SPI PROTOCOL

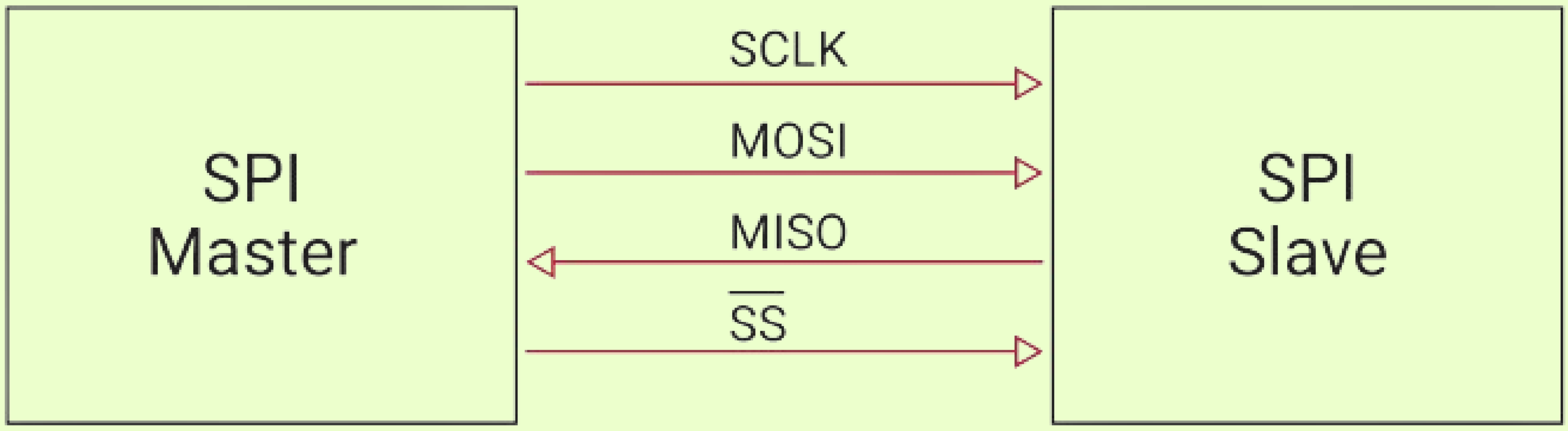
SWIPE

- **SPI** stands for Serial Peripheral interface.
- It is a **synchronous** communication protocol
- It is used to send data between **multiple devices.**
- It is organized into a **master and slave** configuration.
- The master has control over the slaves and the slaves receive instruction from the master.
- It offers a **higher data transfer rate** than many other types of communication interfaces

SPI SIGNALS

S.NO	**SIGNAL**	**DESCRIPTION**
1	MOSI	Master Out — Slave In
2	MISO	Master In — Slave Out
3	SCLK	Serial Clock
4	CS or SS	Chip Select or Slave Select

SPI WIRING



- These wires connect to the **same signal** on both devices, namely SCLK connects to SCLK, MOSI to MOSI, MISO to MISO, and SS to SS.

- In a multi-slave configuration, all signal lines are shared among all slaves, with the exception of the SS line which is independently controlled for each slave.

