Universal Asynchronous Receiver Transmitter

Yogesh M Iggalore

Agenda

Synchronous & Asynchronous Serial Transmission

What is UART?

Difference between RS232 and UART

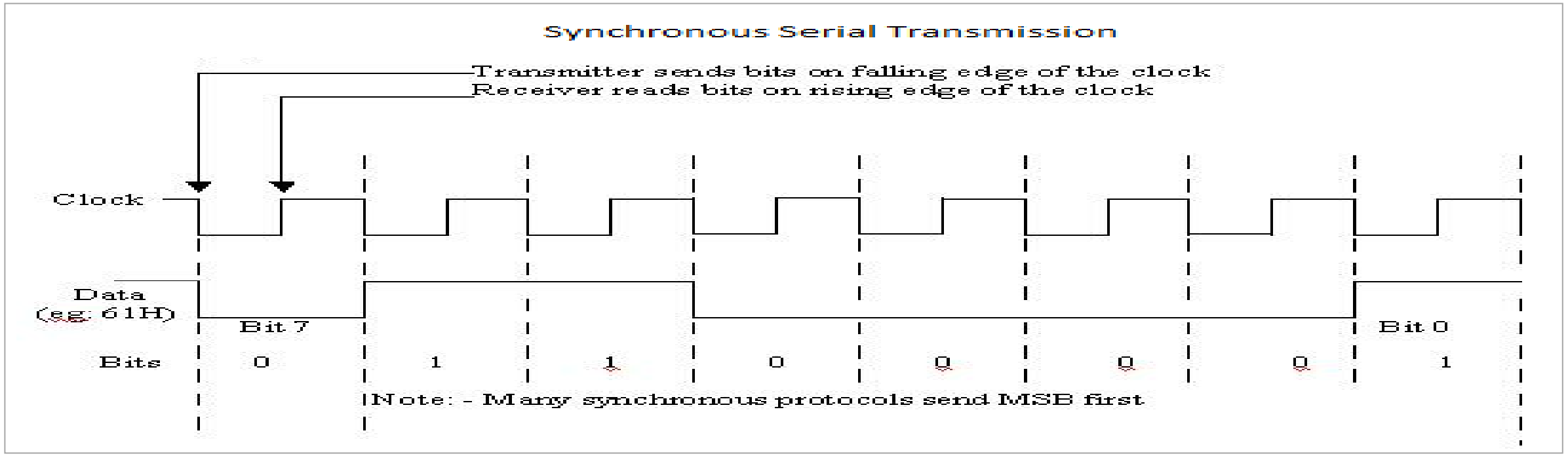
UART – Errors

ESP8266 UART

Project

Synchronous Transmission

Synchronous serial transmission requires that the sender and receiver share a common clock “or” that the sender provide a strobe or other timing signal so that the receiver knows when to “read” the next bit of the data is usually more efficient because only data bits are transmitted between sender and receiver Most of the serial and almost all parallel communication devices use synchronous transfers only

