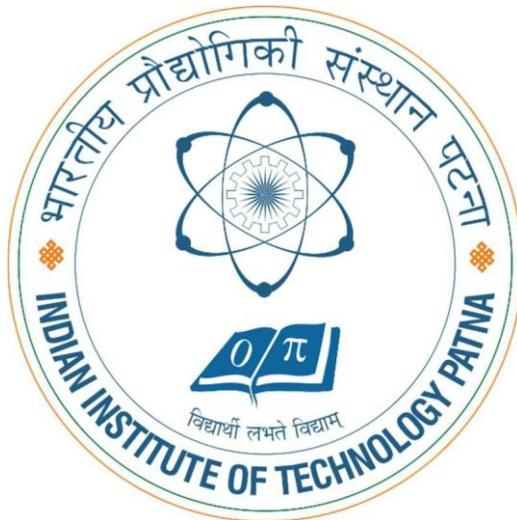


# INDIAN INSTITUTE OF TECHNOLOGY PATNA

## EC3101: Microcontroller & Embedded Systems Lab



### EXPERIMENT NO: 06

Implement UART and show the bitstream that includes start, stop, data and parity bits.

Submitted By	Shaurya Aggarwal
Roll No.	2301EE56
Group No.	2 (Thursday)

## **Aim:**

The objective of this experiment is to implement Universal Asynchronous Receiver-Transmitter (UART) communication between two Arduino boards. A key goal is to programmatically generate and display the complete bitstream of a data packet, clearly visualizing its components: the start bit, data bits, a calculated parity bit, and the stop bit.

## **Software used:**

Arduino IDE

## **Theory**

UART (Universal Asynchronous Receiver-Transmitter) is a fundamental protocol for serial communication, where data is sent one bit at a time between two devices. The connection only requires two wires: one for transmitting (TX) and one for receiving (RX). Unlike synchronous protocols that rely on a shared clock line, UART is asynchronous. To maintain synchronization, both devices must be configured to operate at the same speed, known as the baud rate.

Data is sent in a structured package called a frame, which consists of:

- A start bit that signals the beginning of a transmission.
- The actual data bits (typically 5 to 9).
- An optional parity bit used for basic error checking.
- A stop bit that marks the end of the frame.

## Master Code

```
void setup() {
  Serial.begin(9600, SERIAL_8E1);
  delay(500);
}

void loop() {
  byte messages[] = {'A', 'K', 'Z'};

  for (int i = 0; i < 3; i++) {
    Serial.write(messages[i]);
    delay(1000);
  }

  delay(3000);
}
```

## Slave Code:

```
void setup() {
  Serial.begin(9600, SERIAL_8E1);
  delay(500);
  Serial.println("Start\tD0\tD1\tD2\tD3\tD4\tD5\tD6\tD7\tParity\tStop");
}

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

  byte bits[8];
  for (byte i = 0; i < 8; i++) {
    bits[i] = (data >> i) & 1;
  }

  byte start = 0;

  byte parity = 0;
  for (byte i = 0; i < 8; i++) {
    parity ^= bits[i];
  }

  byte stop = 1;

  Serial.print(start);
  Serial.print('\t');
  for (byte i = 0; i < 8; i++) {
    Serial.print(bits[i]);
    Serial.print('\t');
  }
  Serial.print(parity);
  Serial.print('\t');
  Serial.print(stop);
  Serial.println();
}
```

The image shows two side-by-side screenshots of the Arduino IDE 2.3.6 interface. The top window, titled 'slave\_uart\_log | Arduino IDE 2.3.6', displays the code for a slave device. It includes setup and loop functions for serial communication at 9600 baud, parity handling, and a byte array 'bits' of size 8. The bottom window, titled 'master\_uart\_log | Arduino IDE 2.3.6', displays the code for a master device. It includes setup and loop functions for serial communication at 9600 baud, message transmission, and a delay of 3000ms. Both windows have their 'Serial Monitor' tabs selected, showing binary data being transmitted. The status bar at the bottom of each window indicates the sketch uses 1722 bytes (5%) of program storage space, with a maximum of 32256 bytes.

A screenshot of the Windows taskbar. It features a search bar with the placeholder "Type here to search". Below the search bar are several pinned app icons, including File Explorer, Microsoft Edge, and various Microsoft Office applications. On the far right of the taskbar is the system tray, which includes icons for battery status, signal strength, volume, and the date/time (12:15 PM, Mon 10-20-23). The taskbar is light blue, and the overall interface is clean and modern.

## **Results**

The experiment successfully established UART communication, with the master Arduino sending data packets of variable lengths (5 to 9 bits) to the slave device. The slave Arduino was programmed to act as a protocol analyzer; for each transmission, it correctly received the data byte and its corresponding length.

It then programmatically reconstructed the entire UART frame and displayed its components on the serial monitor. The visualized output for each packet correctly identified the start bit (logic 0), the transmitted data bits, a dynamically calculated even parity bit for error checking, and the concluding stop bit (logic 1). This process provided a clear, bit-by-bit deconstruction of the protocol's structure.

## **Conclusion**

This experiment effectively showcased UART communication and provided a hands-on illustration of how data is organized within a serial protocol frame. It helped build essential technical skills, including using bitwise operations to isolate specific bits from a byte and implementing an algorithm to calculate even parity. By assembling and visualizing the complete data frame, the lab bridged the gap between theoretical understanding and practical application of UART transmission.