

IoT App Development



UART Serial Communication

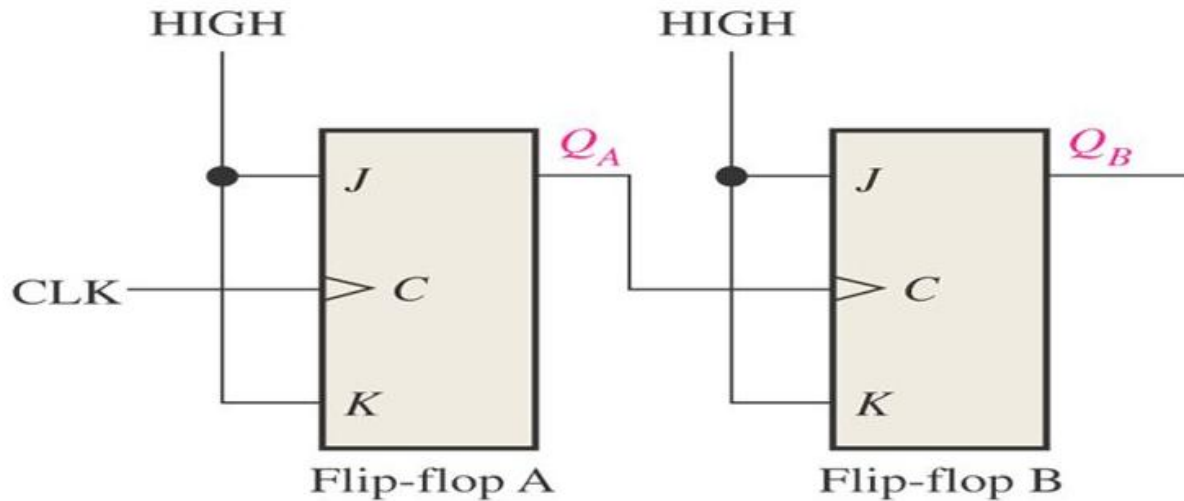
Dr. Manas Khatua

Associate Professor, Dept. of CSE, IIT Guwahati

E-mail: manaskhatua@iitg.ac.in

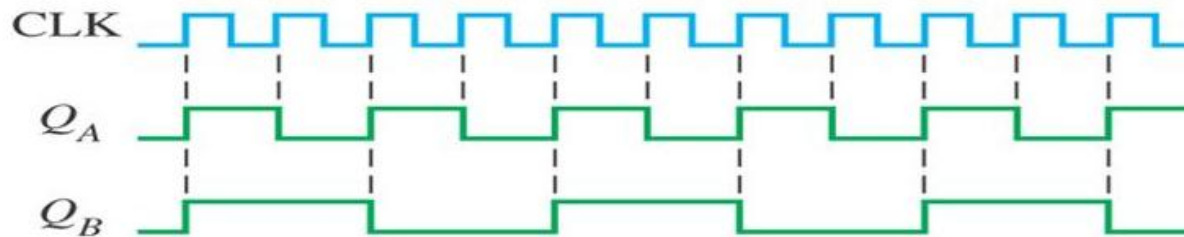
- UART stands for **Universal Asynchronous Receiver Transmitter**
 - one of the earliest modes of communication applied to computers
 - its origin goes back at least as far as the 1960s
 - called '**universal**' because
 - its parameters - speed, data size and so on - are not fixed, and can be configured, however, both side should agree
 - It is used for **asynchronous serial communication**
 - **Serial communication** stands for the process of sending data **one bit at a time** sequentially
 - **Asynchronous** means there is **no clock signalling line to synchronize**, and transmitter and receiver might **turn on at different time instant**
 - Two devices do not necessarily share a common clock.
 - Both systems might for example agree on some fixed Baud rate.
- Baud rate is a measure of the speed of data transfer, expressed as symbol per sec.
 - Also represented by bit per second (bps).
 - Bit Rate = Baud rate x the number of bit per baud
 - Common baud rates: 9600, 4800, 19200, 38400, etc.