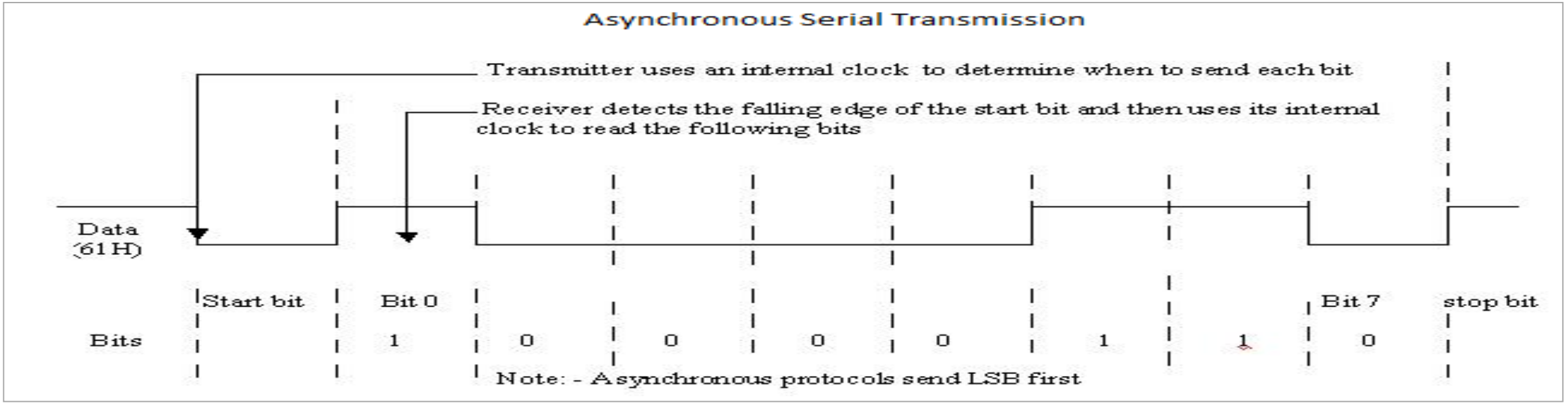


Asynchronous Transmission

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver

The sender and receiver must agree on timing parameters in advance

Special bits are added to each word which are used to synchronize the sending and receiving units



Synchronous vs. Asynchronous transmission

	Synchronous transmission	Asynchronous transmission
Advantages	Lower overhead and thus, greater throughput	Simple, doesn't require synchronization of both communication sides Cheap, timing is not as critical as for synchronous transmission, therefore hardware can be made cheaper Set-up is faster than other transmissions, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard
Disadvantages	Slightly more complex Hardware is more expensive	Large relative overhead, a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information

What is UART?

Universal Asynchronous Receiver/Transmitter

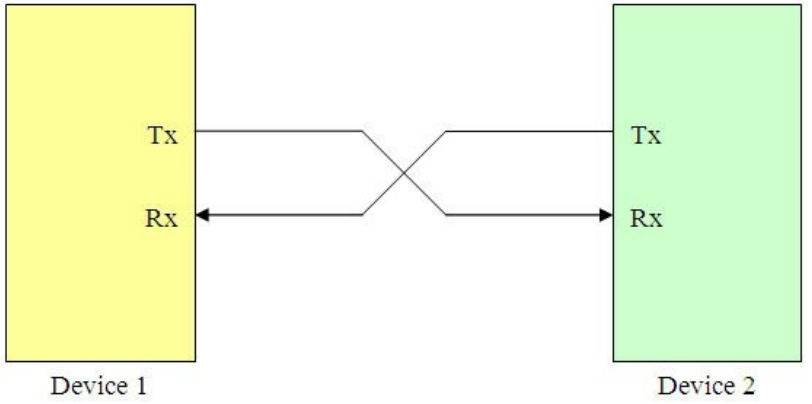
The *universal* designation indicates that the data format and transmission speeds are configurable

Devices communicate in 1:1 mode

One wire for each direction of data (TxD, RxD)

Supports full-duplex operation (TxD and RxD works independently and is not synchronized)

- Typical speeds (in bps) are 1200, 2400, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1382400



