UART Basics

UART = Universal Asynchronous Receiver Transmitter

- TX → Transmit pin
- RX → Receive pin
- No clock line (asynchronous)
- Both sides must agree on **baud rate**

On Arduino UNO:

- TX → Pin **1**
- RX → Pin **0**

Stage 1: Serial Print (Arduino → PC)

Goal:-

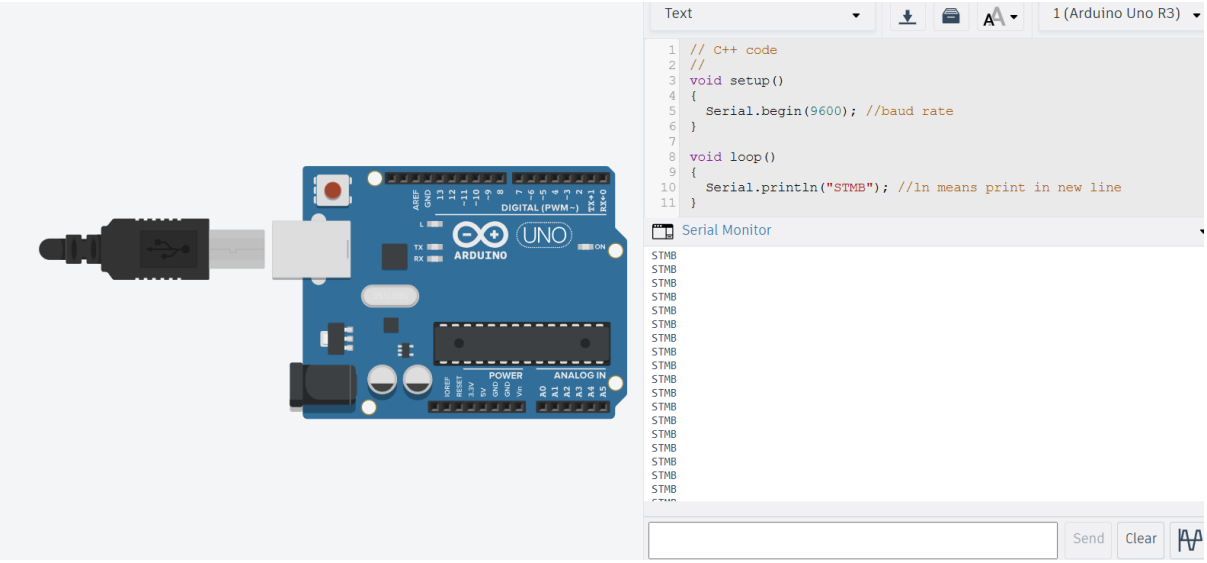
Send text from Arduino to Serial Monitor.

Code:-

```
void setup() {
  Serial.begin(9600);   // Start UART at 9600 baud rate
}


void loop() {
  Serial.println("STMB"); //ln makes to print in new line
  delay(1000);
}
```

Diagram:-



Stage 2: LED Control Using UART (PC → Arduino)