Clock based Synchronization



Truth Table

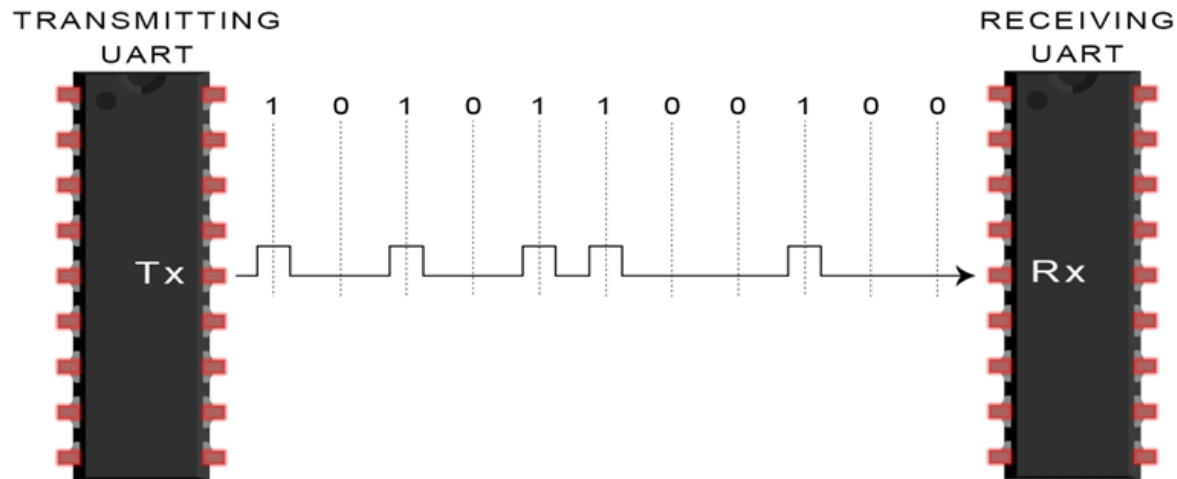
CLK	J	K	Q_{n+1}
↑	0	0	Q_n
↑	0	1	0
↑	1	0	1
↑	1	1	Q_n'



Source: Google Image

Synchronization in UART

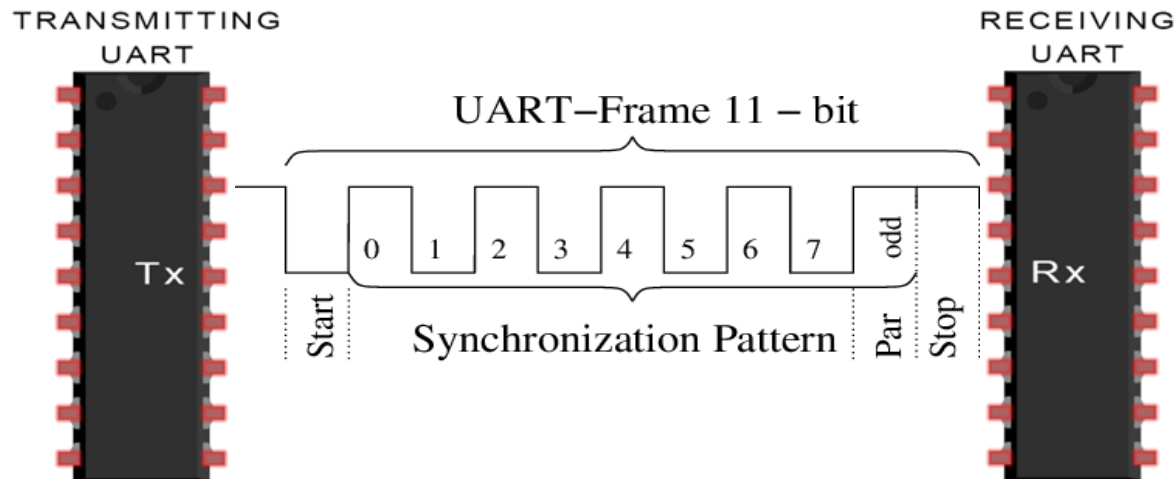
- This **synchronisation** is done in the form of a **high to low** (or low to high) **transition** on the **data line**.
 - The line will ideally **high** in idle
 - then **drop low** when transmission starts.
 - This transition acts to **synchronise** the timing between both devices.
 - The receiver then knows to **clock in enough cycles of data** based on its baud clock.



