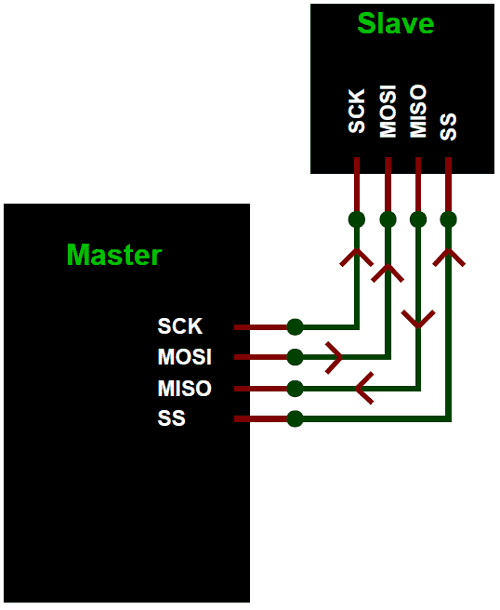
### **What is SPI?**

**SPI (Serial Peripheral Interface)** is a serial communication protocol. SPI interface was found by Motorola in 1970. SPI has a full duplex connection, which means that the data is sent and received simultaneously. That is a master can send data to slave and a slave can send data to master simultaneously. SPI is synchronous serial communication means the clock is required for communication purpose.

Working of SPI

A SPI has a master/Slave communication by using four lines. A SPI can have only one master and can have multiple slaves. A master is usually a microcontroller and the slaves can be a microcontroller, sensors, ADC, DAC, LCD etc.

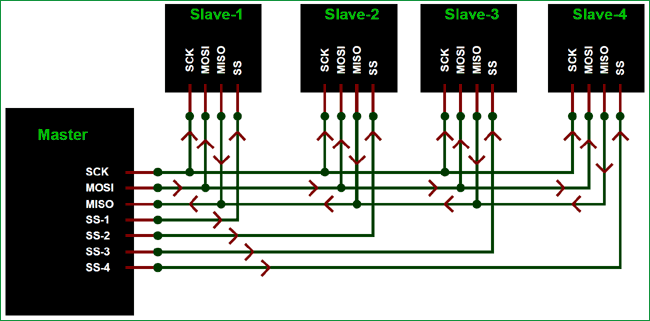
Below is the block diagram representation of SPI **Master with Single Slave**.



**SPI has following four lines MISO, MOSI, SS, and CLK**

* **MISO (Master in Slave Out)** - The Slave line for sending data to the master.
* **MOSI (Master Out Slave In)** - The Master line for sending data to the peripherals.
* **SCK (Serial Clock)** - The clock pulses which synchronize data transmission generated by the master.
* **SS (Slave Select)**–Master can use this pin to enable and disable specific devices.

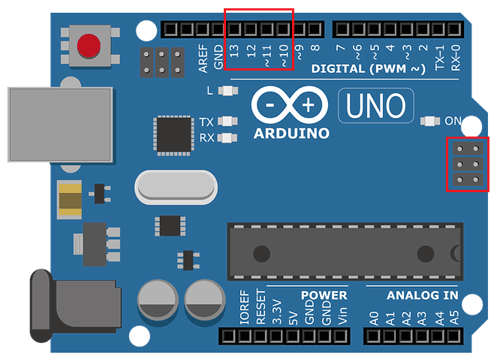
**SPI Master with Multiple Slaves**



**To start communication between master and slave** we need to set the required device's Slave Select (SS) pin to LOW, so that it can communicate with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines of master. As you can see in the above image there are four slaves in which the SCLK, MISO, MOSI are common connected to master and the SS of each slave is connected separately to individual SS pins (SS1, SS2, SS3) of master. By setting the required SS pin LOW a master can communicate with that slave.

SPI Pins in Arduino UNO

The image below shows the SPI pins present Arduino UNO (in red box).



|  |  |
| --- | --- |
| **SPI Line** | **Pin in Arduino** |
| MOSI | 11 or ICSP-4 |
| MISO | 12 or ICSP-1 |
| SCK | 13 or ICSP-3 |
| SS | 10 |

Using SPI in Arduino

Before start programming for **SPI communication between two Arduinos**. We need to learn about the SPI library used in Arduino IDE.

The library **<SPI.h>** is included in the program for using the following functions for SPI communication.

**1. SPI.begin()**

**USE:** To Initialize the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

**2. SPI.setClockDivider(divider)**

**USE:** To Set the SPI clock divider relative to the system clock. The available dividers are 2, 4, 8, 16, 32, 64 or 128.

