

[illegible]

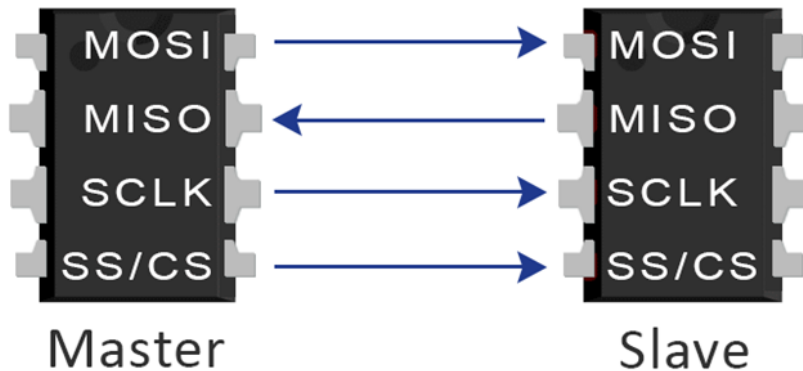
E-mail: manaskhatua@iitg.ac.in

“We suffer as a result of our own actions; it is unfair to blame anybody for it.” – Ma Sarada Devi

- SPI stands for **Serial Peripheral Interface**
 - It is a **Serial Bus Communication Protocol**
 - One of the most popular serial synchronous bus protocols (for short-distance communication)
 - its origin goes back to late 1980s developed by **Motorola**
 - Supports **high-speed synchronous data transfer**
 - Devices communicate in **master/slave mode**
 - **Master** device **initiates** the data communication
 - **Master** device generates a **Serial Clock** for synchronous
 - SPI can be multi-slave but it can't be multi-master
 - **Multiple slave devices** are allowed; each one connected to the master device via Slave Select line
 - The **master** always send **8 to 16 bits data** to the slave during the data transfer process
 - The **slave** always sends **8 bits data** to the master
 - SPI is a **full-duplex** master-slave communication protocol.
 - This means only a single master and a single slave can communicate on the bus at the same time.

SPI Buses

- SPI Interface uses **four wires** for communication.
 - Hence it is also known as a four-wire serial communication protocol.



4 interface pins:

- **MOSI** (Master Out-Slave In)
- **MISO** (Master In-Slave Out)
- **SCLK** (Serial Clock, which produces by the master)
- **SS** (Slave select or Chip select line which is used to select specific slave during the communication)

Pin Name: **MOSI**

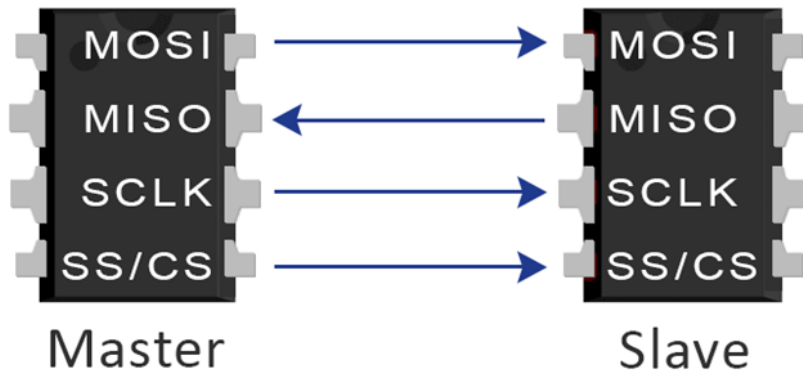
- The MOSI signal is a unidirectional signal used for transfer serial data from the Master to the Slave.
 - When a device is a **Master**, **serial data is output** on this signal.
 - When a device is a **Slave**, **serial data is input** on this signal.

Pin Name: **MISO**

- The MISO signal is a unidirectional signal used to transfer serial data from the slave to the master.
 - When a device is a **Slave**, **serial data is output** on this signal.
 - When a device is a **Master**, **serial data is input** on this signal.
 - When a slave device is **not selected**, the slave drives the signal with **high impedance**

SPI Buses

- SPI Interface uses **four wires** for communication.
 - Hence it is also known as a four-wire serial communication protocol.



