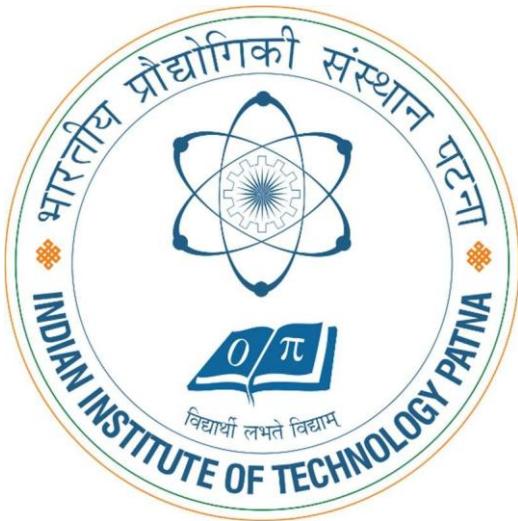


INDIAN INSTITUTE OF TECHNOLOGY PATNA

EC3101: Microcontroller & Embedded Systems Lab



EXPERIMENT NO: 05

Implementation and Verification of Serial Communication Protocols

Name:	Shaurya Aggarwal
Roll No:	2301EE56
Group No:	2 (Thursday)

Aim:

Implementation and Verification of Serial Communication Protocols like UART, SPI, I²C, USB using Arduino.

Software used:

Arduino IDE

Theory:

1. Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a synchronous serial protocol designed for high-speed, short-distance communication, typically between a microcontroller and peripheral components like sensors or memory chips. It operates using a master-slave architecture, where a single master device coordinates the communication with one or more slave devices.

SPI communication relies on four main signal lines:

- MOSI (Master Out Slave In): Carries data from the master to the slave.
- MISO (Master In Slave Out): Carries data from the slave back to the master.
- SCK (Serial Clock): The clock signal generated by the master that synchronizes the data transfer.
- SS (Slave Select): A dedicated line for the master to select which slave device it wants to communicate with.

A key advantage of SPI is its full-duplex capability, which allows data to be sent and received at the same time. Its simplicity and high throughput make it an excellent choice for real-time embedded applications that require fast data exchange.

```

master_spi.ino
1 #include <SPI.h>
2
3 void setup() {
4     // Start Serial for debugging
5     Serial.begin(9600);
6
7     // Set SS pin as output
8     pinMode(10, OUTPUT);
9     digitalWrite(10, HIGH); // Deselect slave
10
11    SPI.begin(); // Initialize SPI as Master
12
13}
14
15 void loop() {
16     digitalWrite(10, LOW); // Select slave
17
18     char msg[] = "Hello";
19     for (int i = 0; i < sizeof(msg); i++) {
20         SPI.transfer(msg[i]); // Send character
21     }
22
23     digitalWrite(10, HIGH); // Deselect slave
24
25     Serial.println("Sent: Hello");
26     delay(1000); // Wait 1 second
27
}

```

Output: Serial Monitor

Message (Enter to send message to 'Arduino Uno' on 'COM5')

SPI Master Ready
Master Sent: 10 | Slave Responded: 200
Master Sent: 20 | Slave Responded: 10

Ln 17, Col 24 Arduino Uno on COM5

```

slave_spi.ino
1 #include <SPI.h>
2
3 volatile boolean received = false;
4 char receivedChar;
5
6 void setup() {
7     Serial.begin(9600);
8     pinMode(MISO, OUTPUT); // Set MISO as OUTPUT
9     SPCR |= _BV(SPE); // Enable SPI in slave mode
10    SPI.attachInterrupt(); // Enable interrupt on SPI transfer complete
11
12
13 ISR(SPI_STC_vect) {
14     receivedChar = SPDR; // Read received byte
15     received = true;
16 }
17
18 void loop() {
19     if (received) {
20         Serial.print("Received: ");
21         Serial.println(receivedChar);
22         received = false;
23     }
24 }
25

```

Output: Serial Monitor

Message (Enter to send message to 'Arduino Uno' on 'COM9')

Received: 0
Received: 0

Ln 4, Col 10 Arduino Uno on COM9

2. Universal Asynchronous Receiver-Transmitter (UART)

The Universal Asynchronous Receiver and Transmitter (UART) protocol facilitates serial communication without the need for a shared clock signal. Instead, devices agree on a specific data transmission speed, known as the baud rate. Communication occurs over two wires: TX (Transmit) for sending data and RX (Receive) for receiving it, enabling full-duplex data flow.