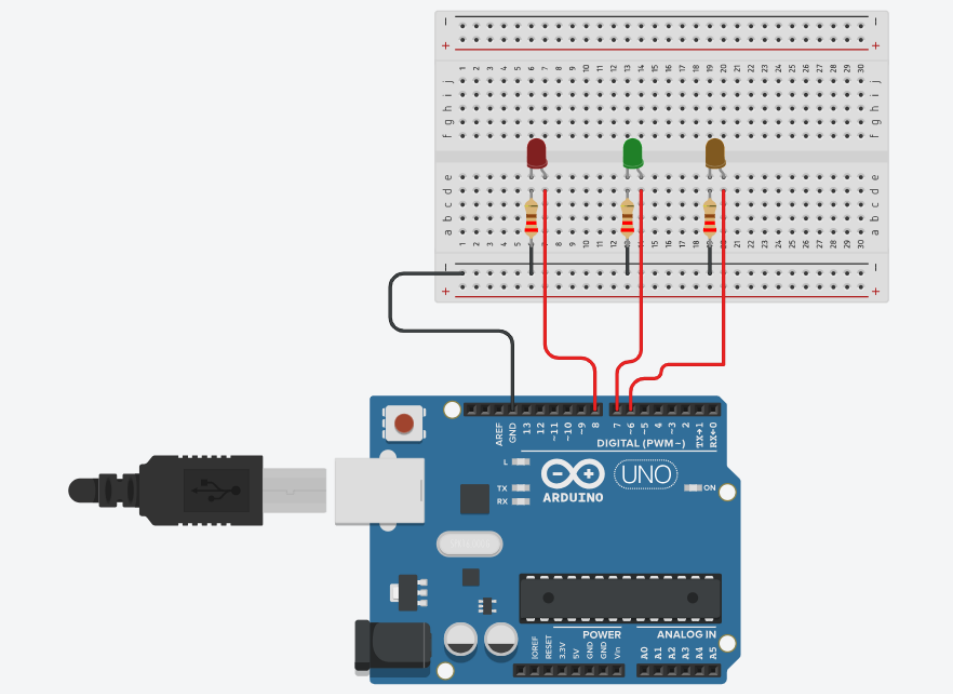
Components

- Arduino Uno - 1
- LED – 3 (Different Colours)
- 220Ω resistor

Connections

- LED + → **Ardiuno Digital Pins**
- LED − → **220Ω → GND**

Circuit Diagram :-



Code:-

```
char data;

void setup() {
  pinMode(8, OUTPUT);   // Red LED
  pinMode(7, OUTPUT);   // Green LED
  pinMode(6, OUTPUT);   // Yellow LED

  Serial.begin(9600);   // UART start
  Serial.println("UART LED Control Ready");
}

void loop() {
  if (Serial.available() > 0) {
    data = Serial.read();

    if (data == 'R') {
      digitalWrite(8, HIGH);
      Serial.println("Red LED ON");
    }
    else if (data == 'r') {
      digitalWrite(8, LOW);
      Serial.println("Red LED OFF");
    }

    else if (data == 'G') {
      digitalWrite(7, HIGH);
      Serial.println("Green LED ON");
```

```
  }
  else if (data == 'g') {
    digitalWrite(7, LOW);
    Serial.println("Green LED OFF");
  }

  else if (data == 'Y') {
    digitalWrite(6, HIGH);
    Serial.println("Yellow LED ON");
  }
  else if (data == 'y') {
    digitalWrite(6, LOW);
    Serial.println("Yellow LED OFF");
  }
 }
}
```

Concept:-

Arduino receives **UART data from PC (Serial Monitor)** and:

- Decodes the command
- Turns ON / OFF the corresponding LED

Important:

- UART here is **PC ↔ Arduino** via USB
- Internally Arduino uses **TX (pin 1) & RX (pin 0)**

UART Concept (Very Important)

1. Serial.begin(9600)

- Initializes UART
- Both sides must match baud rate

2. Serial.available()

- Checks if data has arrived
- Prevents reading garbage

3. Serial.read()

- Reads **1 byte (8 bits)**
- UART is **byte-oriented**

4. Why characters ('R', 'G')?

- Easier than numbers
- Common in **debug commands, ECUs, BMS tools**

UNDERSTANDING THE UART FLOW (VERY IMPORTANT)

What Actually Happens Internally

1. If we type R in Serial Monitor
2. PC sends **ASCII 'R' (0x52)** via UART
3. Arduino receives it on **RX (pin 0 internally)**
4. Serial.read() reads **1 byte**
5. Program compares and switches LED

Stage 3 : REAL UART → Arduino ↔ Arduino communication

Goal of This Experiment

- **Arduino 1 (Master)** sends data using UART
- **Arduino 2 (Slave)** receives data and controls LEDs
- Understand **TX–RX cross connection**

Components:-

- 2 × Arduino UNO
- Breadboard
- 3 LEDs + resistors
- **UART WIRING (MOST IMPORTANT PART)**
- **UART Cross Connection Rule**

Arduino Master		**Arduino Slave**
TX (Pin 1)	→	RX (Pin 0)
RX (Pin 0)	→	TX (Pin 1)
GND	→	GND

- **TX → RX, RX → TX (always cross)**
  Common GND is mandatory
- **LED Connections (Slave Arduino Only)**