4 interface pins:

- **MOSI** (Master Out-Slave In)
- **MISO** (Master In-Slave Out)
- **SCLK** (Serial Clock)
- **SS** (Slave select or Chip select)

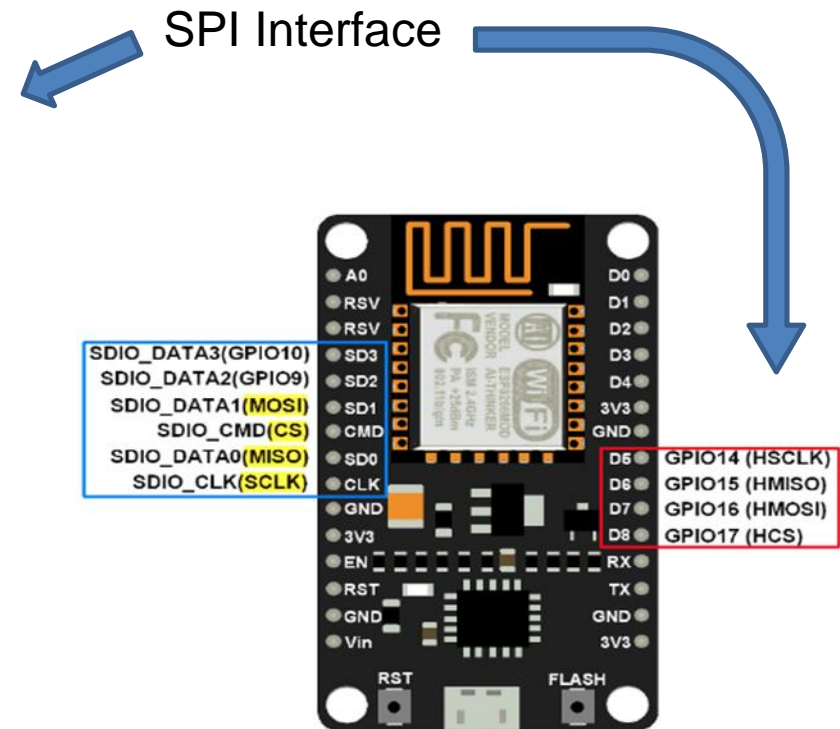
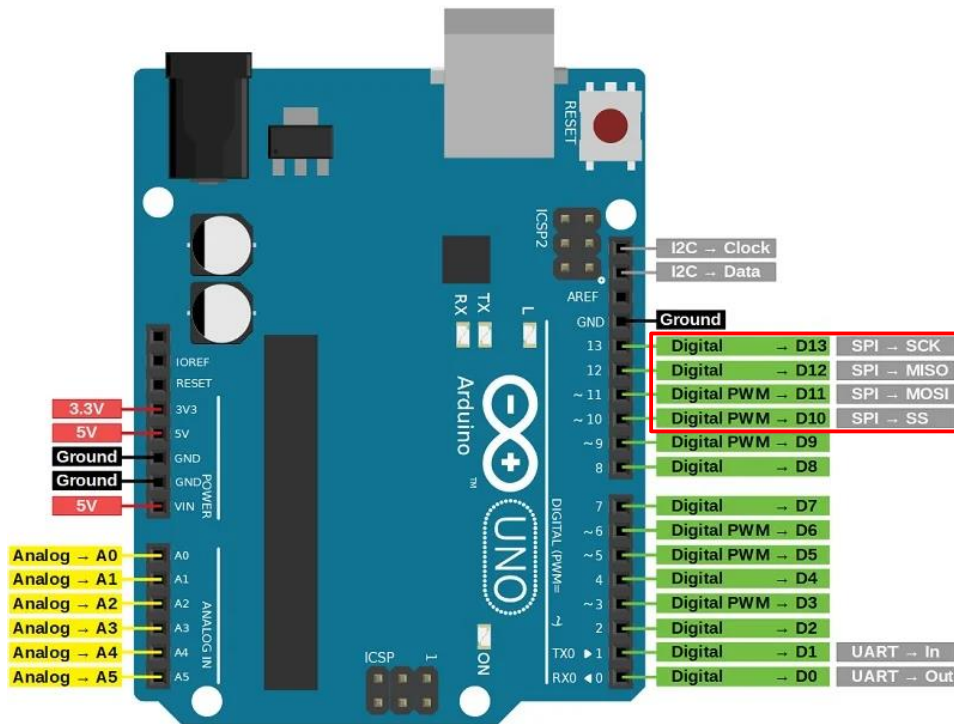
Pin Name: **SCLK / SCK**