UART - Data transmission

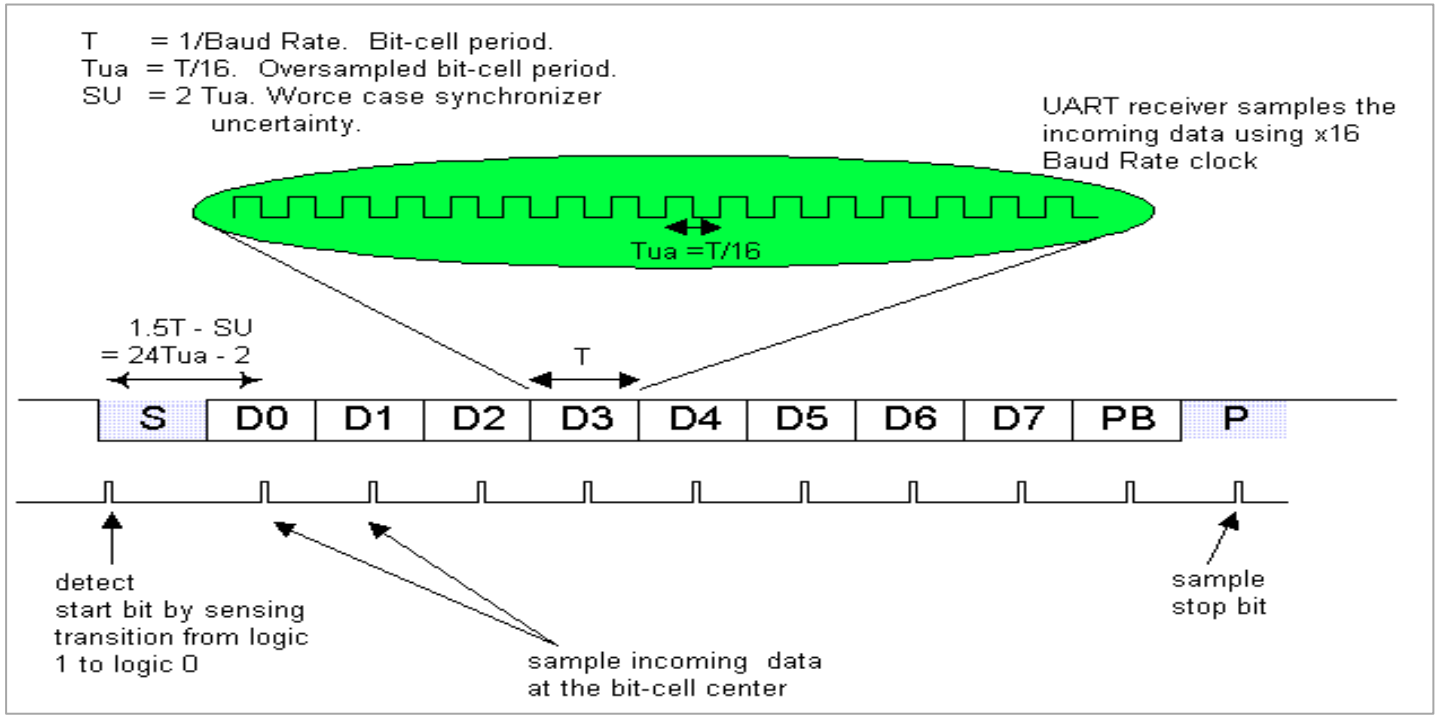
The idle or no data state is logic high

Each byte is sent as a logic low start bit
- sampled immediately by receiver

Then the data bits
- usually 8, but configurable
- typically LSB first
- sampled at the center of the bit time

An optional parity bit

One or more stop bits - logic high



UART Clock Requirements

A UART needs a clock input for bit timing

UART baud rates are usually much lower than the MCU system clock, so the system clock cannot be directly used as the UART clock

Typically Timers/clock dividers are used to generate the UART baud rate by dividing down the system clock

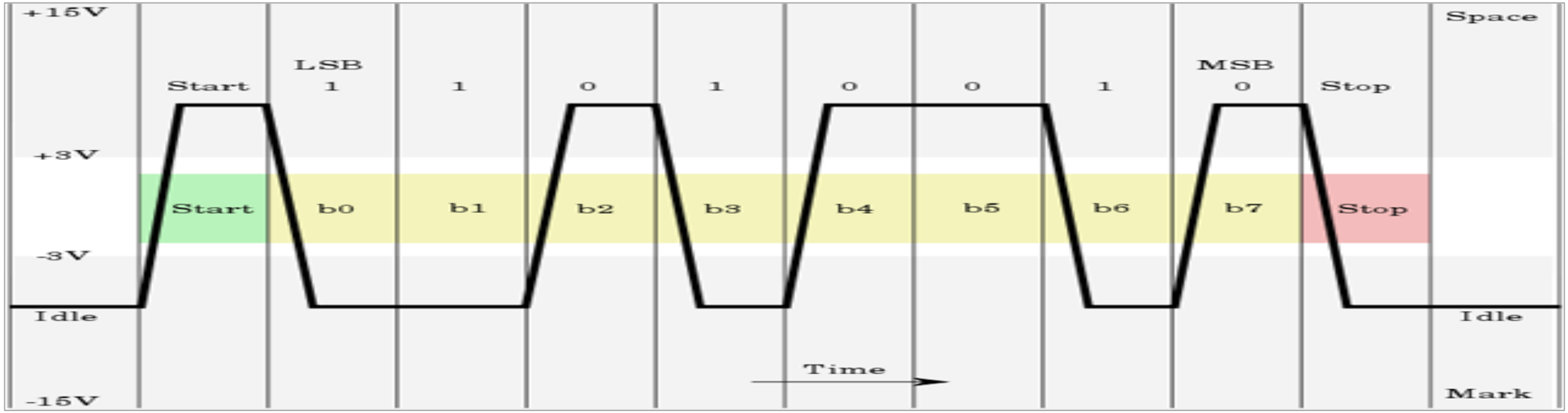
Example: MCU system clock—24 MHz; UART baud rate—57600