LED Color	**Pin**
Red	8
Green	7
Yellow	6

Master → sends commands
Slave → receives & acts

SLAVE ARDUINO CODE (Receiver + LED Control) :-

```
char data;


void setup() {
  pinMode(8, OUTPUT);   // Red
  pinMode(7, OUTPUT);   // Green
  pinMode(6, OUTPUT);   // Yellow


  Serial.begin(9600);   // UART start
}


void loop() {
  if (Serial.available() > 0) {
    data = Serial.read();


    if (data == 'R') digitalWrite(8, HIGH);
    else if (data == 'r') digitalWrite(8, LOW);


    else if (data == 'G') digitalWrite(7, HIGH);
    else if (data == 'g') digitalWrite(7, LOW);
```

```
    else if (data == 'Y') digitalWrite(6, HIGH);
    else if (data == 'y') digitalWrite(6, LOW);
  }
}
```

MASTER ARDUINO CODE (Transmitter) :-

```
void setup() {
  Serial.begin(9600);   // UART start
}

void loop() {
  Serial.write('R');   // Red ON
  delay(1000);

  Serial.write('r');   // Red OFF
  delay(1000);

  Serial.write('G');   // Green ON
  delay(1000);

  Serial.write('g');   // Green OFF
  delay(1000);

  Serial.write('Y');   // Yellow ON
  delay(1000);

  Serial.write('y');   // Yellow OFF
  delay(1000);
}
```

KEY UART CONCEPTS

Why Serial.write() here?

- Sends **raw byte**
- Faster & cleaner than Serial.print()

Why same baud rate?

- UART is **asynchronous**
- Baud mismatch = garbage data

Why common ground?

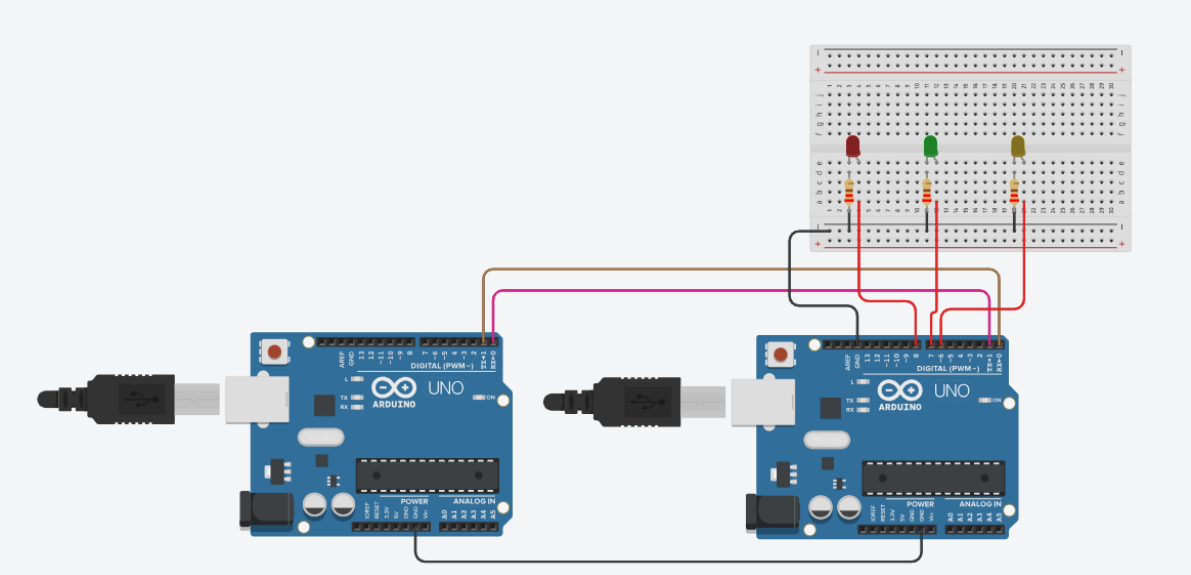
- Voltage reference must be same

WHAT HAPPENS ELECTRICALLY (IMPORTANT)

Let's say Master sends 'R':