CLOCK SIGNAL

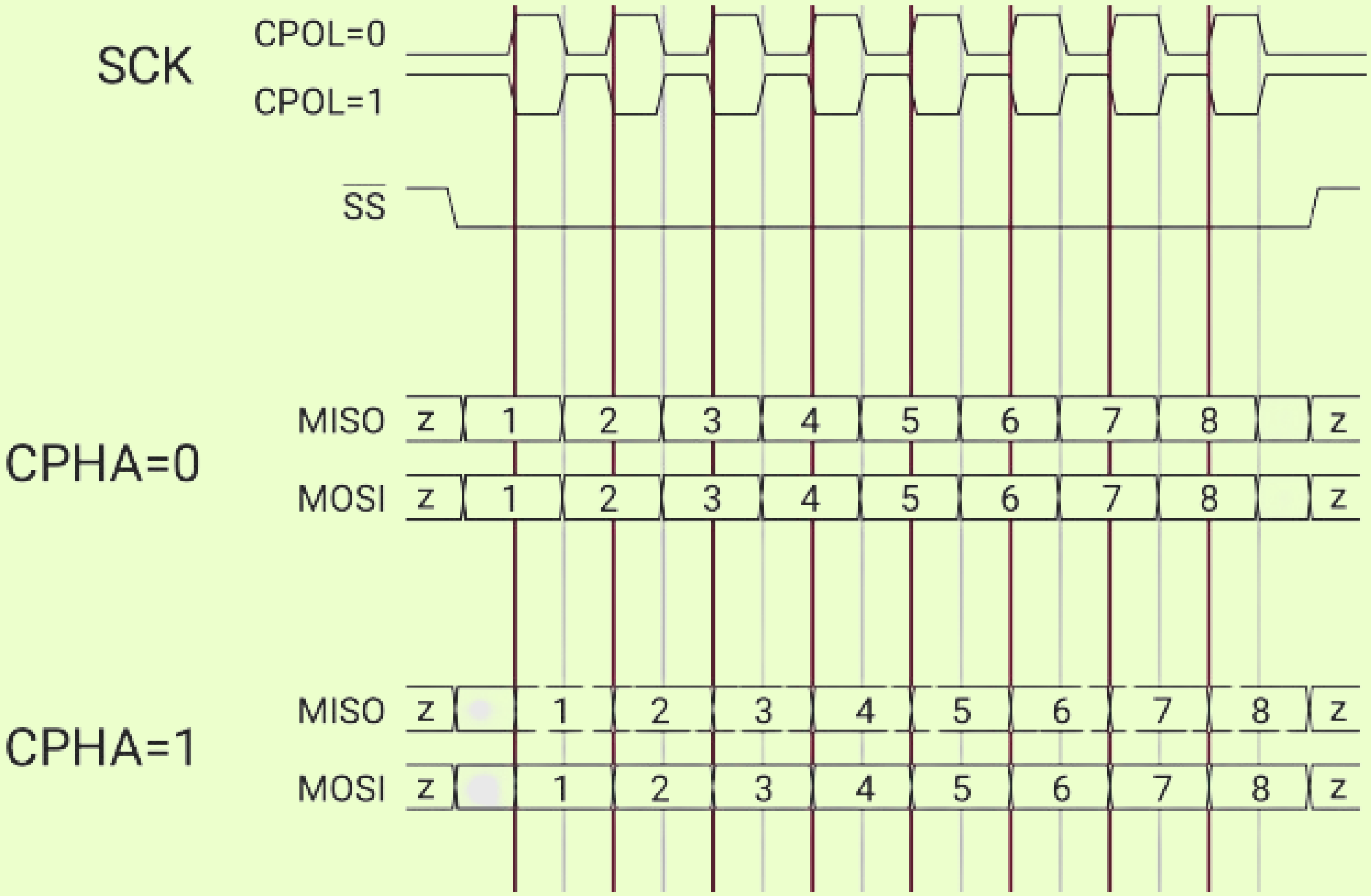
- The clock signal is generated by the master device to a specific frequency and is used to synchronize the data being transmitted and received between devices.

