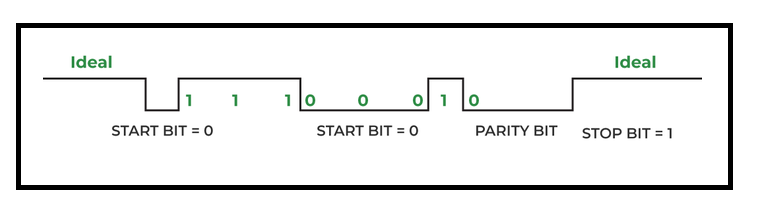
If DEVICE A wants to send data, then it will be sending data on the transmitter’s pin and here receiver of this DEVICE B will receive it over and if DEVICE A wants to receive the data, then that is possible on the RX line that will be forwarded by TX of DEVICE B. On comparing this serial communication of UART with parallel then it can be observed that in parallel multiple buses are required. Based on the number of lines bus complexity of UART is better but parallel communication is good in terms of speed.

So, when speed is required at that time we should select parallel communication and for a low-speed application, UART must be used and the bus complexity will be less.

The configuration of UART is done before transmission, both of these devices are connected with protocol and should know the speed of data transmission. First, define the speed of both devices. Now, configure the speed of DEVICE A and B for data transmission which is referred to as Baud Rate so here Baud Rate will be the same for DEVICE A and B otherwise both of these devices cannot understand at what speed and at what rate data is coming. After that, configure the data length so here DEVICE A and DEVICE B both are configured at fixed data length if DEVICE A is transmitting data, then it is configured with fixed data. Like if DEVICE A is configured with the eight-bit size of data then DEVICE B should also be configured at the same size of data which is eight bits. After this, check data transmission or receiving time, forward start bits, and stop bits.

Now we will see the data format and when the communication is according to UART protocol. We are using NRZ encoding for data communication.

### UART Data Format



 DEVICE A is sending data to DEVICE B, and the transmitter of DEVICE A will send data to the receiver of Device B then it will be logic high. Now, send the start bit that will be logic 0 and once we have the start bit, DEVICE B will know that somebody is communicating. Now there is the same speed configuration with both devices. So, after the start bit, DEVICE A can forward data.

Consider 8 bits of data length, so we will be forwarding 8 bits and those 8 bits will be received by DEVICE B a parity bit can also be used which is optional, but this is quite effective. By using the parity bit, it can be identified whether the received data is correct or not. Suppose we are sending 1 1 1 0 0 0 1 0. Now, we have 4 ones; an even number of ones are there hence the parity is even and for that, logic 0 will be assigned. Suppose we are receiving data with some error, say zero is converted into one; Now incorrect data that is 1 1 1 1 0 0 1 0 for this incorrect data parity will be 0 as there are 5 ones, here is a mismatch in the parity bit and hence it is confirmed that the received data has some error.

### Pros of UART Protocol

* It is having less physical interfacing based on only two lines.
* Simple to configure data and data size. Speed is also configurable. In the majority of cases, this baud rate is 9600 for the UART protocol. Full duplex mode configuration is possible by using two wires so that is the major advantage of UART.
* Error detention is possible

### Cons of UART Protocol

* UART is having serial communication, therefore, it has less speed.
* Start bit, stop bit, and the parity bit is other overhead.
* Since this is asynchronous communication so here there are many things that we need to do in configuration, for instance, we should configure both devices at the same speed because the clock signal is absent.

**SPI (Serial Peripheral Interface)**

SPI protocol is the communication protocol that is used for connecting low-speed devices and the devices can be RFID card readers, ADC, 2.4 GHz wireless transmitters and receivers, and memory card, readers. All these devices work with SPI protocol to establish communication with microcontrollers. This is the full-duplex synchronous serial communicating protocol where the information transmission happens concurrently from both directions.

The unique feature of this protocol is that the information transmission takes place without any intermissions. It follows a continuous stream for data transfer of ‘n’ number of bits.

**Interfacing of SPI Protocol:**

Communication through SPI protocol takes place in a master-slave relationship. The Master acts as a regulatory device and the slave works as per the instructions received from the master. The easier configuration in SPI is a single master and single slave system whereas one master system holds the ability to control more slaves as well.