Synchronization in UART

- There are **two problems** with the data line based synchronization scheme:
 - 1) The **single high to low transition is not enough** to synchronise timing over a long period of time.
 - So, they need **periodic re-synchronisation**.
 - 2) If the **first bit of data** it wants to send is also represented by a **high level**, then there are no transition, and vice-versa.
 - So, they need some way to **distinguish the first bit**.

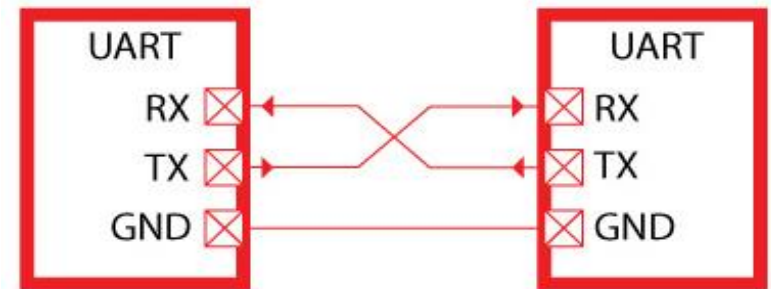
Both of these problems are solved in UART by using **start** and **stop bits**.



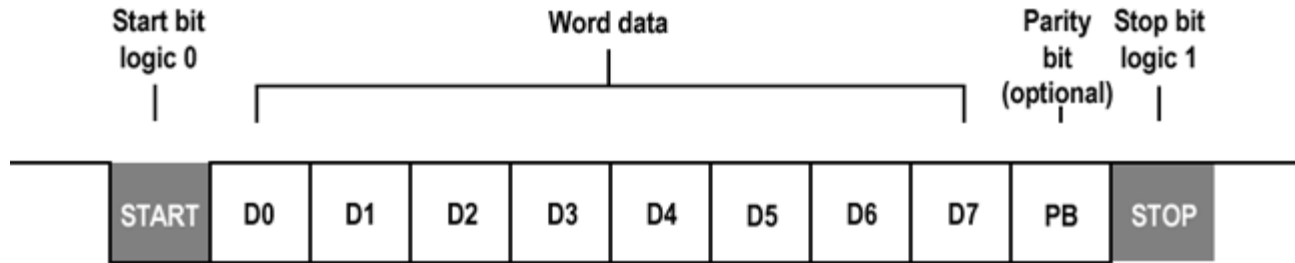
UART Communication

➤ UART Communication Properties

- Communication may be simplex, full duplex, or half duplex
 - ✓ **Simplex**: One direction only, transmitter to receiver
 - ✓ **Half Duplex**: Devices take turns transmitting and receiving
 - ✓ **Full Duplex**: Devices can send and receive at the same time
- Data **format** and transmission **speed** are **configurable**
- Communication goes through the **two independent lines/wires**
 - ✓ **TX** (transmission)
 - ✓ **RX** (reception).



Data Framing

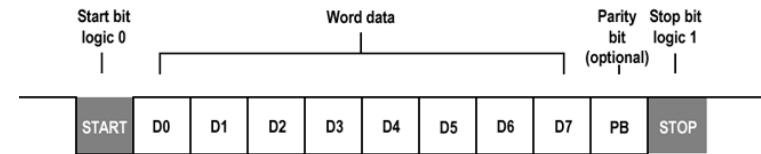


Start Bit:

- The UART transmission line is normally held at a **high voltage level when idle** i.e. it is not transmitting data
- Its **start bit** is set to **logic low** as it is used to signal the receiver that a new framing is coming.