- This signal can be configured by the master by using two properties known as

1. Clock polarity (CPOL)
2. Clock phase (CPHA)

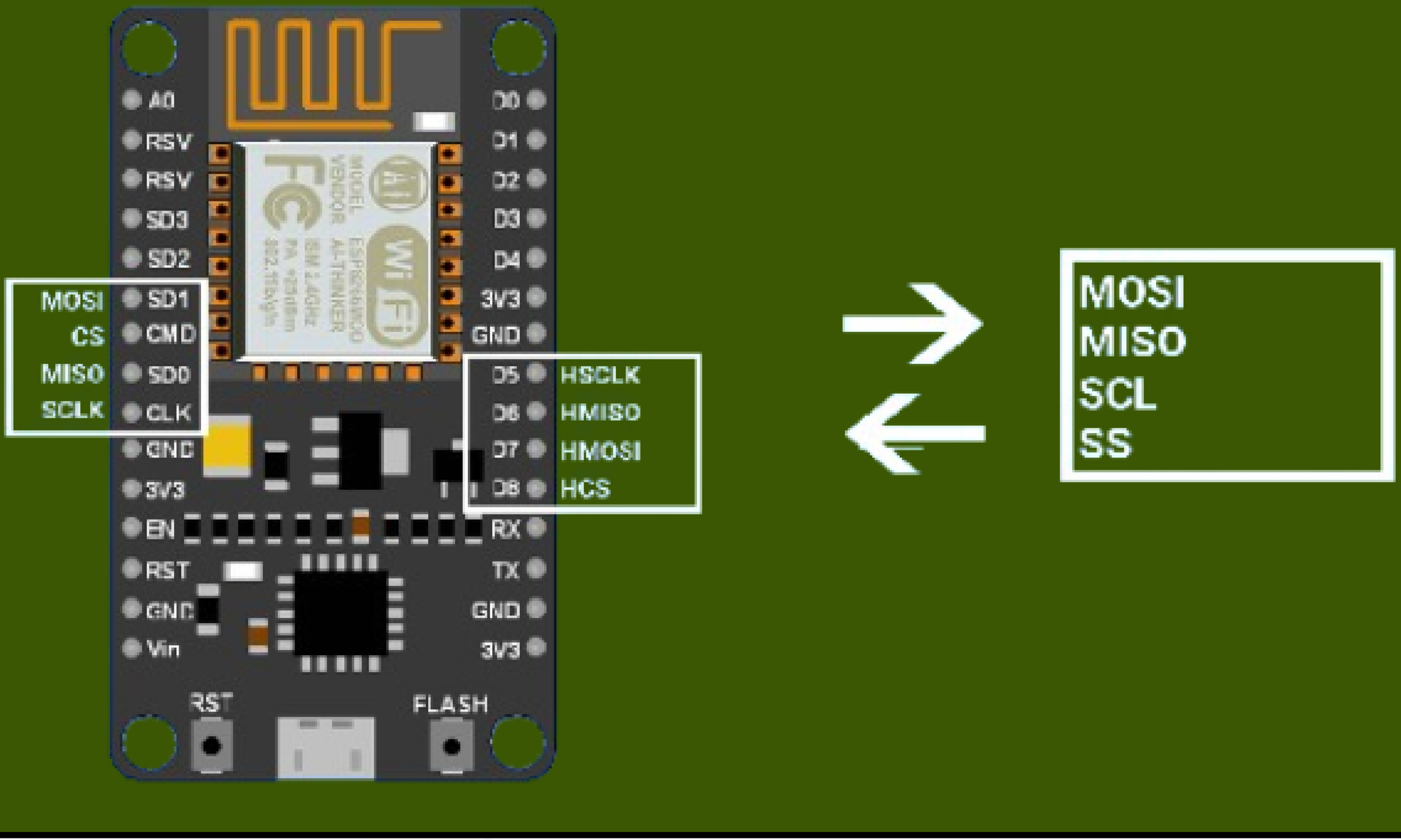
- Clock polarity determines the polarity of the clock signal and can be configured to idle either low (0) or high (1).

- A clock signal that idles low has a high pulse and a rising leading edge, whereas a clock signal that idles high has a low pulse and a falling leading edge