In a few scenarios, SPI is also termed as a 4-wired serial communication bus. It also has the feature of differential signaling and it offers a single simple communication way. The 4-wired serial peripheral interface protocol works with four signals which are:

Chip selection signal/Slave selection

MOSI – Master Out Slave In

MISO – Master In Slave Out

**Clock**

The clock signal is termed as serial clock SCK/SCLK which is generated by the master. The data transmission between slave and master is matched with this clock signal. The SPI devices have the capability of supporting a broad range of clock frequencies than that of I2C protocols. The interfacing between master and slave can be done as

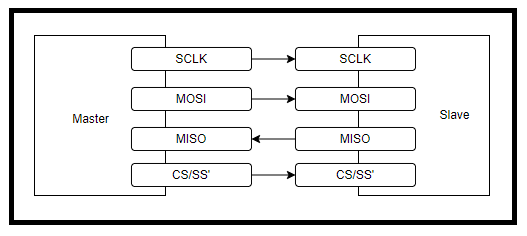
One master – One slave

One master – Multiple slaves

The slave selection is done by the chip select signal/slave select signal where which is an active LOW signal and moves to HIGH when the slave gets disconnected from the bus. When the device works in the mode of one master and multiple slaves, then every slave should need separate chip select signals from the master.

The MISO and MOSI are the data transmission lines where MOSI has the functionality of transmitting data from master to slave and MISO has the functionality of transmitting data from slave to master. In the cases of slaves only without a master, the MOSI is termed as Serial Data In (SDI) and MISO is termed as Serial Data Out (SDO).

**SPI Architecture:**



**Working of Serial Peripheral Interface**

As discussed that SPI device can work with one slave or multiple slaves, but has only a master. Let us look at the detailed operation and data transmission in this protocol.

**CLOCK Signal**

The CLK synchronizes the master output with the sampling bits of a slave. For every clock cycle, a single bit of data will be transmitted where the data transmission speed is known as the clock signal frequency. Always, the communication in SPI devices is started by the master as it generates the clock signal.

The CLK in the SPI device can be altered by clock phase (CPHA) and clock polarity (CPOL) properties. Apart from setting the clock frequency, the master should also synchronize CPHA and CPOL corresponding to the data.

Clock polarity decides the clock signal polarity. Using an inverter, the polarities can be altered. CPOL indicates that the clock signal’s base value is ‘0’ so that the idle state is ‘0’ and the active is ‘1’.

When the clock edge is ‘0’, data transmission happens at the time of the LOW to HIGH transition. This indicates leading and trailing edges correspond to rising and falling edges.

When the clock edge is ‘1’, data transmission happens at the time of the HIGH to LOW transition. This indicates leading and trailing edges correspond to falling and rising edges.

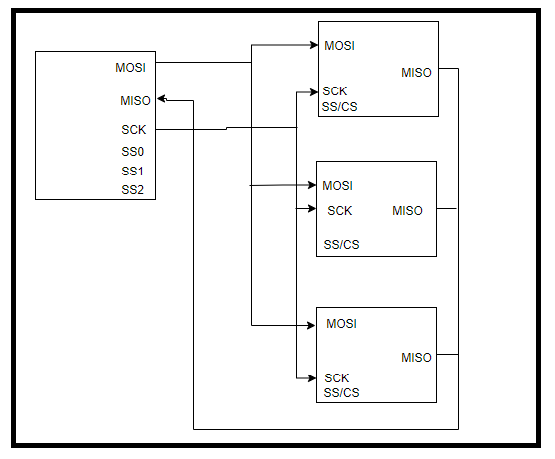
The clock phase decides the data timing corresponding to clock pulses. CPHA indicates that the clock signal’s base value is ‘1’ so that the idle state is ‘1’ and the active is ‘0’.

When the clock edge is ‘0’, data transmission happens at the time of the HIGH to LOW transition.

When the clock edge is ‘1’, data transmission happens at the time of the LOW to HIGH transition.

**Slave Select (SS)**

Master can select to which slave it can start communication by making slave select signal to voltage level ‘0’ on the select line. In the idle/no-transmission state, the slave select signal moves to voltage level ‘1’. When there are multiple slave select pins, then multiple slaves are in parallel connection. Whereas when a single slave select pin is present, then multiple slaves are connected to the master in daisy chain format. The below picture shows how slave select pins are connected.



**Serial Peripheral Interface With Multiple Slaves**

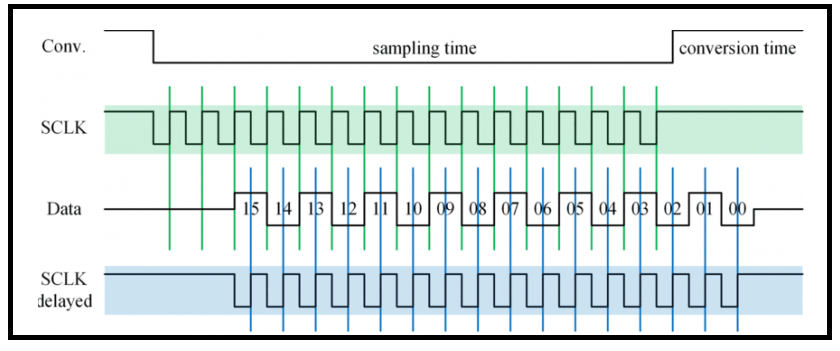
**Data Transmission in SPI**

Initially, the master configures the clock to the frequency level (few MHz) which is supported by the slave. Then, the master chooses the slave device by making voltage level ‘0’ on the selection line. When there is a need for analog to digital conversion, then there will be a waiting period so the master has to wait before sending the clock cycles.