1. Master TX toggles **HIGH/LOW** bits
2. Slave RX sees:
    - Start bit
    - 8 data bits (0x52)
    - Stop bit
3. UART hardware converts bits → byte
4. Serial.read() gets the byte
5. Code turns **Red LED ON**

Circuit Diagram:-



ADD-ON 1 :TWO-WAY UART between two Ardiuno's (Full-Duplex Communication) :-

GOAL

- **Master Arduino** sends a command
- **Slave Arduino** executes it
- **Slave sends ACK (response)** back
- **Master receives & prints it**

CONNECTIONS (NO CHANGE)

Keep **everything exactly the same**:

- TX ↔ RX (crossed)
- RX ↔ TX
- Common GND
- LEDs on Slave (pins 8, 7, 6)
- Led on master ardiuno (pin 5) to check the ack sends by slave ardiuno after performing each operation (LED Blinking).

FINAL SLAVE CODE (OPERATION + ACK) :-

```
char data;


void setup() {
  pinMode(8, OUTPUT);   // Red
  pinMode(7, OUTPUT);   // Green
  pinMode(6, OUTPUT);   // Yellow


  Serial.begin(9600);
}


void loop() {
  if (Serial.available() > 0) {
    data = Serial.read();
```

```
switch (data) {
  case 'R':
    digitalWrite(8, HIGH);
    Serial.write('A');   // ACK
    break;

  case 'r':
    digitalWrite(8, LOW);
    Serial.write('A');
    break;

  case 'G':
    digitalWrite(7, HIGH);
    Serial.write('A');
    break;

  case 'g':
    digitalWrite(7, LOW);
    Serial.write('A');
    break;

  case 'Y':
    digitalWrite(6, HIGH);
    Serial.write('A');
    break;

  case 'y':
    digitalWrite(6, LOW);
```

```
      Serial.write('A');
      break;

    default:
      Serial.write('E');   // Error
      break;
   }
 }
}
```

FINAL MASTER CODE (COMMAND + ACK VERIFY) :-

```
char commandList[] = {'R','r','G','g','Y','y'};
int index = 0;

void setup() {
 pinMode(5, OUTPUT);   // ACK LED
 Serial.begin(9600);
}

void loop() {
 Serial.write(commandList[index]);   // Send command
 delay(300);                         // Allow slave to act

 if (Serial.available() > 0) {
  char response = Serial.read();

  if (response == 'A') {
    digitalWrite(5, HIGH);  // ACK received
    delay(600);             // Visible ACK
```

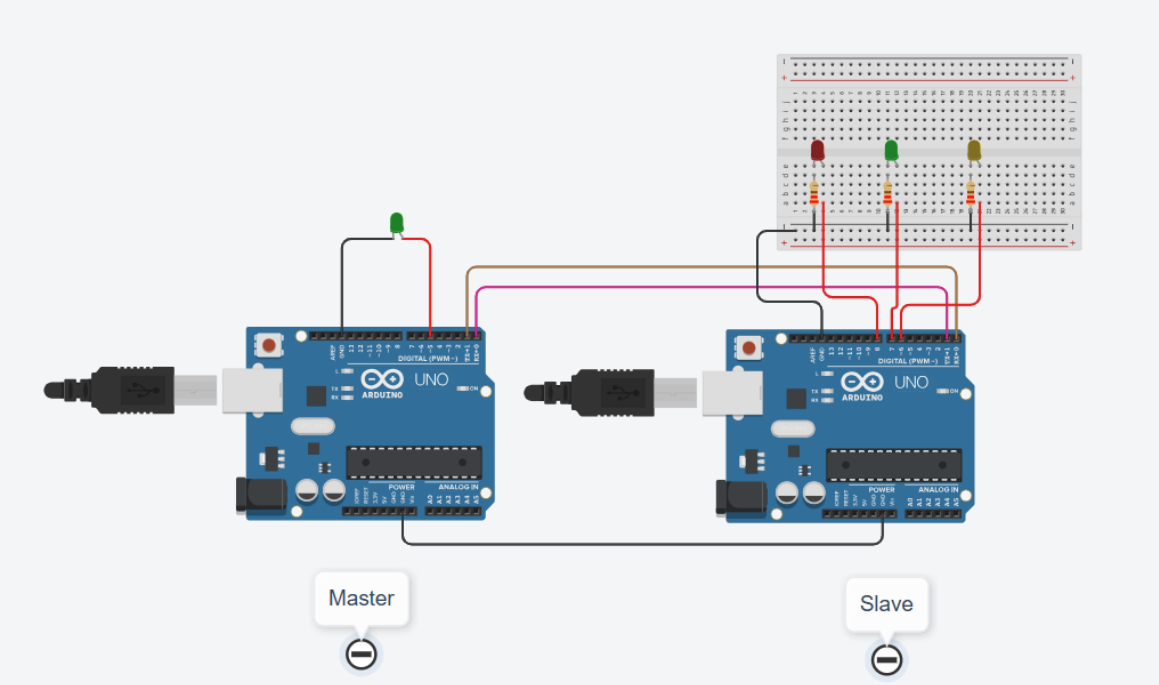
```
    digitalWrite(5, LOW);
  }
 }

 index++;
 if (index >= 6) index = 0;

 delay(800);   //  Gap between commands
}
```