Dividers:

* SPI\_CLOCK\_DIV2
* SPI\_CLOCK\_DIV4
* SPI\_CLOCK\_DIV8
* SPI\_CLOCK\_DIV16
* SPI\_CLOCK\_DIV32
* SPI\_CLOCK\_DIV64
* SPI\_CLOCK\_DIV128

**3. SPI.attachInterrupt(handler)**

**USE:** This function is called when a slave device receives data from the master.

**4. SPI.transfer(val)**

**USE:** This function is used to simultaneous send and receive the data between master and slave.

(SPI)

A Serial Peripheral Interface (SPI) bus is a system for serial communication, which uses up to four conductors, commonly three. One conductor is used for data receiving, one for data sending, one for synchronization and one alternatively for selecting a device to communicate with. It is a full duplex connection, which means that the data is sent and received simultaneously. The maximum baud rate is higher than that in the I2C communication system.

Board SPI Pins

SPI uses the following four wires −

* **SCK** − This is the serial clock driven by the master.
* **MOSI** − This is the master output / slave input driven by the master.
* **MISO** − This is the master input / slave output driven by the master.
* **SS** − This is the slave-selection wire.

The following functions are used. You have to include the SPI.h.

* **SPI.begin()** − Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.
* **SPI.setClockDivider(divider)** − To set the SPI clock divider relative to the system clock. On AVR based boards, the dividers available are 2, 4, 8, 16, 32, 64 or 128. The default setting is SPI\_CLOCK\_DIV4, which sets the SPI clock to one-quarter of the frequency of the system clock (5 Mhz for the boards at 20 MHz).
* **Divider** − It could be (SPI\_CLOCK\_DIV2, SPI\_CLOCK\_DIV4, SPI\_CLOCK\_DIV8, SPI\_CLOCK\_DIV16, SPI\_CLOCK\_DIV32, SPI\_CLOCK\_DIV64, SPI\_CLOCK\_DIV128).
* **SPI.transfer(val)** − SPI transfer is based on a simultaneous send and receive: the received data is returned in receivedVal.
* **SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode))** − speedMaximum is the clock, dataOrder(MSBFIRST or LSBFIRST), dataMode(SPI\_MODE0, SPI\_MODE1, SPI\_MODE2, or SPI\_MODE3).

We have four modes of operation in SPI as follows −

* **Mode 0 (the default)** − Clock is normally low (CPOL = 0), and the data is sampled on the transition from low to high (leading edge) (CPHA = 0).
* **Mode 1** − Clock is normally low (CPOL = 0), and the data is sampled on the transition from high to low (trailing edge) (CPHA = 1).
* **Mode 2** − Clock is normally high (CPOL = 1), and the data is sampled on the transition from high to low (leading edge) (CPHA = 0).
* **Mode 3** − Clock is normally high (CPOL = 1), and the data is sampled on the transition from low to high (trailing edge) (CPHA = 1).
* **SPI.attachInterrupt(handler)** − Function to be called when a slave device receives data from the master.

Now, we will connect two Arduino UNO boards together; one as a master and the other as a slave.

* (SS) : pin 10
* (MOSI) : pin 11
* (MISO) : pin 12
* (SCK) : pin 13

The ground is common. Following is the diagrammatic representation of the connection between both the boards −

Let us see examples of SPI as Master and SPI as Slave.

SPI as MASTER

Example

#include <SPI.h>

void setup (void) {

Serial.begin(115200); //set baud rate to 115200 for usart

digitalWrite(SS, HIGH); // disable Slave Select

SPI.begin ();

SPI.setClockDivider(SPI\_CLOCK\_DIV8);//divide the clock by 8

}

void loop (void) {

char c;

digitalWrite(SS, LOW); // enable Slave Select

// send test string

for (const char \* p = "Hello, world!\r" ; c = \*p; p++) {

SPI.transfer (c);

Serial.print(c);

}

digitalWrite(SS, HIGH); // disable Slave Select

delay(2000);

}

SPI as SLAVE

Example

#include <SPI.h>

char buff [50];

volatile byte indx;

volatile boolean process;

void setup (void) {

Serial.begin (115200);

pinMode(MISO, OUTPUT); // have to send on master in so it set as output

SPCR |= \_BV(SPE); // turn on SPI in slave mode

indx = 0; // buffer empty

process = false;

SPI.attachInterrupt(); // turn on interrupt

}

ISR (SPI\_STC\_vect) // SPI interrupt routine {

byte c = SPDR; // read byte from SPI Data Register

if (indx < sizeof buff) {

buff [indx++] = c; // save data in the next index in the array buff

if (c == '\r') //check for the end of the word

process = true;

}

}

void loop (void) {

if (process) {

process = false; //reset the process

Serial.println (buff); //print the array on serial monitor

indx= 0; //reset button to zero

}

}