Stop Bits:

- To signal the end of the data packet, the sending UART drives the data transmission line **from a low voltage to a high voltage**.



Parity:

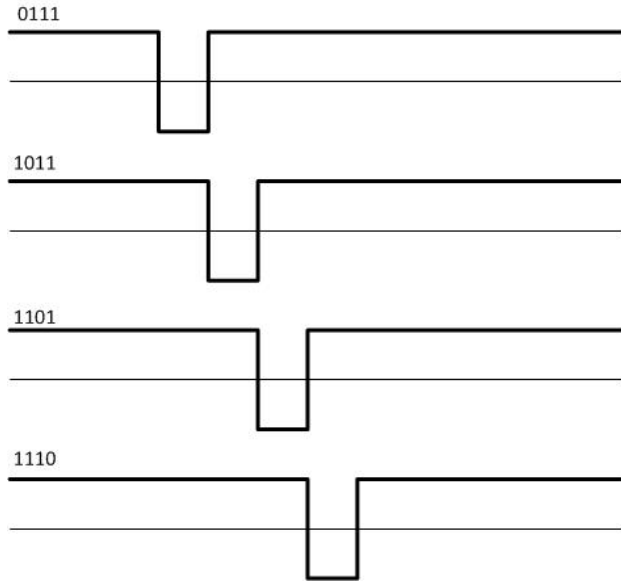
- It is an optional bit.
- It is used to detect the wrong data packets, i.e. error during transmission
- Two type of parity bit used: **even parity**, **odd parity**.
 - **Even parity:**
 - If the count of bits with value 1 is odd, the parity bit value is set to 1, **making the total count of occurrences of 1s** in the whole set (including the parity bit) an **even** number.
 - If the count of 1s in a given set of bits is already even, the parity bit's value is 0.
 - **Odd parity:**
 - the coding is reverse of even parity.

Data Frame:

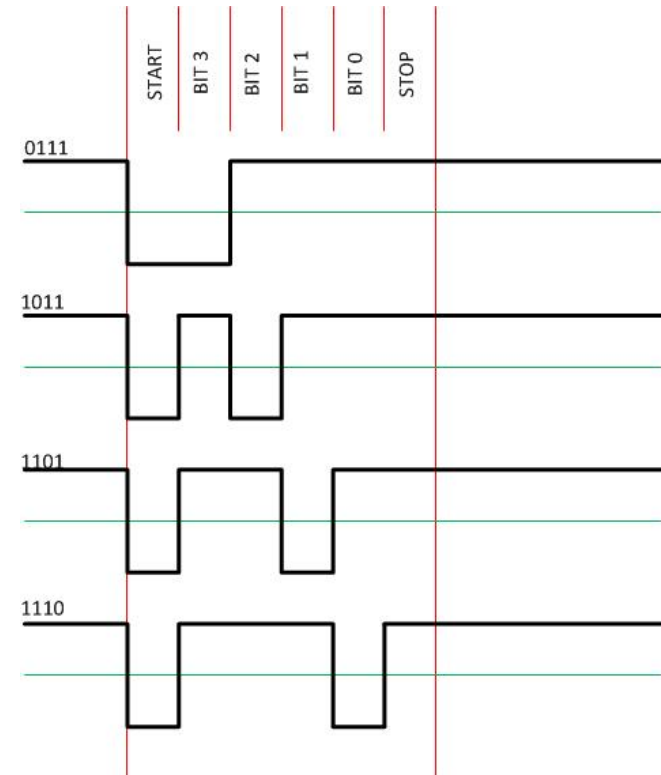
- If **one parity bit** is used, then next 5-8 bits carry the “**data bits**” (information). Otherwise, data bits can be 5-9 bits.
- Certainly, the **standard data size is 8-bit**, but other sizes have their uses.
- A 7-bit data chunk can be more efficient than 8, especially if you're just transferring 7-bit ASCII characters.