- Clock signals were utilized by the SPI to synchronize data flow across the SPI Interface.
 - The **frequency of the clock signal** controls how quickly information is transferred as one bit of data is sent during each clock cycle.
 - Clock is only **active** during the data transfer.
 - Master device **configures and generates** the clock signal

Pin Name: **SS / CS**

- The Slave Select or Chip Select line is used to select specific slave during the communication.
 - Master Device can **handle** multiple slave devices on the bus by selecting them one by one
 - Because the SS line connects each slave with the master, there is no unique address for each slave like for the I2C communication

Arduino UNO and NodeMCU

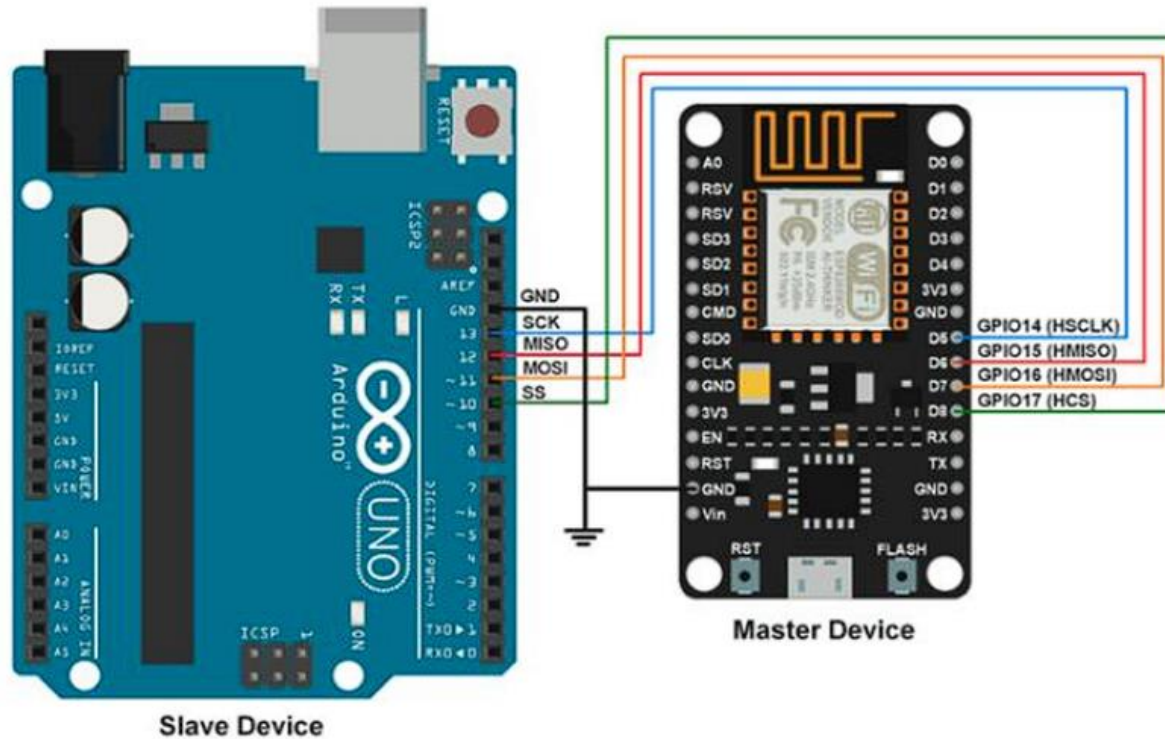


NodeMCU based ESP8266 has **Hardware SPI** with four pins available for SPI communication.

- With this SPI interface, we can connect any SPI enabled device with NodeMCU and make communication possible with it.

