Arduino’s Serial Port and Serial Communication

Introduction to Serial Communication

Serial communication is a method of transmitting data between a computer and Arduino or between two devices using a single data line. Arduino has a built-in **UART (Universal Asynchronous Receiver-Transmitter)** that allows serial communication through its **TX (Transmit) and RX (Receive) pins**.

Types of Serial Communication in Arduino

1. **UART (Universal Asynchronous Receiver-Transmitter)** – Uses TX and RX pins for data transmission.
2. **I2C (Inter-Integrated Circuit)** – Uses two pins, **SDA (A4)** and **SCL (A5)**, for communication with multiple devices.
3. **SPI (Serial Peripheral Interface)** – Uses MOSI, MISO, and SCK for high-speed data exchange.

Working with Serial Communication in Arduino

1. Initializing Serial Communication

To enable serial communication in Arduino, use:

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

Here, **9600** is the **baud rate** (bits per second). Other common values are **115200, 57600, 38400**.

2. Sending Data via Serial Monitor

You can send data from Arduino to the Serial Monitor using:

```
void loop() {
   Serial.println("Hello, Arduino!");  // Print message to Serial Monitor
   delay(1000);
}
```

- Serial.print() → Prints text without a newline.
- Serial.println() → Prints text with a newline.

3. Receiving Data from Serial Monitor

To read input from the Serial Monitor:

cpp
CopyEdit

```
void loop() {
    if (Serial.available() > 0) {  // Check if data is received
        String data = Serial.readString();  // Read input as a string
        Serial.print("You entered: ");
        Serial.println(data);
    }
}
```

- Serial.available() → Checks if data is available.
- Serial.read() → Reads a single byte.
- Serial.readString() → Reads the entire input as a string.

Example: Controlling an LED using Serial Communication

This program turns an LED ON or OFF based on user input.

```
int ledPin = 13;

void setup() {
    Serial.begin(9600);  // Start Serial communication
    pinMode(ledPin, OUTPUT);
}

void loop() {
    if (Serial.available() > 0) {
        char command = Serial.read();  // Read single character input
        if (command == '1') {
            digitalWrite(ledPin, HIGH);  // Turn LED ON
            Serial.println("LED ON");
        } else if (command == '0') {
            digitalWrite(ledPin, LOW);  // Turn LED OFF
            Serial.println("LED OFF");
        }
    }
}
```

Output of the Given Program

Scenario 1: User Inputs '1' in Serial Monitor

User Input: `1`
Output in Serial Monitor:

```
LED ON
```

LED Status: ✅ **Turns ON** (Pin 13 is set to HIGH)

Scenario 2: User Inputs '0' in Serial Monitor

User Input: `0`
Output in Serial Monitor:

```
LED OFF
```

LED Status: ❌ **Turns OFF** (Pin 13 is set to LOW)

Explanation of Output Behavior

1. **When the user enters '1'** in the Serial Monitor and presses **Enter**,
   - The program reads the input using `Serial.read()`.
   - Since `command == '1'`, Arduino **turns ON the LED (pin 13)**.
   - `"LED ON"` is printed on the **Serial Monitor**.
2. **When the user enters '0'**,
   - The program reads the input.
   - Since `command == '0'`, Arduino **turns OFF the LED (pin 13)**.
   - `"LED OFF"` is printed on the **Serial Monitor**.

Example Serial Monitor Output

```
1   [ENTER]
LED ON

0   [ENTER]
LED OFF

1   [ENTER]
LED ON

0   [ENTER]
LED OFF
```

Applications of Serial Communication

- Debugging Arduino programs using the Serial Monitor.
- Sending sensor data to a computer for logging.
- Controlling Arduino remotely using a computer or Bluetooth module.
- Communicating between multiple Arduino boards.