Use of Start bit

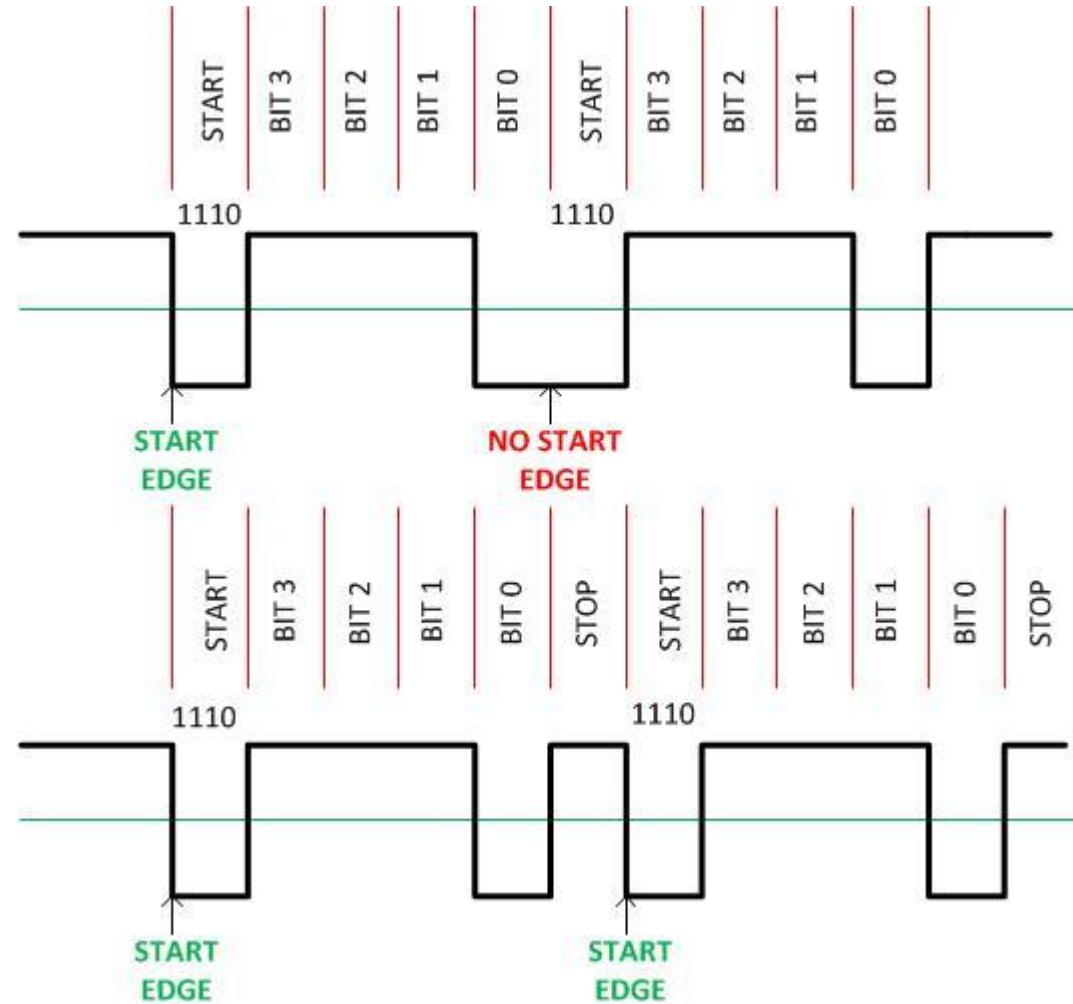
As time matters with asynchronous signals, it is needed to send a "LISTEN UP" signal in the form of start bit.



Without any time reference, they all look alike as a voltage signal.