Note: ESP8266 has SPI pins (SD1, CMD, SD0, CLK) which are exclusively used for communication with flash memory on ESP-12E; hence, **they can't be used for SPI applications**.

Example



Source : <https://www.electronicwings.com/nodemcu/nodemcu-spi-with-arduino-ide>

Installing necessary packages

- Open Arduino IDE
 - Install package for Node MCU (ESP8266) inside Arduino IDE
 - Go to tools -> Board -> Select NodeMCU 1.0 (ESP-12E Module)
 - Go to tools -> Select port for NodeMCU (Ports are mentioned as COM1, COM2, COM3,... etc.)
 - Write the code for **NodeMCU as master device** and upload the code in Node MCU (See next page for code)
-
- Go to tools -> Board -> select Arduino UNO
 - Go to tools -> Select port for Arduino UNO
 - Write the code for **Arduino UNO as slave device** and upload the code in Arduino UNO (See next page for code)
 - Open the serial monitor for Arduino Uno to see the output

NodeMCU (Master Device Code)



```
#include<SPI.h>
```

```
char buff[]="Hello Arduino\n";
```

```
void setup() {  
    Serial.begin(9600);           /* begin serial with 9600 baud */  
    SPI.begin();                 /* begin SPI */  
}
```

```
void loop() {  
    for(int i=0; i<sizeof buff; i++) /* transfer buff data per second */  
        SPI.transfer(buff[i]);  
    delay(1000);  
}
```


Arduino Uno (Slave Device Code)



```
#include <SPI.h>
```

```
char buff [100];
```

```
volatile byte index;
```

```
volatile bool receivedone; /* use reception complete flag */
```

```
void setup (void){
```

```
    Serial.begin (9600);
```

```
    SPCR |= bit(SPE); /* Enable SPI */
```

```
    pinMode(MISO, OUTPUT); /* Make MISO pin as OUTPUT */
```

```
    index = 0;
```

```
    receivedone = false;
```

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

```
}
```

```
void loop (void){
```

```
    if (receivedone) /* Check and print received buffer if any */
```

```
    {
```

```
        buff[index] = 0;
```

```
        Serial.println(buff);
```

```
        index = 0;
```

```
        receivedone = false;
```

```
    }
```

```
}
```

```
// SPI interrupt routine
```

```
ISR (SPI_STC_vect){
```

```
    uint8_t oldsrgr = SREG;
```

```
    cli();
```

```
    char c = SPDR;
```

```
    if (index < sizeof buff)
```

```
    {
```

```
        buff [index++] = c;
```

```
        if (c == '\n'){ /* Check for newline character as end of msg */
```

```
            receivedone = true;
```

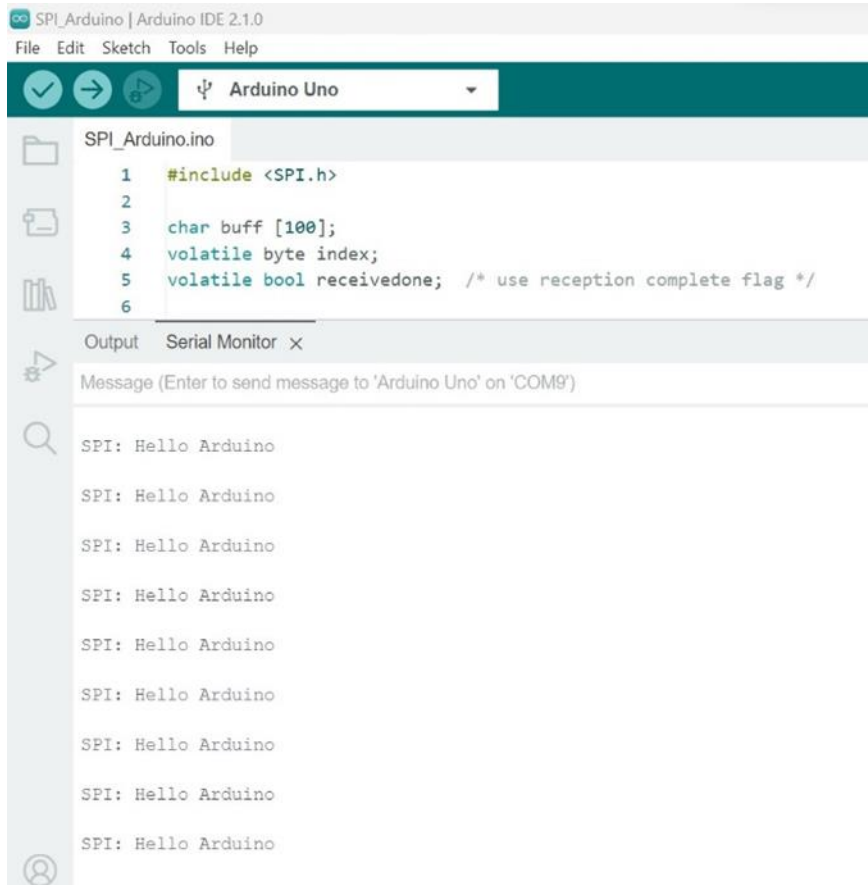
```
        }
```

```
    }
```

```
    SREG = oldsrgr;
```

```
}
```