A bit time accuracy of 2.5% or better is required at both the transmitter and receiver ends to be able to communicate without errors

- Ensure that the system clock is ±2 % max over PVT (process, voltage and temperature)

RS-232

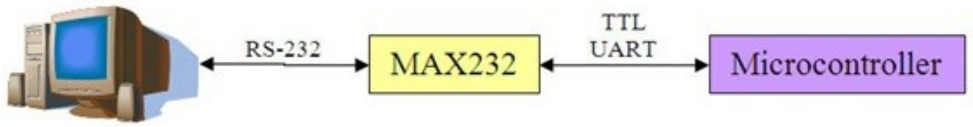
RS-232 (Recommended Standard 232) is a standard for serial binary data signals connecting between a Data Terminal Equipment (DTE, eg Computer) and a Data Communication Equipment (DCE, eg: Modem). Commonly used in computer serial ports. One major difference between TTL level UART and RS-232 is the voltage level Valid signals in RS-232 are ±3 to ±15V, and signals, near 0V is not a valid RS-232 level.



Difference between RS232 and UART

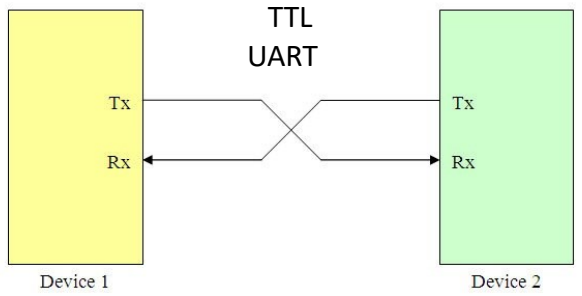
RS232

- a specification for serial communications between a Data Communication Equipment(DCE) and Data Terminal Equipment (DTE) (eg, computer and modem);
- it defines electrical characteristics
- has different voltage levels than UART
- valid signals are plus or minus 3 to 15 volts
- specifies various handshaking signals
- specifies the standard D-connector pinouts



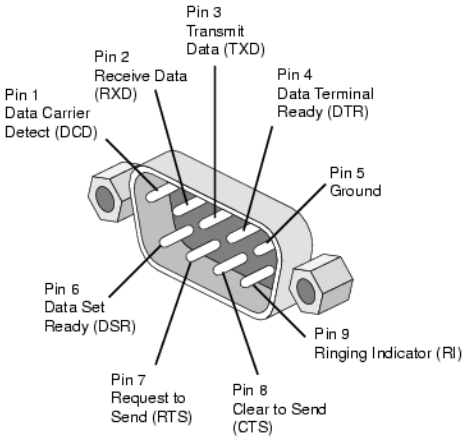
UART

- is an electronic circuit which handles communication over an asynchronous serial interface
- UART is only used on PCB level, to communicate between processors
- or to be extended outside the PCB through a transceiver, such as for example a RS-232 transceiver.
- Typically the voltage levels are 0 and 5 (or Vdd)



RS232 - Signals

Signal			Origin	
Name	**Typical purpose**	**Abbrevia-tion**	**DTE**	**DCE**
Data Terminal Ready	Indicates presence of DTE (PC) to DCE (Modem).	DTR	●	
Data Carrier Detect	Asserted by DCE when a connection has been established with remote equipment	DCD		●
Data Set Ready	DCE is ready to receive commands or data.	DSR		●
Ring Indicator	DCE has detected an incoming ring signal on the telephone line.	RI		●
Request To Send	DTE requests the DCE prepare to receive data.	RTS	●	
Clear To Send	Indicates DCE is ready to accept data.	CTS		●
Transmitted Data	Carries data from DTE to DCE.	TxD	●	
Received Data	Carries data from DCE to DTE.	RxD		●
Common Ground		GND	common	
Protective Ground		PG	common	



UART - Errors

Overrun error

An "overrun error" occurs when the receiver cannot process the character that just came in before the next one arrives. Various devices have different amounts of buffer space to hold received characters. The CPU must service the UART in order to remove characters from the input buffer. If the CPU does not service the UART quickly enough and the buffer becomes full, an Overrun Error will occur, and incoming characters will be lost.