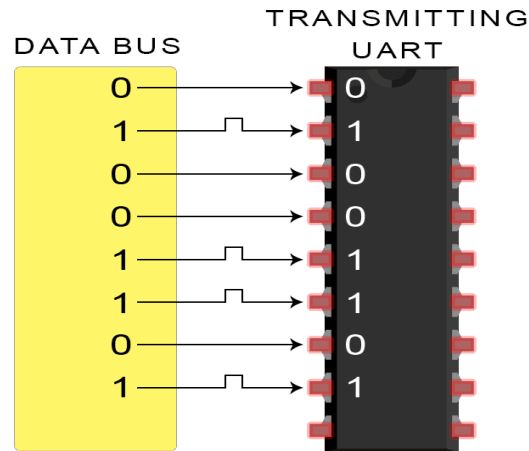
Use of Stop bits



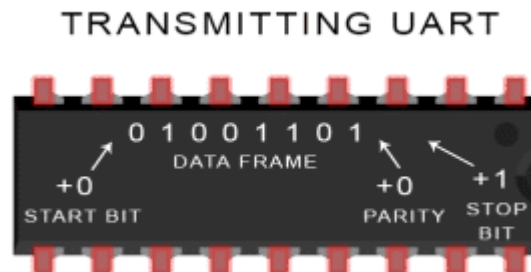
- In the first figure, **no transition for start bit !**
- So, the **stop bit ensures** the line goes back to the idle state at the end of the transmission.
- It also gives the receiver a time window to handle the bytes just received, reset, and get ready to listen for the next start bit.
- **Note:** Start & Stop bits are used for synchronization, but not to define the data length

Steps of UART Communication

1. The transmitting UART receives data in parallel from the data bus:



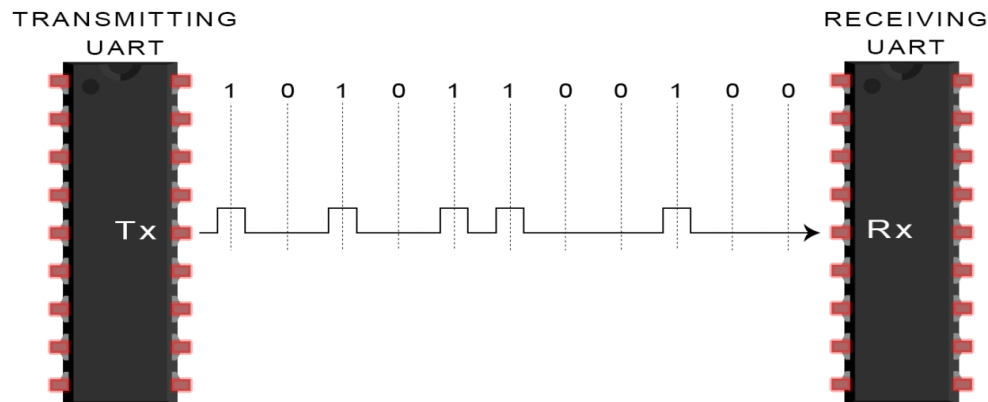
2. The transmitting UART adds the start bit, (even) parity bit, and stop bit(s) to the data frame:



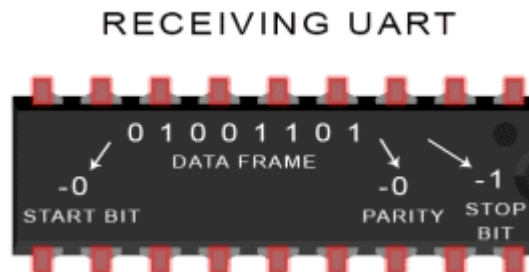
Cont...

3. The entire packet is **sent serially** from the transmitting UART to the receiving UART. In most cases, the data is sent with the least significant bit first.