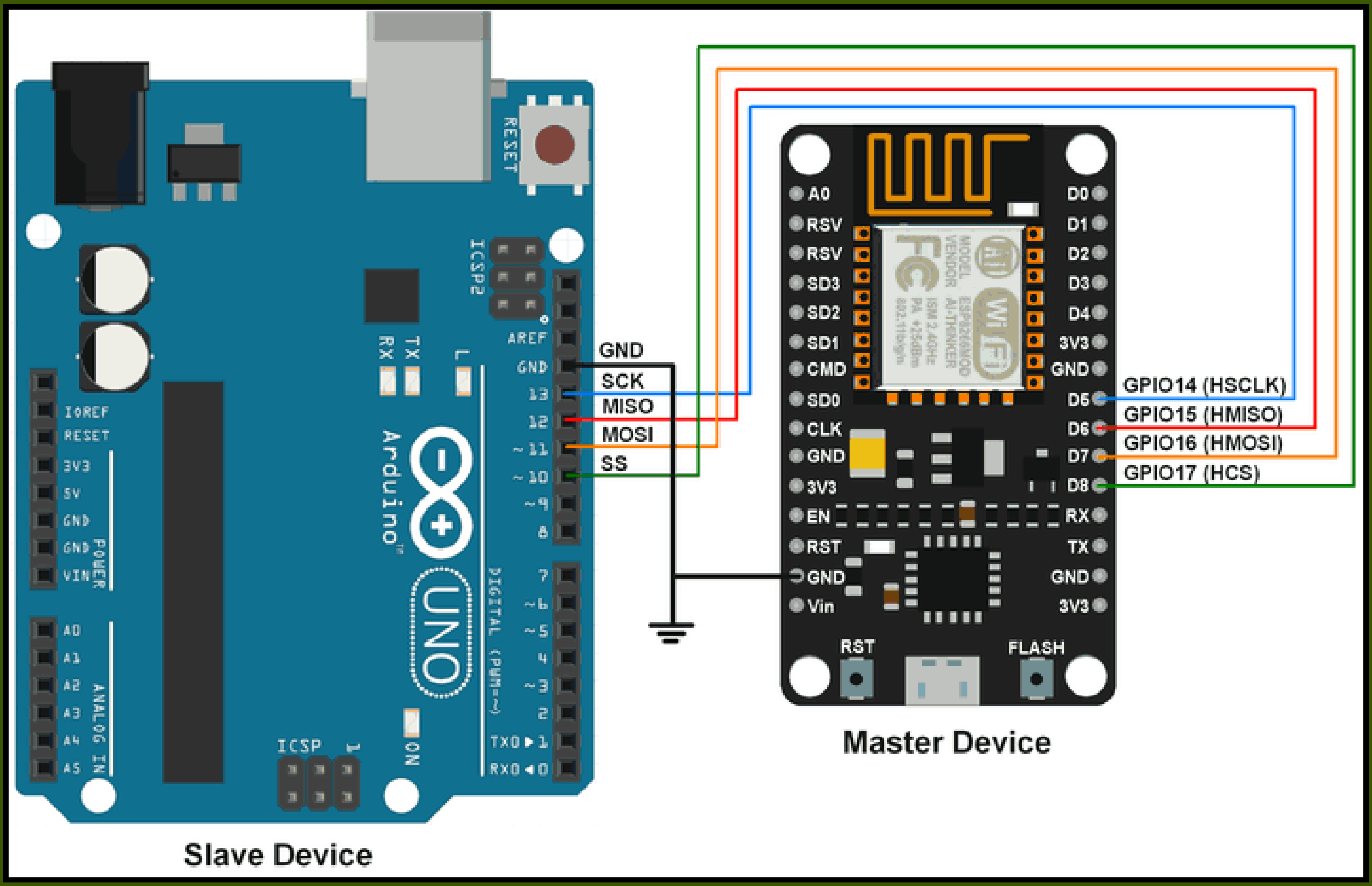


The clock phase determines the timing in which the data is to be modified and read. If the clock phase is set to zero, the data is modified on the trailing edge of the clock signal and the data is read on the leading edge. Conversely, if this property is set to one, data is changed on the leading edge of the clock signal and read on the trailing edge. As the clock cycles, data is sent bit by bit, simultaneously, over the MOSI and MISO lines.