Output



The screenshot displays the Arduino IDE 2.1.0 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for checking, running, and uploading, along with a dropdown menu set to 'Arduino Uno'. The main editor window shows the code for 'SPI_Arduino.ino' with the following content:

```
1  #include <SPI.h>
2
3  char buff [100];
4  volatile byte index;
5  volatile bool receivedone; /* use reception complete flag */
6
```

Below the code editor, the 'Serial Monitor' tab is active, showing a message input field and a list of received data. The received data consists of ten identical lines: 'SPI: Hello Arduino'.

Advantages of SPI



- ✓ High-speed communication for fast data transfer.
- ✓ Simultaneous full-duplex data transmission and reception.
- ✓ Low protocol overhead, efficient for simple data transfers.
- ✓ Efficient for communicating with multiple devices using individual Chip Select lines.
- ✓ Flexible clock polarity and phase for compatibility with various peripherals.
- ✓ Ideal for short-distance communication on PCBs.
- ✓ Real-time communication with precise timing due to synchronous nature.
- ✓ Widely supported by microcontrollers, processors, and peripherals.

Disadvantages of SPI



- ✓ Requires **more pins**, potentially leading to higher pin count.
- ✓ **Not ideal for longer distance** communication
- ✓ **Lacks built-in addressing**, manual differentiation needed for devices.
- ✓ **Doesn't support multiple masters** on the same bus.
- ✓ **Consume more power** due to continuous clock and full-duplex nature.
- ✓ Voltage level compatibility challenges between devices.
- ✓ Clock **synchronization complexities**, especially with different devices.
- ✓ Not suited for building large, complex device networks.

SPI Transfer formats - Clock Polarity & Phase



- SPI communication data is driven in 4 modes
 - It is decided by the combination of CPOL & CPHA.

- **Clock polarity(CPOL) :**

The CPOL bit controls the idle state value of the clock when no data is transferred.

This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state. So, in the SPI control register, there is a bit called CPOL, and you can make that pin as either a 0 or 1.

- **Clock phase(CPHA) :**

It decides the clock phase. CPHA controls at which clock edge that is the 1st or 2nd edge of SCLK, the slave should sample the data.

The combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

- CPOL & CPHA has to match with SPI slaves for proper data transfer.

SPI Transfer formats - Clock Polarity & Phase



SPI mode	Clock polarity (CPOL)	Clock phase (CPHA)	Clock Polarity in Idle State	Data is sampled on
0	0	0	Logic Low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic Low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic High	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic High	Data sampled on rising edge and shifted out on the falling edge