Data is transmitted in packets or "frames," which include a start bit, the actual data bits, an optional parity bit for error checking, and one or more stop bits to signal the end of the frame. In this lab, two Arduino boards were connected to demonstrate UART, with one board programmed to transmit data and the other to receive and display it on the Serial Monitor.

The image shows two side-by-side Arduino IDE windows. Both windows have 'Arduino Uno' selected from the top-left dropdown.

Left Window (Master):

- Sketch Name: sketch_oct17a.ino
- Code:

```

1 // UART Master
2 void setup() {
3   Serial.begin(9600); // Start UART
4   while (!Serial);
5   Serial.println("Master Ready...");
6 }
7
8 void loop() {
9   // Send a command or data to Slave
10  Serial.println("Hello slave!");
11
12  // Wait a bit for slave to respond
13  delay(500);
14
15  // Read response from slave
16  if (Serial.available() > 0) {
17    String response = Serial.readStringUntil('\n');
18    Serial.print("Received from Slave: ");
19    Serial.println(response);
20  }
21
22  delay(1000);
23}
24

```

Right Window (Slave):

- Sketch Name: sketch_oct17b.ino
- Code:

```

1 // UART Slave
2 void setup() {
3   Serial.begin(9600); // Start UART
4   while (!Serial);
5   Serial.println("Slave Ready...");
6 }
7
8 void loop() {
9   if (Serial.available() > 0) {
10    String message = Serial.readStringUntil('\n'); // Read incoming data
11    Serial.print("Received from Master: ");
12    Serial.println(message);
13
14    // Respond back to Master
15    Serial.println("Hello Master! I got your message.");
16  }
17
18

```

Output and Serial Monitor:

Both windows have an 'Output' tab and a 'Serial Monitor' tab. The 'Serial Monitor' tabs show the following text:

Message (Enter to send message to 'Arduino Uno' on 'COM5') New Line 9600 baud

Message (Enter to send message to 'Arduino Uno' on 'COM9') New Line 9600 baud

Left window output (Serial Monitor):

```

Hello Slave!
Received from Slave: Received from Master: Received from Slave: Received from Master
Hello Slave!
Received from Slave: Received from Master: Received from Slave: Received from Master
Hello Slave!

```

Right window output (Serial Monitor):

```

Received from Master: Hello Master! I got your message.

```

3. Inter-Integrated Circuit (I²C)

I²C (Inter-Integrated Circuit) is a popular serial protocol ideal for communication between microcontrollers and various peripherals over short distances. Its most notable feature is that it only requires two wires to create a bus that can be shared by multiple devices:

- SDA (Serial Data Line): Transmits the actual data.
- SCL (Serial Clock Line): Carries the synchronizing clock signal.

The protocol supports both multi-master and multi-slave configurations, where each device on the bus is assigned a unique 7-bit or 10-bit address. Communication is initiated when a master sends a start condition, followed by the slave's address. The receiving device confirms receipt of each byte with an acknowledgment (ACK) signal. While I²C is slower than SPI, its simple wiring makes it extremely efficient for connecting numerous devices to a single microcontroller.

The image shows two side-by-side Arduino IDE windows. Both windows have the title bar 'Arduino Uno' and the status bar 'sketch_oct17d.ino' and 'sketch_oct17c.ino' respectively. The left window contains the code for the Master (sketch_oct17d.ino), which initializes the I2C bus as master, sends 'Hello Slave!' to the slave, and then requests data from the slave. The right window contains the code for the Slave (sketch_oct17c.ino), which initializes the I2C bus as slave, receives data from the master, and prints it to the Serial Monitor. Both windows have an 'Output' tab at the bottom showing the Serial Monitor. The Master's Serial Monitor shows the message 'Hello Master!*****' and 'Sent: Hello Slave!'. The Slave's Serial Monitor shows the received messages: 'Received: Hello Slave!', 'Received: Hello Slave!', 'Received: Hello Slave!', and 'Received: Hello Slave!'. The baud rate is set to 9600 in both monitors.

```

// Master (sketch_oct17d.ino)
void setup() {
    Wire.begin(); // Join I2C bus as master
    Serial.begin(9600);
    Serial.println("Master Ready");
}

void loop() {
    // Send data to slave
    Wire.beginTransmission(SLAVE_ADDR);
    Wire.write("Hello Slave!");
    Wire.endTransmission();
    Serial.println("Sent: Hello Slave!");

    delay(500);

    // Request data from slave
    Wire.requestFrom(SLAVE_ADDR, 20); // Request up to 20 bytes
    while (Wire.available()) {
        char c = Wire.read();
        Serial.print(c);
    }
    Serial.println();
    delay(1000);
}