Underrun error

An "underrun error" occurs when the UART transmitter has completed sending a character and the transmit buffer is empty. In asynchronous modes this is treated as an indication that no data remains to be transmitted, rather than an error, since additional stop bits can be appended. This error indication is commonly found in USARTs, since an underrun is more serious in synchronous systems.

Framing error

A "framing error" occurs when the designated "start" and "stop" bits are not valid. As the "start" bit is used to identify the beginning of an incoming character, it acts as a reference for the remaining bits. If the data line is not in the expected idle state when the "stop" bit is expected, a *Framing Error* will occur.

Parity error

A "parity error" occurs when the number of "active" bits does not agree with the specified parity configuration of the UART, producing a *Parity Error*. Because the "parity" bit is optional, this error will not occur if parity has been disabled. Parity error is set when the parity of an incoming data character does not match the expected value.

ESP8266 UART

- ESP8266 has 2 UART, UART0 and UART1
- UART0 pins
  - U0TXD - GPIO01
  - U0RXD - GPIO03
  - U0CTS - GPIO13
  - U0RTS - GPIO15
- UART1 pins
  - U1TXD - GPIO02
- UART0 swap pins
  - U0TXD - GPIO15
  - U0RXD - GPIO13
- UART1 only Tx
- UART0 used code downloading and communication

UART Lib

`void begin(unsigned long baud, SerialConfig config)`

Description	This function to start serial module
Parameter	Baudrate, serial config
Return	Void
Example	Serial.begin(9600, SERIAL_8N1); // starts UART0 with baudrate 9600 parity none and stopbit 1

`void end();`

Description	This function to stops serial module
Parameter	void
Return	Void
Example	Serial.end(); // stops UART0

UART Lib continue..

`void updateBaudRate(unsigned long baud);`

Description	This function to update the baudrate value
Parameter	Baudrate
Return	Void
Example	Serial.updateBaudRate(19200); //updating baudrate of UART0 to 19200

`size_t setRxBufferSize(size_t size);`

Description	This function sets rx buffer size
Parameter	size
Return	size
Example	Serial. setRxBufferSize(32); //sets rx buffer size of UART0 to 32 bytes

UART Lib continue..

`size_t getRxBufferSize()`

Description	This function returns rx buffer size
Parameter	Void
Return	Size_t
Example	ui8Size = Serial.getRxBufferSize(); //returns rx buffer size of UART0

`void swap();`

Description	This function swaps rx and tx pins of UART0
Parameter	Void
Return	Void
Example	Serial. swap(); //swaps rx and tx pins of UART0

UART Lib continue..

`int available(void)`

Description	This function returns available bytes to read, -1 indicates no bytes to read
Parameter	Void
Return	Int
Example	iBytes = Serial. available(); //returns available byte size

`int read(void)`

Description	This function read character in rx buffer if no rx data then return -1
Parameter	Void
Return	int
Example	Byte = Serial. read(); // reads character sents to UART0

UART Lib continue..

`size_t read(char* buffer, size_t size)`

Description	This function reads mentioned byte size
Parameter	Void
Return	Size_t
Example	ui8Size = Serial. read(buffer, 10); //reads 10 characters from UART0

`size_t write(uint8_t c)`

Description	This function write bytes
Parameter	Uint8_t
Return	Size_t
Example	Serial. write(10); //writes value 10 to UART0

UART Lib continue..

`size_t write(const uint8_t *buffer, size_t size)`

Description	This function writes bytes array of mentioned size
Parameter	Uint8_t array, uint8_t size
Return	Size_t
Example	Serial.write(buffer, 10); //writes 10 bytes array

`void print();`

Description	This function prints mentioned data types, data types can int, float, char, string
Parameter	Data types
Return	Void
Example	Serial. print(“test”); Serial. print(‘a’); Serial. print(10); Serial. print(22.34);

UART Lib continue..

`void println();`

Description	This function prints mentioned data types with \r\n, data types can int, float, char, string
Parameter	Data types
Return	Void
Example	Serial. println(“test”); Serial. println(‘a’); Serial. println(10); Serial. println(22.34);

Note: Read serial library for other functions