The receiving UART **samples** the data line **at the pre-configured baud rate**:

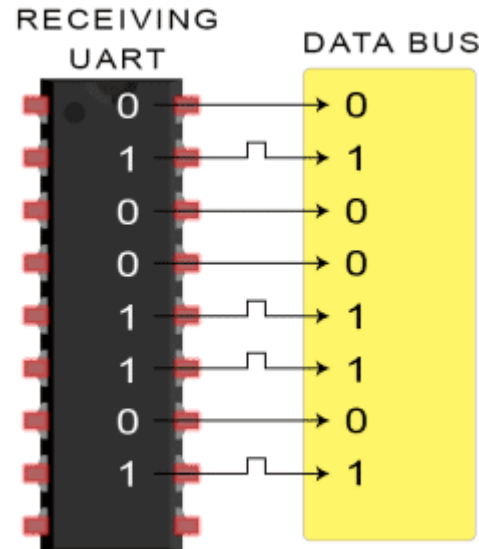


4. The receiving UART **discards** the **start bit**, **parity bit**, and **stop bit** from the data frame:



Cont...

5. The receiving UART **converts the serial** data back **into parallel** and transfers it to the data bus on the receiving end:



Note:

- For communication between the devices, both of them have to have **same baud rate**
 - The baud rate between the transmitting and receiving UARTs can only **differ by max 10%**.

Basic Serial Setting



- *Example: 9600 8N1* (9600 baud, 8 data bits, no parity, 1 stop bit)
 - It is one of the most commonly used serial protocols.
- A device transmitting the ASCII characters 'OK' would have to create two packets of data.
- The ASCII value of O (that's uppercase) is 79, which breaks down into an 8-bit binary value of 01001111, while K's binary value is 01001011.



- Since we're transferring at 9600 bps, the **time spent holding each of those bits high or low** is $1/(9600 \text{ bps})$ or $104 \mu\text{s}$ per bit.
- For every byte of data transmitted, there are actually 10 bits being sent: a start bit, 8 data bits, and a stop bit.
- So, at 9600 bps, **we're actually sending** 9600 bits per second or **960 (9600/10) bytes per second**.

Advantages and Disadvantages



Advantages:

- 1) Well documented and **widely used** method.
- 2) Only uses **two wires**
- 3) **No clock** signal is necessary
- 4) Has a parity bit to allow for **error checking**
- 5) The **structure** of the data packet can be changed as long as both sides are set up for it.

Disadvantages:

- 1) The **size** of the dataframe is limited to max 9 bits.
- 2) Doesn't support multiple **master-slave** systems
- 3) The **baud rate** of each UART must be within 10% of each other.

Applications and Standards



- UART is normally used in microcontrollers for exact requirements
- Today, UART is being used in many applications like
 - GPS Receivers,
 - Bluetooth Modules,
 - GSM and GPRS Modems,
 - Wireless Communication Systems,
 - RFID based applications
 - etc.
- The first of the serial data transmission standards was RS232, or more correctly RS-232 or COM-port. This was developed in 1962.
- There are more recent standards which allow high speed data transmission along with multiple transmitters and receivers
 - e.g. USB, DP, HDMI, DDR, Ethernet, PCI Express, SAS/SATA, Thunderbolt, etc.

Lessons Learned



- ✓ What is serial communication
 - ✓ What is asynchronous communication
 - ✓ UART communication mechanism
 - ✓ Pros and Cons of UART communication
 - ✓ Applications of UART communication
-

Thanks!



Synchronization in UART



- This **synchronisation** is done in the form of a **high to low** (or low to high) **transition** on the **data line**.
 - The line will ideally **high** in idle
 - then **drop low** when transmission starts.
 - This transition acts to **synchronise** the timing between both devices.
 - The receiver then knows to **clock in enough cycles of data** based on its baud clock.
- However, there are **two problems** with this scheme:
 - 1) The **single high to low transition is not enough** to synchronise timing over a long period of time.
 - So, they need **periodic re-synchronisation**.
 - 2) If the **first bit of data** it wants to send is also represented by a **high level**, then there are no transition, and vice-versa.
 - So, they need some way to **distinguish the first bit**.