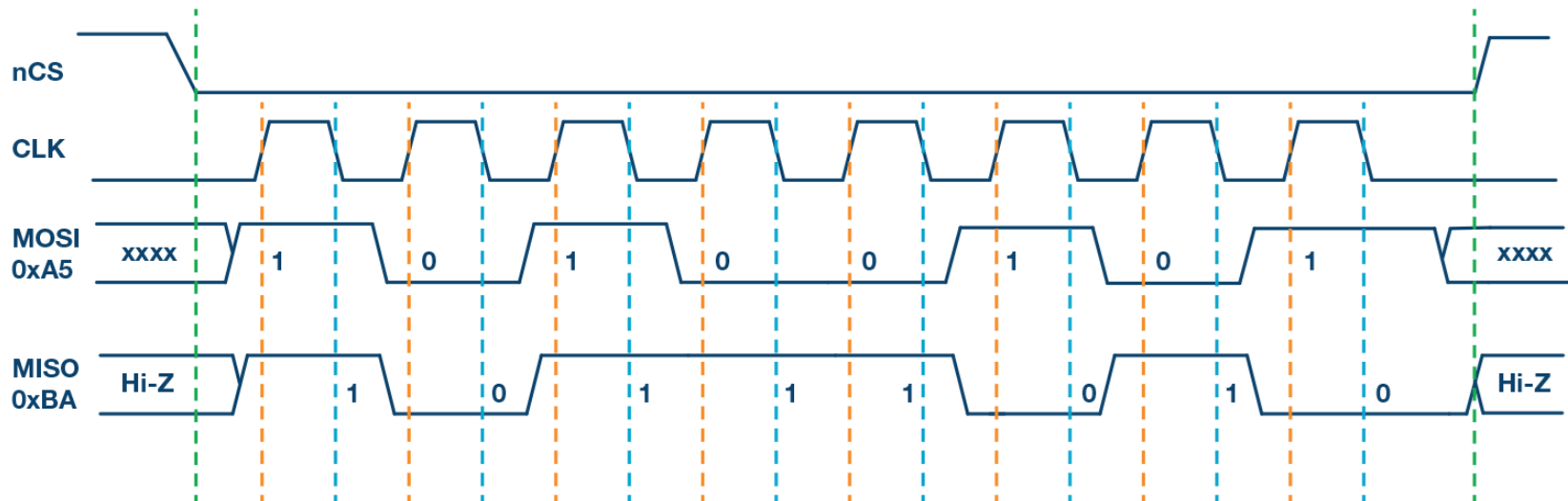
Mode 0

Data - MOSI and MISO line

Dotted green line - start and end of the transmission

Blue line - Shifting edge

Orange line - Sampling edge



SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.