NodeMCU SPI with Arduino IDE

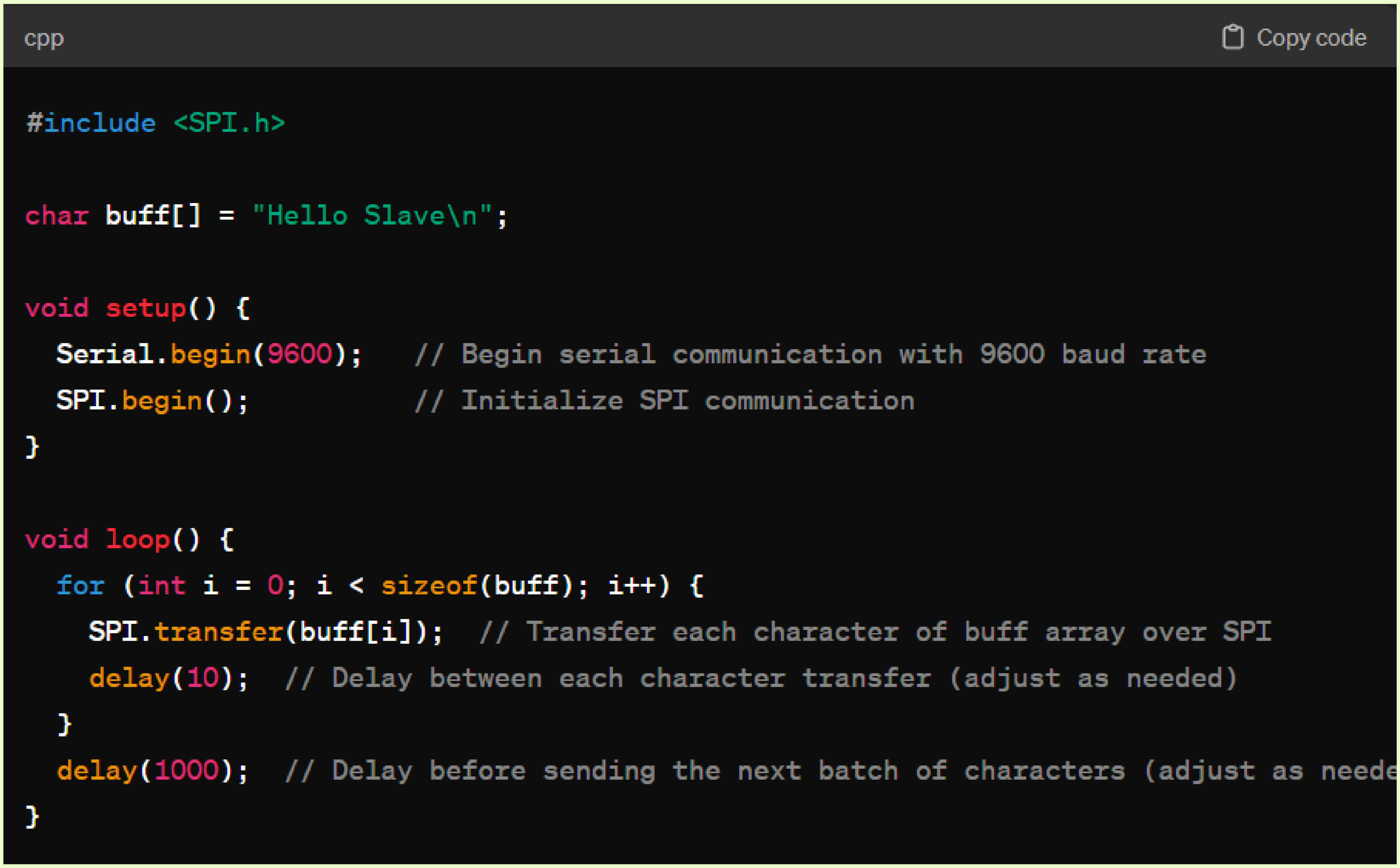


MOSI
MISO
SCL
SS



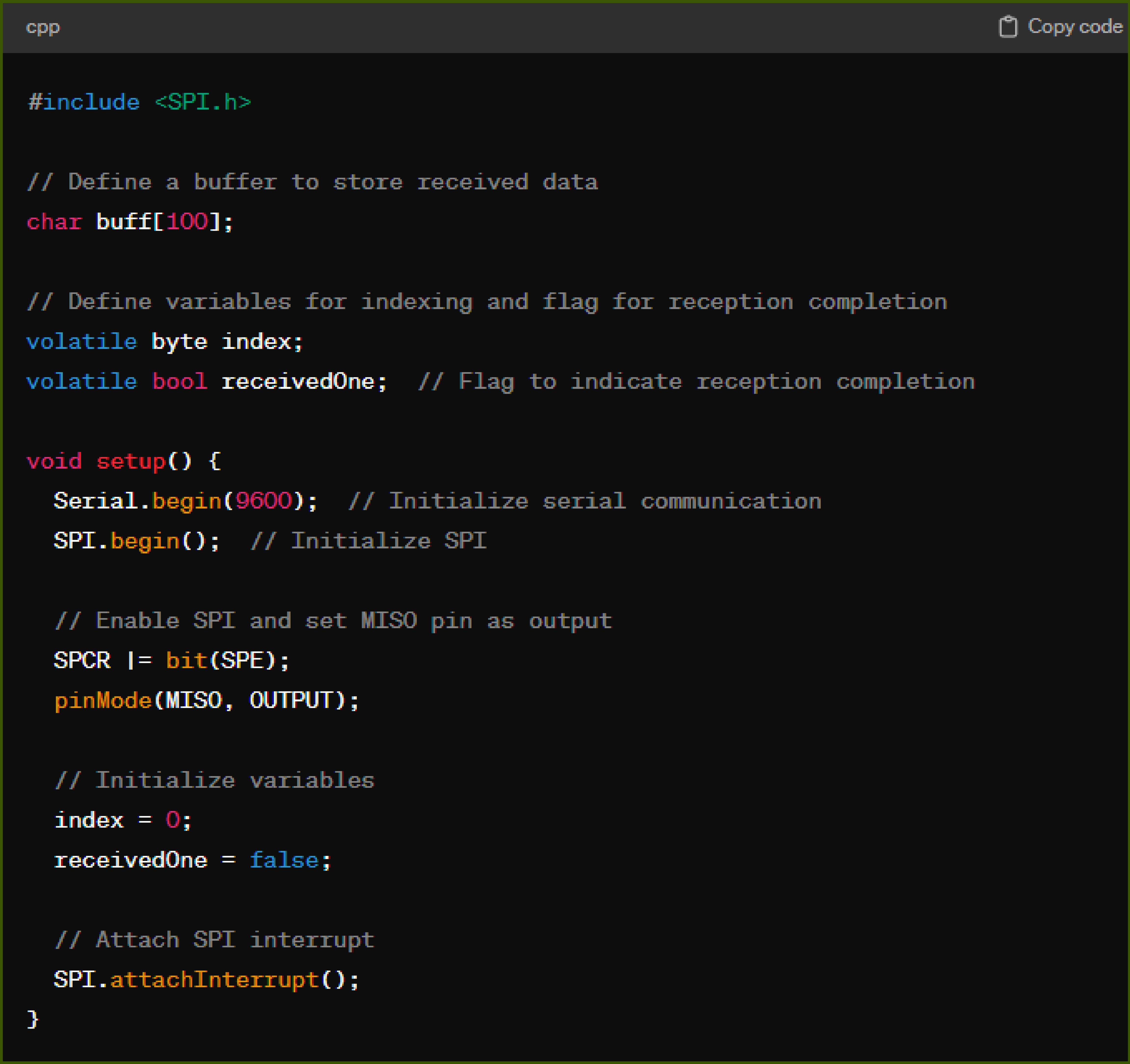
CODE

NodeMCU Master SPI Code using Arduino IDE

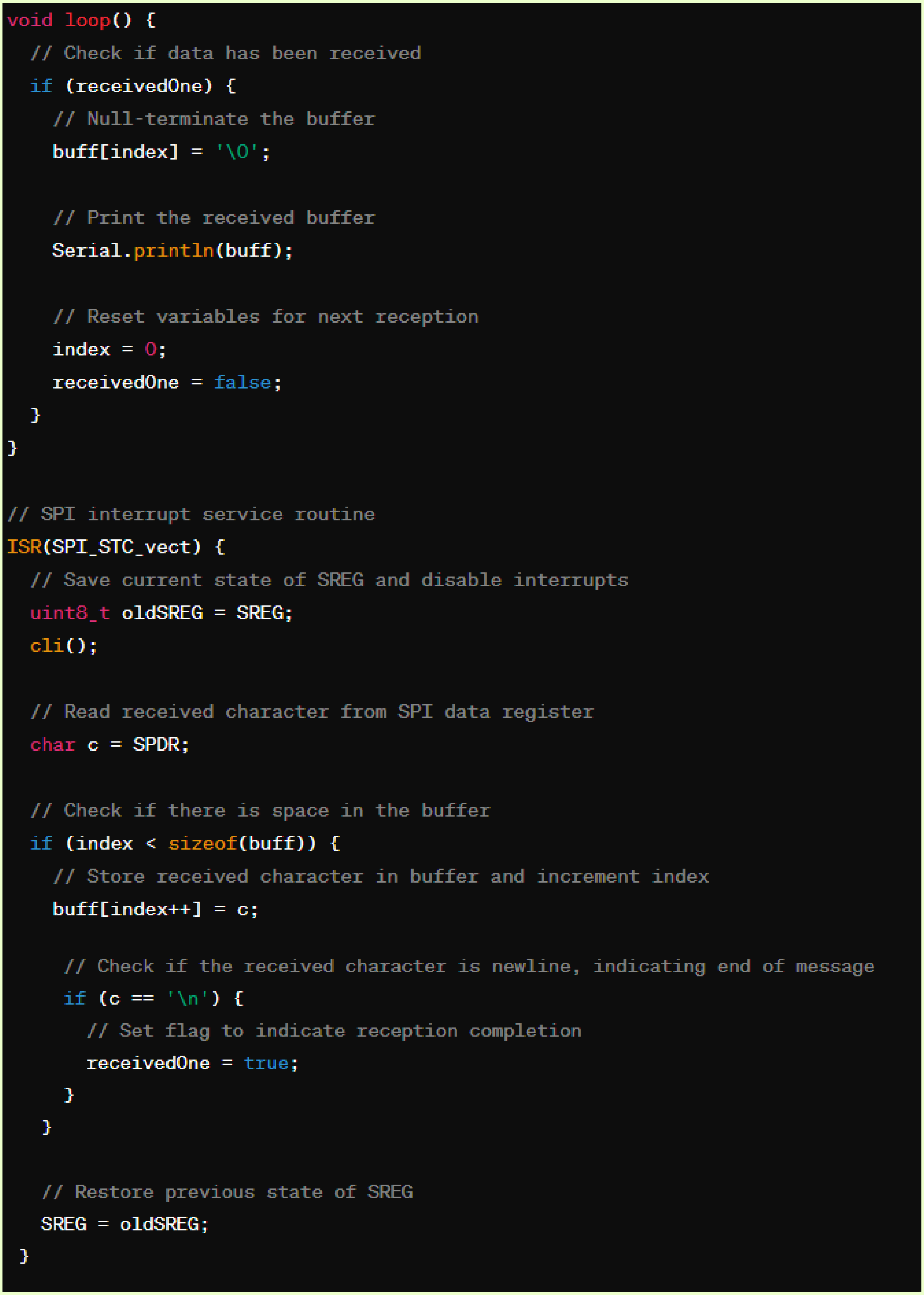


```cpp
#include <SPI.h>

char buff[] = "Hello Slave\n";

void setup() {
  Serial.begin(9600);   // Begin serial communication with 9600 baud rate
  SPI.begin();          // Initialize SPI communication
}

void loop() {
  for (int i = 0; i < sizeof(buff); i++) {
    SPI.transfer(buff[i]);  // Transfer each character of buff array over SPI
    delay(10);  // Delay between each character transfer (adjust as needed)
  }
  delay(1000);  // Delay before sending the next batch of characters (adjust as neede
}
```