Concept	**Meaning**
Full-duplex	Send & receive at same time
ACK	Confirms command execution
Serial.println()	Sends ASCII response
UART reliability	Verified via response

Circuit Diagram:-



ADD-ON 2 :- BAUD RATE MISMATCH DEMO

What to do

- Master: 9600
- Slave: 4800

Observe

- Garbage data / no LED action

WHAT IS ACTUALLY GOING WRONG (IMPORTANT)

Let's say:

- Master sends at 9600 bps
- Slave expects 4800 bps (Assume for example)

Slave samples bits too slowly:

Expected bits: **1 0 1 0 0 1 0 1**

Read bits: **1 1 0 1 ?**

-> Start bit detected incorrectly
-> Data bits shift
-> Stop bit invalid
-> Byte rejected or misread

UART frame:

| Start | D0 D1 D2 D3 D4 D5 D6 D7 | Stop |

- Receiver uses baud rate to decide when to sample
- Wrong baud = wrong sampling instant

HOW THESE UART EXPERIMENTS MAP TO HARDWARE

We did:
1. Serial Print
2. LED control via UART
3. Arduino ↔ Arduino UART (with ACK)

Now we'll decode what the UART hardware is actually doing.

1. UART IS A HARDWARE BLOCK (INSIDE MCU)

Inside every microcontroller (including Arduino / ATmega328P) there is a UART peripheral.

It contains:

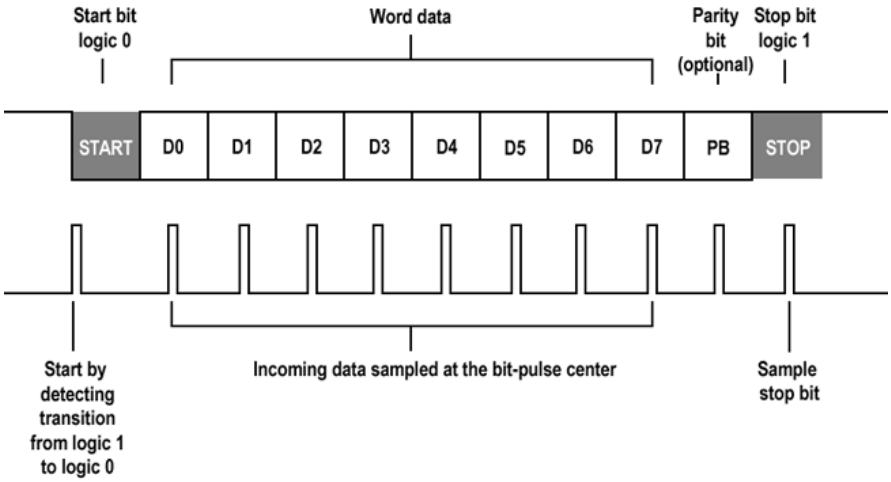
- Shift registers
- Baud rate generator
- Control & status registers
- TX and RX logic

