lamaPLC Communication: UART / USART basic

USART

A **USART** (*universal synchronous/asynchronous receiver/transmitter*) is hardware that enables a device to communicate using serial protocols. It can function in a slower asynchronous mode, like a universal asynchronous receiver/transmitter (UART), or in a faster synchronous mode with a clock signal. USARTs are no longer common in consumer PCs but are still used in industrial equipment and embedded systems.

USART vs. UART

A UART device can use asynchronous communication protocols. A USART device can use both asynchronous and synchronous communication protocols. Therefore, a USART can do anything a UART can do and more. Because a USART requires more complex circuitry and more communication lines to fully implement, many devices may only implement a UART to save on cost, complexity or power usage.

Asynchronous and synchronous serial communication

In serial communication, each bit of data is sent one at a time on a transmit wire. This is a serial communications interface. If the sender and the receiver don't agree on how the data is sent, such as the order and length of time of each bit, then the data becomes garbled, and they won't understand each other. Asynchronous and synchronous are two different ways to standardize how serial data is sent.

Asynchronous serial data

In asynchronous mode, only one data line is used to send data from the transmitter to the receiver. There is no shared synchronization signal from the sender to the receiver. So, the receiver has no way to know how fast or slow the data is coming. To circumvent this, both the sender and receiver must be manually configured beforehand to use the same data rate. A common shared baud rate is 9,600 bits per second.

UART

A *universal asynchronous receiver-transmitter* (**UART**) is a peripheral device 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-422, RS-485, GPS-moduls and raw TTL for short debugging links. Early teletypewriters used current loops.

The universal asynchronous receiver-transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires.

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
- Flow Control

For the voltage level, 2 UART modules work well when they both have the same voltage level, e.g 3V-3V between the 2 UART modules. To use 2 UART modules at different voltage levels, a level switch circuit needs to be added externally

Data framing

A UART frame consists of 5 elements:

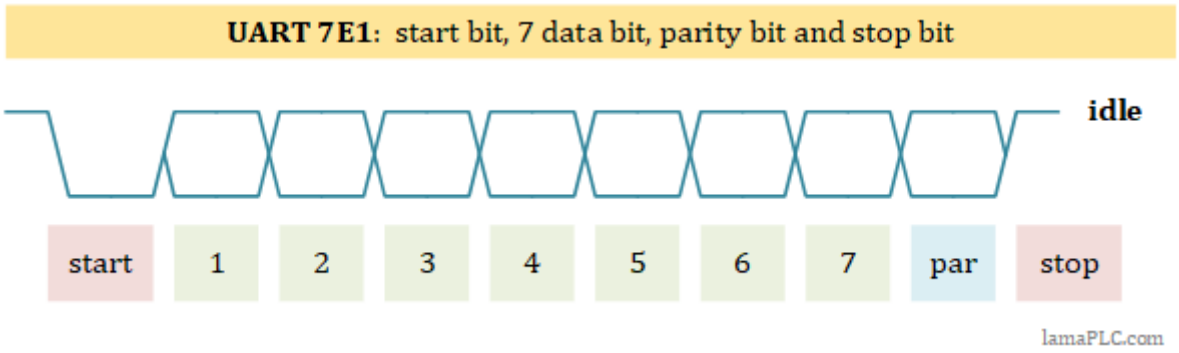
- Idle (logic high (1))
- Start bit (logic low (0))
- Data bits
- Parity bit
- Stop (logic high (1))

UART frame, field length in Bits			
1	**5-9**	**0-1**	**1-2**
Start Bit	Data Frame	Parity Bits	Stop Bits

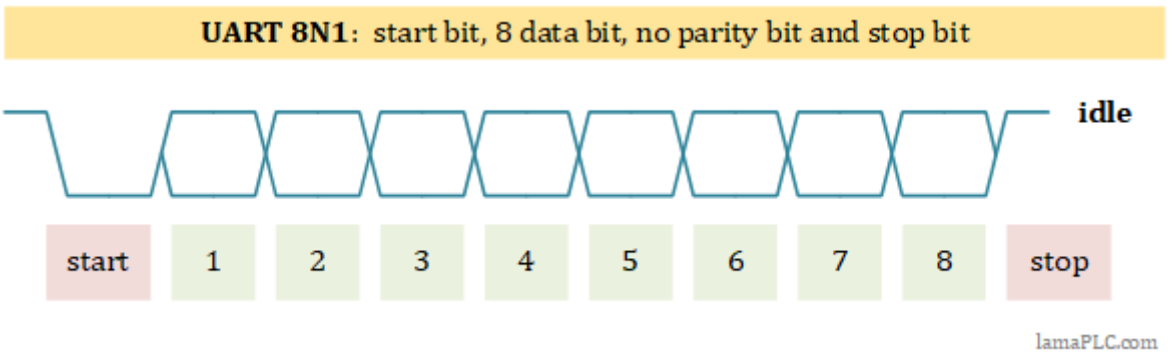
In the most common settings of 8 data bits, no parity and 1 stop bit (aka 8N1), the protocol efficiency is 80%. For comparison, Ethernet's protocol efficiency when using maximum throughput frames with payload of 1500 bytes is up to 95% and up to 99% with 9000 byte jumbo frames. However due to Ethernet's protocol overhead and minimum payload size of 42 bytes, if small messages of one or a few bytes are to be sent, Ethernet's protocol efficiency drops much lower than the UART's 8N1 constant efficiency of 80%.