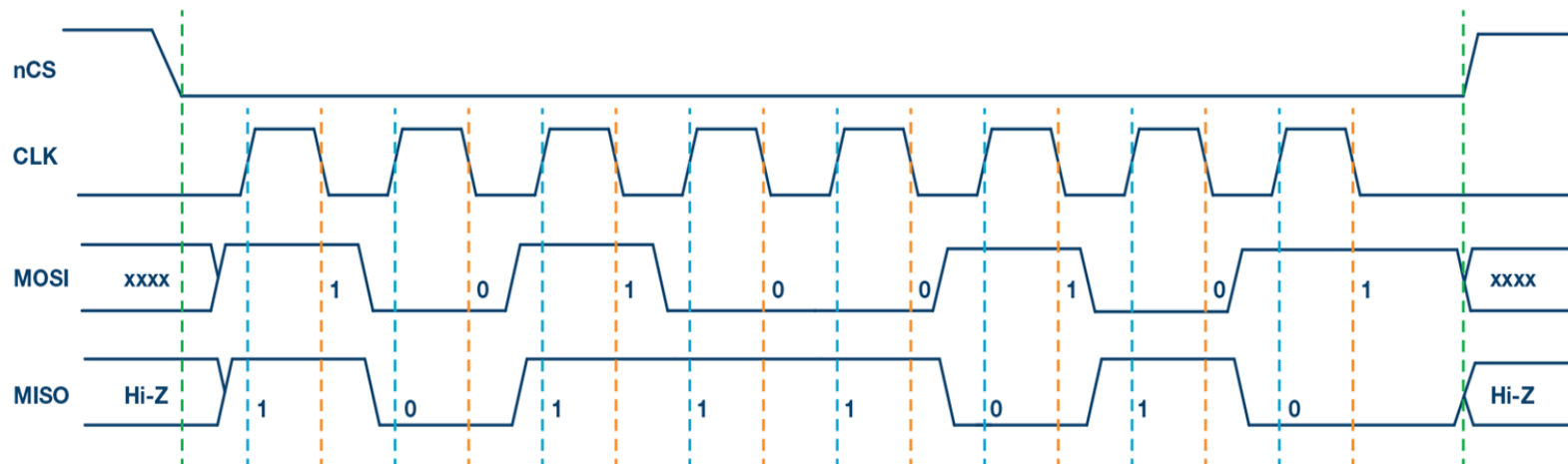
Mode 1

Data - MOSI and MISO line

Dotted green line - start and end of the transmission

Blue line - Shifting edge

Orange line - Sampling edge



SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on falling edge and shifted on rising edge.

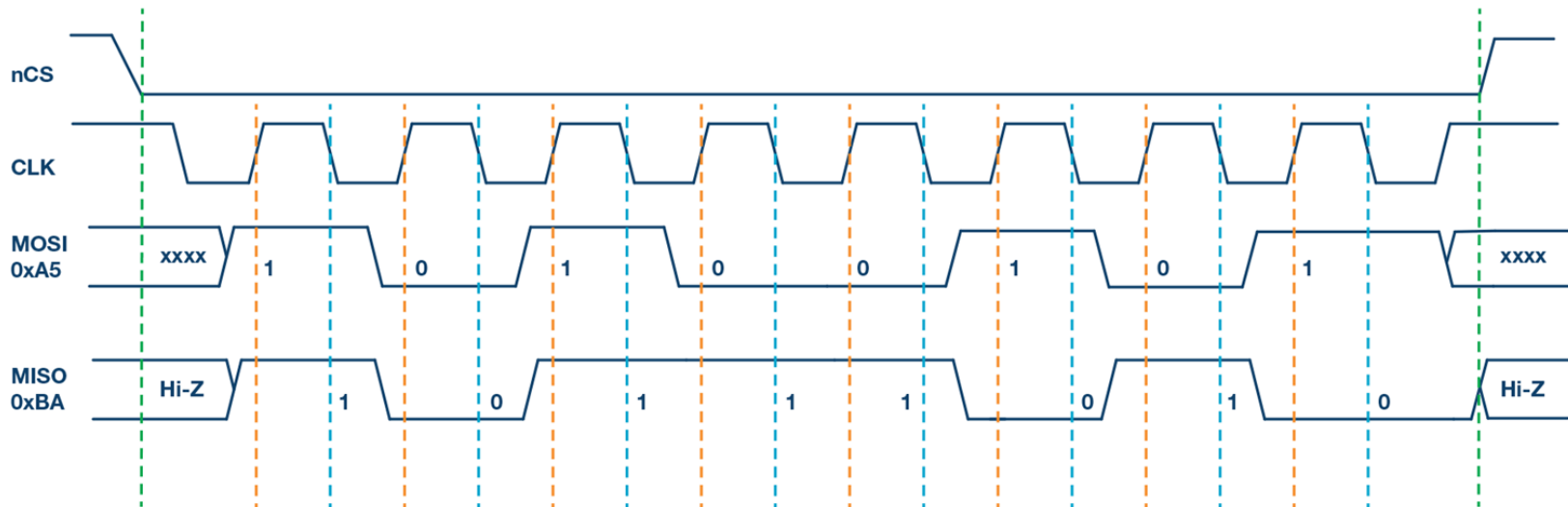
Mode 2

Data - MOSI and MISO line

Dotted green line - start and end of the transmission

Blue line - Shifting edge

Orange line - Sampling edge



SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on rising edge and shifted on falling edge.