At every clock cycle, transmission happens in a full-duplex way where transmits one-bit using the MOSI line and slave reads that bit. In the same slave transmits one bit using the MISO line and the master reads that bit. This approach is continued even for one-directional data transmission.

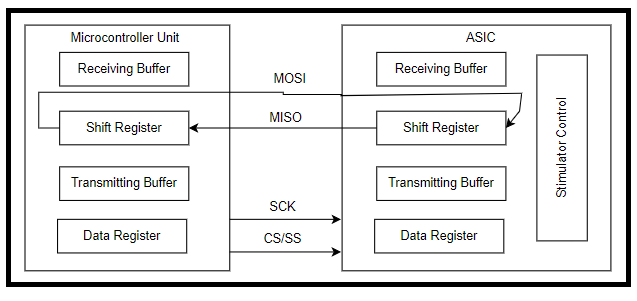
Data transmissions usually require two 8-bit shift registers one for the master and the other for the slave where these registers are connected in the form of the virtual ring. Data transmission from master to slave initially happens with MSB bit. On clock edge, both the slave and master send out a bit through the transmission line. On the successive clock edge, at the receiver end, the transmitted bit gets sampled from the transmission line and sets as a new LSB of the shift register.

The SPI protocol timing diagram for data transmission is as follows:



When all the register bits are shifted in and out, it represents the shift register values that are exchanged between slave and master. When there is a requirement for the exchange of more information, then the reload takes place in shift registers, and the operation iterates. When the transmission completes, the master stops switching the CLK signal and deselects the slave.

This functionality is explained in the below SPI protocol block diagram.



**SPI Protocol Block Diagram**

**Serial Peripheral Interface Specifications**

* The clock frequency range of the SPI protocol is a maximum of 500 kHz.
* The data transmission time at a frequency of 500 kHz is 38 µ
* The digital output load at a frequency of 500 kHz is a maximum of 1 nF.
* The internal analog to digital conversion is 150 µ

**I2C**

I2C stands for Inter-Integrated Circuit. It is a bus interface connection protocol incorporated into devices for serial communication. It was originally designed by Philips Semiconductor in 1982. Recently, it is a widely used protocol for short-distance communication. It is also known as Two Wired Interface(TWI).

**Working of I2C Communication Protocol :**

It uses only 2 bi-directional open-drain lines for data communication called SDA and SCL. Both these lines are pulled high.

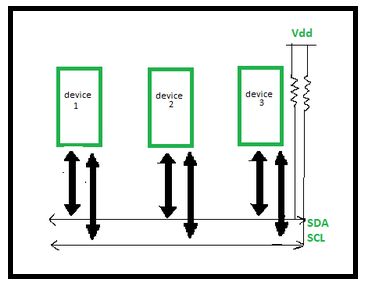
Serial Data (SDA) – Transfer of data takes place through this pin.

Serial Clock (SCL) – It carries the clock signal.

I2C operates in 2 modes –

* Master mode
* Slave mode

Each data bit transferred on SDA line is synchronized by a high to the low pulse of each clock on the SCL line.



According to I2C protocols, the data line can not change when the clock line is high, it can change only when the clock line is low. The 2 lines are open drain, hence a pull-up resistor is required so that the lines are high since the devices on the I2C bus are active low. The data is transmitted in the form of packets which comprises 9 bits. The sequence of these bits are –

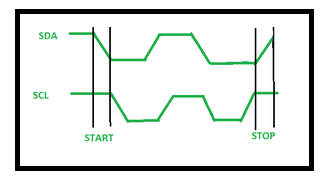
Start Condition – 1 bit

Slave Address – 8 bit

Acknowledge – 1 bit

**Start and Stop Conditions :**

START and STOP can be generated by keeping the SCL line high and changing the level of SDA. To generate START condition the SDA is changed from high to low while keeping the SCL high. To generate STOP condition SDA goes from low to high while keeping the SCL high.



**Repeated Start Condition :**

Between each start and stop condition pair, the bus is considered as busy and no master can take control of the bus. If the master tries to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a new START condition. It is called a REPEATED START condition.

**Read/Write Bit :**

A high Read/Write bit indicates that the master is sending the data to the slave, whereas a low Read/Write bit indicates that the master is receiving data from the slave.