The idle, no data state is high-voltage, or powered. This is a historic legacy from telegraphy, in which the line is held high to show that the line and transmitter are not damaged.

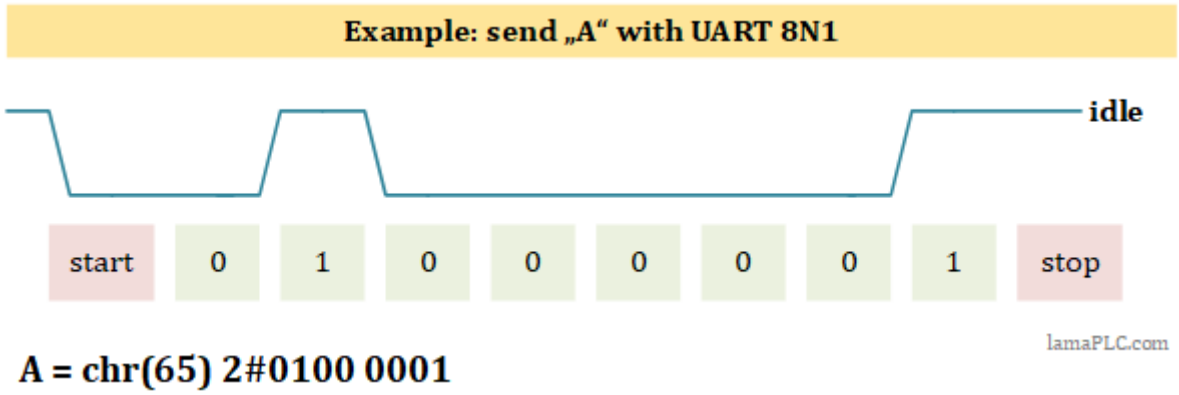
Each character is framed as a logic low start bit, data bits, possibly a parity bit and one or more stop bits. In most applications the least significant data bit (the one on the left in this diagram) is transmitted first, but there are exceptions (such as the IBM 2741 printing terminal).



The most typical UART, the 8N1, has the following structure:



Telegram of the letter “A” with the UART 8N1 structure:



Start bit

The start bit signals to the receiver that a new character is coming.

Data bit

The next five to nine bits, depending on the code set employed, represent the character.

Parity bit

If a parity bit is used, it would be placed after all of the data bits.
The parity bit is a way for the receiving UART to tell if any data has changed during transmission.

Stop bit

The next one or two bits are always in the mark (logic high, i.e., '1') condition and called the stop bit(s). They signal to the receiver that the character is complete. Since the start bit is logic low (0) and the stop bit is logic high (1) there are always at least two guaranteed signal changes between characters.

UART (serial) with Arduino

UART (serial) transmission is perhaps the most commonly used communication method by Arduino, along with I²C and SPI. Communication via UART is enabled by the Serial class, which has a number of methods available, including reading & writing data.

Serial Class

With the Serial class, you can send / receive data to and from your computer over USB, or to a device connected via the Arduino's RX/TX pins.

When sending data over USB, can use Serial. This data can be viewed in the Serial Monitor in the Arduino IDE.
When sending data over RX/TX pins, we use Serial1.
The GIGA R1 WiFi, Mega 2560 and Due boards also have Serial2 and Serial3
The Serial class have several methods with some of the essentials being:

- begin() - begins serial communication, with a specified baud rate (many examples use either 9600 or 115200).
- print() - prints the content to the Serial Monitor.
- println() - prints the content to the Serial Monitor, and adds a new line.
- available() - checks if serial data is available (if you send a command from the Serial Monitor).
- read() - reads data from the serial port.

- write() - writes data to the serial port.

For example, to initialize serial communication on both serial ports, we would write it as:

```
Serial.begin(9600);      // init communication over USB
Serial1.begin(9600);     // communication over RX/TX pins
```

Sources

Wikipedia (here)

UART topics on lamaPLC

Page	Date	Tags
• lamaPLC Communication: 1-Wire	2025/05/31 21:56	1-wire, communication, bus, microlan, i2c, uart, usart, ds18b20
• lamaPLC Communication: GPS	2024/11/15 20:15	communication, satellite, navigation, gps, ocx, cdma, glonass, beidou, galileo, qzss, uart, arduino
• lamaPLC Communication: UART / USART basic	2025/02/11 20:21	bus, communication, uart, rs-232, rs-422, rs-485
• lamaPLC: A0221AU / A02YYUW Waterproof Ultrasonic Distance Sensor	2025/11/13 21:51	a0221au, a02yyuw, waterproof, ultrasonic, distance, sensor, uart, ip67, serial
• lamaPLC: ESP32 / ESP8266	2025/11/21 23:07	esp8266, esp32, esp32-c2, esp32-c3, esp32-c5, esp32-c6, esp32-c61, esp32-h2, esp32-s2, esp32-s3, esp32-p4, espressif systems, communication, ethernet, ip, wi-fi, thread, zigbee, matter, homekit, bluetooth, mqtt, adc, spi, uart, i2c, i2s, rmt, pwm, usb, usb otg, twai

bus, communication, UART, RS-232, RS-422, RS-485