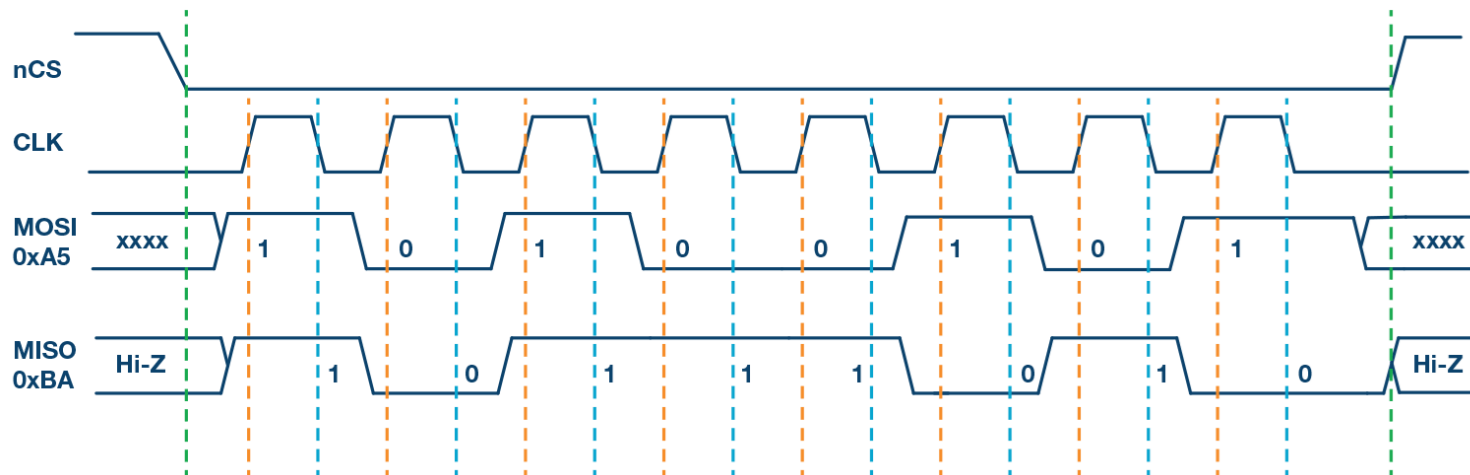
Mode 3

Data - MOSI and MISO line

Dotted green line - start and end of the transmission

Blue line - Shifting edge

Orange line - Sampling edge



SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on falling edge and shifted on rising edge.

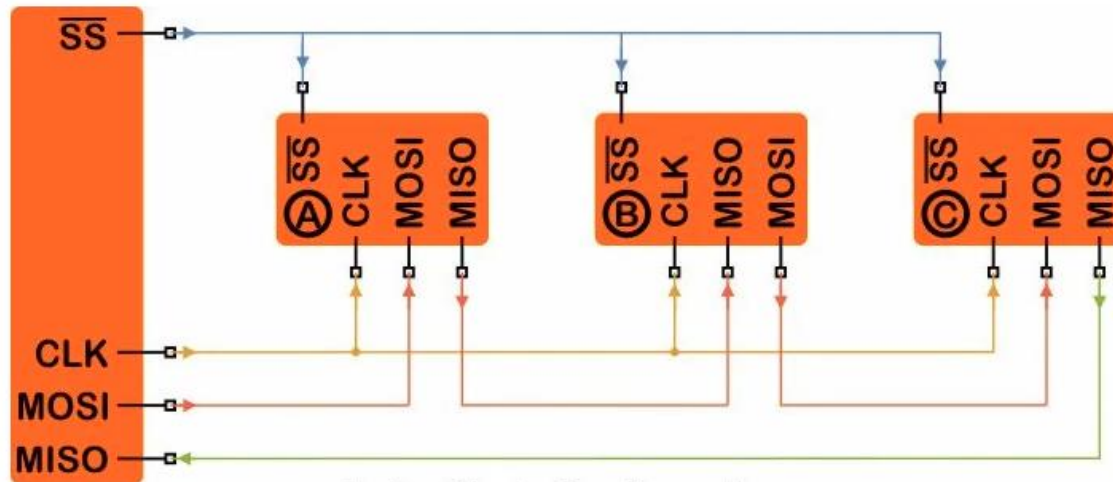
Daisy-Chain Configuration

In this configuration, data moves from one device to the next. The final slave device can return data to the master.

In the daisy-chain configuration, all slaves share a common slave-select line.

Data is shifted out of the master into the first slave, and then out of the first slave into the second, and so on.

The data cascades down the line until the last slave in the series, which can then use its MISO line to send data to the master device.



Thanks!