**ACK/NACK Bit :**

After every data frame, follows an ACK/NACK bit. If the data frame is received successfully then ACK bit is sent to the sender by the receiver.

**Addressing :**

The address frame is the first frame after the start bit. The address of the slave with which the master wants to communicate is sent by the master to every slave connected with it. The slave then compares its own address with this address and sends ACK.

**I2C Packet Format :**

In the I2C communication protocol, the data is transmitted in the form of packets. These packets are 9 bits long, out of which the first 8 bits are put in SDA line and the 9th bit is reserved for ACK/NACK i.e. Acknowledge or Not Acknowledge by the receiver.

# CAN (Controller Area Network) protocol

CAN stands for **Controller Area Network** protocol. It is a protocol that was developed by **Robert Bosch** in around 1986. The CAN protocol is a standard designed to allow the microcontroller and other devices to communicate with each other without any host computer. The feature that makes the CAN protocol unique among other communication protocols is the broadcast type of bus. Here, broadcast means that the information is transmitted to all the nodes. The node can be a sensor, microcontroller, or a gateway that allows the computer to communicate over the network through the USB cable or ethernet port. The CAN is a message-based protocol, which means that message carries the message identifier, and based on the identifier, priority is decided. There is no need for node identification in the CAN network, so it becomes very easy to insert or delete it from the network. It is a serial half-duplex and asynchronous type of communication protocol. The CAN is a two-wired communication protocol as the CAN network is connected through the two-wired bus. The wires are twisted pair having 120Ω characteristics impedance connected at each end. Initially, it was mainly designed for communication within the vehicles, but it is now used in many other contexts. Like UDS, and KWP 2000, CAN also be used for the on-board diagnostics.

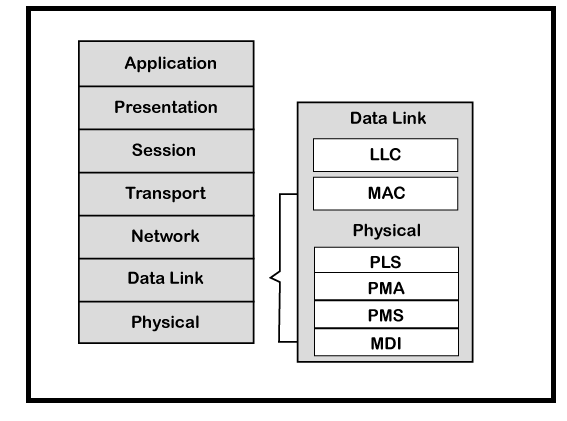
**Applications of CAN protocol**

Initially, CAN protocol was designed to target the communication issue that occurs within the vehicles. But later on, due to the features it offers, it is used in various other fields. The following are the applications of CAN protocol:

* Automotive (passenger vehicles, trucks, buses)
* Electronic equipment for aviation and navigation
* Industrial automation and mechanical control
* Elevator and escalators
* Building automation
* Medical instruments and equipment
* Marine, medical, industrial, medical

**CAN layered architecture**

As we know that the OSI Module partitions the communication system into 7 different layers. But the CAN layered architecture consists of two layers, i.e., data link layer and physical layer.



* **Data-link layer**: This layer is responsible for node to node data transfer. It allows you to establish and terminate the connection. It is also responsible for detecting and correcting the errors that may occur at the physical layer. Data-link layer is subdivided into two sub-layers:
  1. **MAC:** MAC stands for Media Access Control. It defines how devices in a network gain access to the medium. It provides Encapsulation and Decapsulation of data, Error detection, and signaling.
  2. **LLC:** LLC stands for Logical link control. It is responsible for frame acceptance filtering, overload notification, and recovery management.
* **Physical layer:** The physical layer is responsible for the transmission of raw data. It defines the specifications for the parameters such as voltage level, timing, data rates, and connector.

CAN specifications define CAN protocol and CAN physical layer, which are defined in the CAN standard [ISO](https://www.javatpoint.com/iso-full-form) 11898. ISO 11898 has three parts:

* ISO 11898-1: This part contains the specification of the Data-link layer and physical signal link.
* ISO 11898-2: This part comes under CAN physical layer for high speed CAN. The high- speed CAN allows data rate upto 1 Mbps used in the power train and the charges area of the vehicle.
* ISO 11898-3: This part also comes under CAN physical layer for low-speed CAN. It allows data rate upto 125 kbps, and the low speed CAN is used where the speed of communication is not a critical factor.

CiA DS-102: The full form of CiA is CAN in Automation, which defines the specifications for the CAN connector.

As far as the implementation is concerned, the CAN controller and CAN transceiver are implemented in the software with the help of the application, operating system, and network management functions.