```

```

// Slave (sketch_oct17c.ino)
#include <Wire.h>
#define SLAVE_ADDR 0x08

void setup() {
    Wire.begin(SLAVE_ADDR); // Join I2C bus as slave
    Wire.onRequest(sendData); // Function called when master requests data
    Wire.onReceive(receiveData); // Function called when master sends data
    Serial.begin(9600);
    Serial.println("Slave Ready");
}

String received = "";
void loop() {
    if (received != "") {
        Serial.print("Received: ");
        Serial.println(received);
        received = "";
    }
    delay(100);
}

void receiveData(int byteCount) {
    received = "";
    while (Wire.available()) {
        char c = Wire.read();
        received += c;
    }
}

```

4. Universal Serial Bus (USB)

The Universal Serial Bus (USB) is a standardized protocol designed to simplify the connection of peripheral devices—such as keyboards, mice, and storage drives—to computers. It is a host-controlled protocol, where the computer (the host) initiates and manages all communication with the connected peripherals (the devices).

Key features of USB include plug-and-play functionality, which allows devices to be automatically recognized and configured, and hot-swapping, which means devices can be connected or disconnected while the system is running. A standard USB cable contains four wires: two for power (VCC and GND) and two for differential data signals (D+ and D-). This versatile standard is widely used for its high-speed data transfer capabilities and its ability to deliver power to connected devices.

The screenshot shows the Arduino IDE interface. The top part displays the code for 'ledblinking.ino' in the 'Code' tab:

```
1 // Simple USB Serial Communication Example
2 void setup() {
3     Serial.begin(9600);          // Start USB serial communication
4     while (!Serial) {
5         ; // wait for the serial connection (only needed on some boards)
6     }
7     Serial.println("USB Communication Ready!");
8 }
9
10 void loop() {
11     // Send data to PC
12     Serial.println("Hello from Arduino!");
13     delay(1000);
14
15     // Check if any data is coming from PC
16     if (Serial.available() > 0) {
17         char received = Serial.read();
18         Serial.print("You sent: ");
19         Serial.println(received);
20     }
21 }
```

The bottom part shows the 'Serial Monitor' tab with the following output:

Output Serial Monitor X

```
hi
Hello from Arduino!
Hello from Arduino!
Hello from Arduino!
Hello from Arduino!
You sent: h
Hello from Arduino!
You sent: i
```

Result

The experiment successfully demonstrated the distinct characteristics of four primary serial communication protocols:

- SPI: Confirmed as a high-speed, synchronous, full-duplex protocol ideal for one-to-one or one-to-few connections where performance is critical. It uses four wires and a dedicated slave select line instead of addresses.
- I²C: Verified as a two-wire synchronous protocol that excels at connecting multiple slave devices to a single bus using a unique addressing scheme. It is simpler to wire than SPI for multi-device setups but operates at a lower speed.
- UART: Shown to be an effective asynchronous protocol for point-to-point, full-duplex communication. It relies on a pre-configured baud rate for synchronization and is commonly used for debugging and connecting modules like GPS or Bluetooth.
- USB: Implemented as a host-controlled protocol for connecting peripherals to a computer. Its architecture supports plug-and-play, power delivery, and high-speed, packet-based data transfer, making it a universal standard.

Conclusion

The experiment demonstrated and confirmed the functionality of USB, UART, I²C, and SPI communication protocols using the Arduino platform. It showed that Arduino can effectively handle various serial communication tasks, including USB-based PC interfacing and communication with microcontrollers and peripherals via UART, I²C, and SPI. The lab work highlights Arduino's suitability for embedded systems, whether for straightforward point-to-point connections or more complex multi-device networks. All four protocols were successfully tested using Arduino Uno and the Arduino IDE.