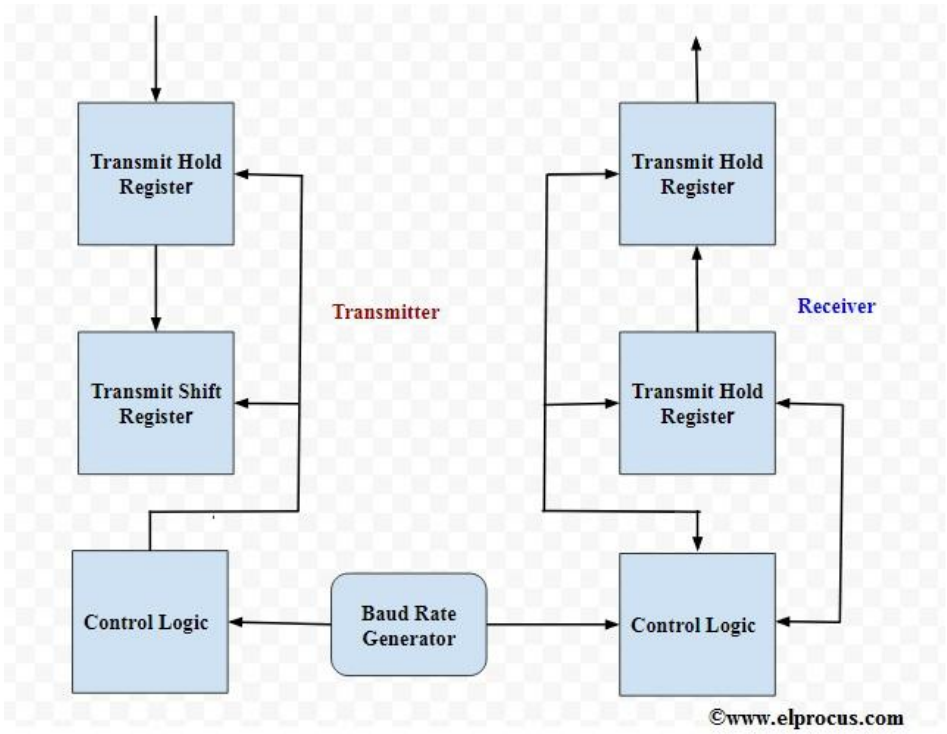
In our code (Serial.begin()) just configures this hardware.

2.WHAT HAPPENS WHEN YOU SEND A BYTE

Let’s say Master sends 'R'.

Inside the hardware:





1. UART loads 0x52 ('R') into **TX shift register**
2. Adds:
    - **Start bit** (0)
    - **8 data bits** (LSB first)
    - **Stop bit** (1)
3. Outputs bits on **TX pin (Pin 1)** at **9600 bps**

Electrical signal on wire:

Idle HIGH → Start LOW → 01010010 → Stop HIGH

3.WHAT THE RECEIVER HARDWARE DOES

On Slave side:

1. RX pin sees **falling edge** (start bit)
2. Baud-rate generator starts timing
3. Samples each bit at the **center**
4. Reconstructs byte
5. Stores it in **RX buffer**
6. Sets **RX flag**

Then:

`Serial.available() > 0` becomes TRUE

4. WHY TX ↔ RX CROSSING IS REQUIRED

- TX hardware **drives** the line
- RX hardware **samples** the line

If:

- TX → TX → conflict
- RX → RX → nothing driven

That's why:

TX → RX

RX → TX

5. WHY COMMON GROUND IS HARDWARE-CRITICAL

UART logic levels are:

- HIGH ≈ +5V
- LOW ≈ 0V

RX compares voltage **with respect to GND**.

No common GND = wrong reference = bit errors.

6. WHAT BAUD RATE REALLY MEANS IN HARDWARE

Baud rate generator:

System Clock / Divider = Bit Time

If divider mismatch:

- RX samples too early / too late
- Start bit misdetected
- Frame error occurs

This is why your **baud mismatch test failed**.

7. WHAT ACK PROVES AT HARDWARE LEVEL

Your ACK experiment proves:

- RX buffer works
- TX shift register works
- Full-duplex UART hardware works
- Interrupt/flag mechanism works