UART Protocol

**1. What is UART?**

* **UART = Universal Asynchronous Receiver/Transmitter**
* A hardware protocol for **serial communication** (used by Arduino’s Serial).
* **Asynchronous** → no shared clock, both devices must agree on a speed (baud rate).
* Data is sent **bit by bit** over two wires (TX ↔ RX).
* Requires a **common ground**.
* Modes:
  + **Simplex** → one-way only
  + **Half-duplex** → both sides, but one at a time
  + **Full-duplex** → both sides simultaneously

**2. Baud Rate**

* **Baud rate = symbols per second**.
* In UART, **1 symbol = 1 bit** → baud ≈ bits/sec.
* Example: 9600 baud = 9600 bits per second.
* But each frame has **extra bits** (start/stop/parity).
* 1 start + 8 data + 1 stop = 10 bits per byte
* **Effective throughput = baud ÷ 10 characters per second.**

**Example**

* **9600 baud:**
  + 9600 ÷ 10 = **960 bytes/sec**
  + 1 byte ≈ **1.04 ms**
* **115200 baud:**
  + 115200 ÷ 10 = **11,520 bytes/sec**
  + 1 byte ≈ **86 µs**

**3. UART Frame (8N1 format – Arduino default)**

Idle (HIGH) → Start bit (LOW) → 8 Data bits (LSB first) → Stop bit (HIGH)

* **Idle line:** HIGH (logic 1, also called *mark*).
* **Start bit:** LOW (*space*) → signals new frame.
* **Data bits:** Sent LSB → MSB (5–9 possible, usually 7 or 8).
* **Optional parity bit:** For error detection.
* **Stop bit(s):** HIGH (end of frame, 1 or 2).

**4. How does the receiver know?**

* Waits for **idle HIGH**.
* Detects **falling edge** → start bit.
* Uses **baud rate** to schedule sampling points (middle of each bit).
* Collects configured number of data bits.
* Expects stop bit = HIGH → otherwise **framing error**.

**5. Common Configurations**

* **8N1** → 8 data bits, No parity, 1 stop bit (Arduino default).
* **7E1** → 7 data bits, Even parity, 1 stop bit.
* **8O2** → 8 data bits, Odd parity, 2 stop bits.  
  👉 Both ends **must match exactly**.

**6. Parity Bit**

* Optional, for **single-bit error detection**.
* **Even parity:** total 1s in frame must be even.
* **Odd parity:** total 1s must be odd.
* Limitation → cannot detect multiple flipped bits reliably.

**7. Errors**

* **Framing error:** stop bit not seen correctly (baud mismatch, noise).
* **Parity error:** wrong parity bit (if enabled).
* **Overrun error:** buffer fills up before being read.

**8. Voltage Levels**

* **Arduino Uno (ATmega328P):** 5 V logic.
* **ESP32 / STM32 / many MCUs:** 3.3 V logic.
* Mismatched levels require a **level shifter**.
* Idle state HIGH makes it easy to detect broken lines.

**9. Practical Arduino Example**

Serial.begin(9600); // set baud rate

if (Serial.available() > 0) {

char c = Serial.read(); // read one character

}

* At **9600 baud**, "Hello\n" (6 chars = 60 bits) takes ~6.25 ms.
* Arduino (16 MHz) executes ~16,000 instructions per ms → **serial is slow** relative to CPU speed.

**10. Why buffer handling matters**

* Arduino’s serial buffer = **64 bytes**.
* Data may still be arriving while you’re reading.
* Use **end markers** (\n, \r) or delimiters to detect complete messages.
* Don’t assume while (Serial.available()) = full message.

**✅ Key Takeaways**

* UART = simple, asynchronous protocol, **1 wire per direction + ground**.
* **Baud rate** sets speed, but overhead reduces actual data rate.
* Must agree on **baud rate + frame format** (e.g., 8N1).
* Weak error detection (only parity).
* **Buffers can overflow** if data not read quickly enough.
* Arduino is much faster than UART → design code to handle partial data.
* Still widely used in **embedded systems**, though replaced in many areas by SPI, I²C, USB, Ethernet.