Arduino Uno Slave SPI Code

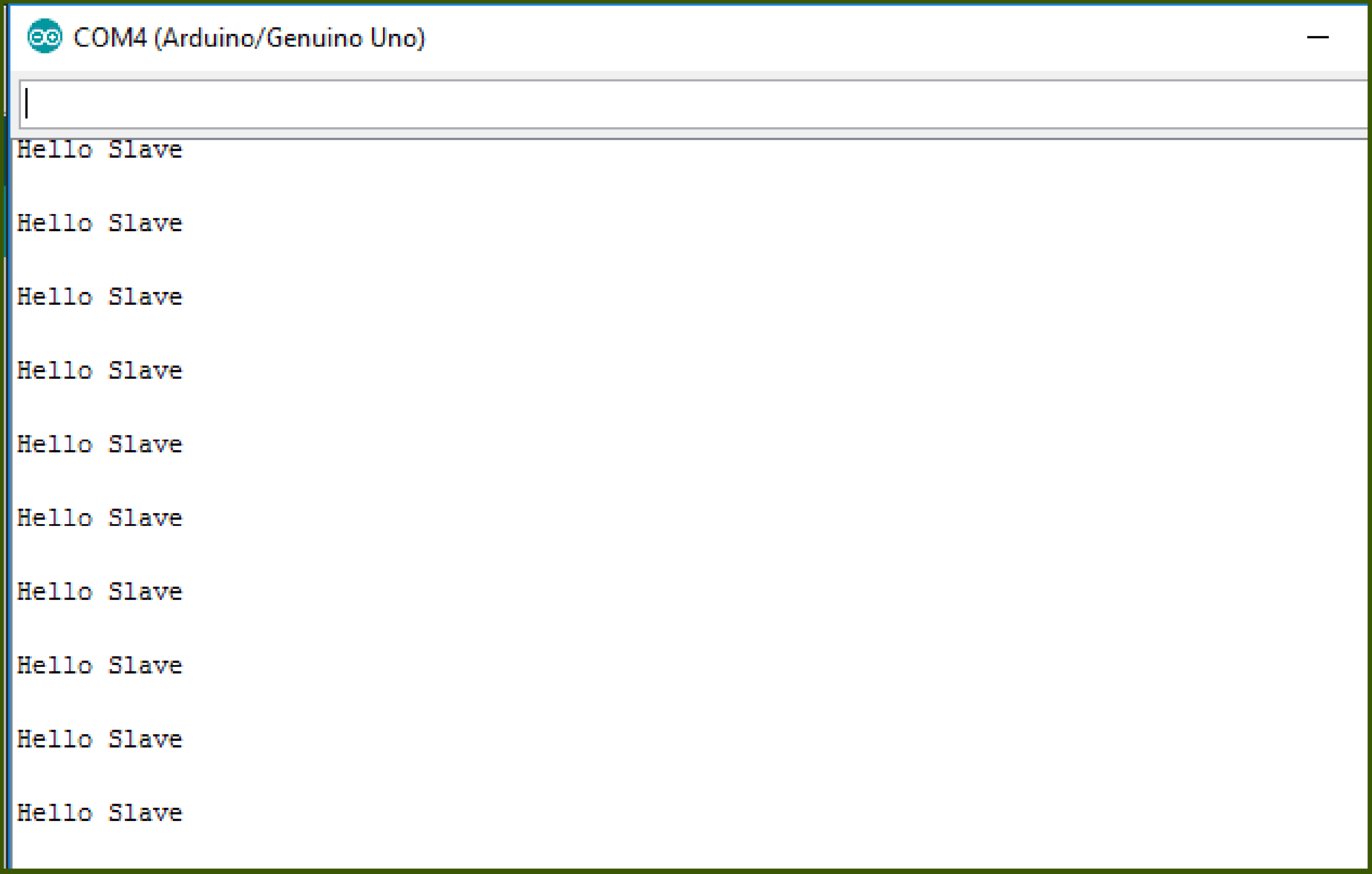


```cpp
#include <SPI.h>

// Define a buffer to store received data
char buff[100];

// Define variables for indexing and flag for reception completion
volatile byte index;
volatile bool receivedOne;  // Flag to indicate reception completion

void setup() {
  Serial.begin(9600);  // Initialize serial communication
  SPI.begin();  // Initialize SPI

  // Enable SPI and set MISO pin as output
  SPCR |= bit(SPE);
  pinMode(MISO, OUTPUT);

  // Initialize variables
  index = 0;
  receivedOne = false;

  // Attach SPI interrupt
  SPI.attachInterrupt();
}
```



```
void loop() {
  // Check if data has been received
  if (receivedOne) {
    // Null-terminate the buffer
    buff[index] = '\0';

    // Print the received buffer
    Serial.println(buff);

    // Reset variables for next reception
    index = 0;
    receivedOne = false;
  }
}

// SPI interrupt service routine
ISR(SPI_STC_vect) {
  // Save current state of SREG and disable interrupts
  uint8_t oldSREG = SREG;
  cli();

  // Read received character from SPI data register
  char c = SPDR;

  // Check if there is space in the buffer
  if (index < sizeof(buff)) {
    // Store received character in buffer and increment index
    buff[index++] = c;

     // Check if the received character is newline, indicating end of message
     if (c == '\n') {
       // Set flag to indicate reception completion
       receivedOne = true;
     }
   }

   // Restore previous state of SREG
   SREG = oldSREG;
 }
```

SLAVE OUTPUT



```
Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave

Hello Slave
```