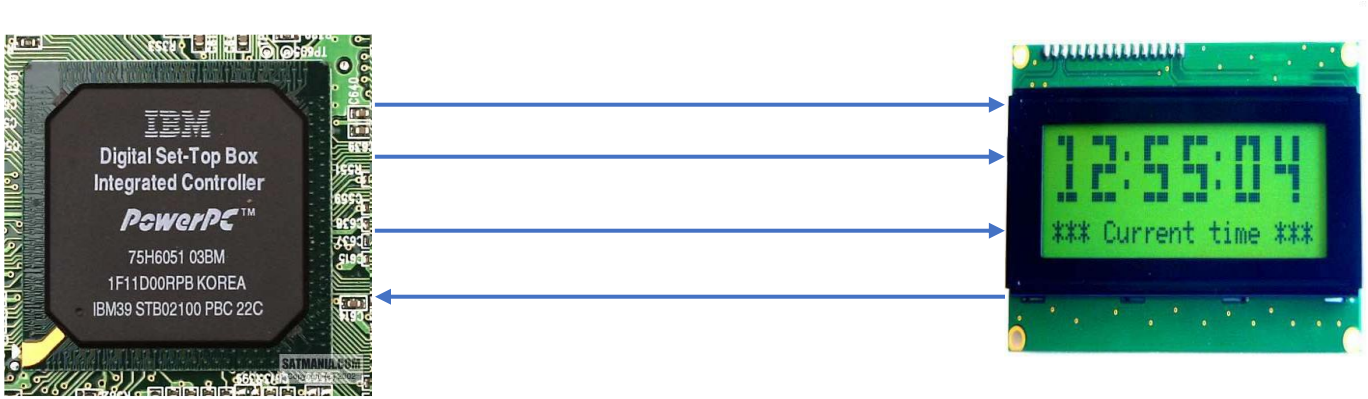
Serial Peripheral Interface (SPI):

What is it?

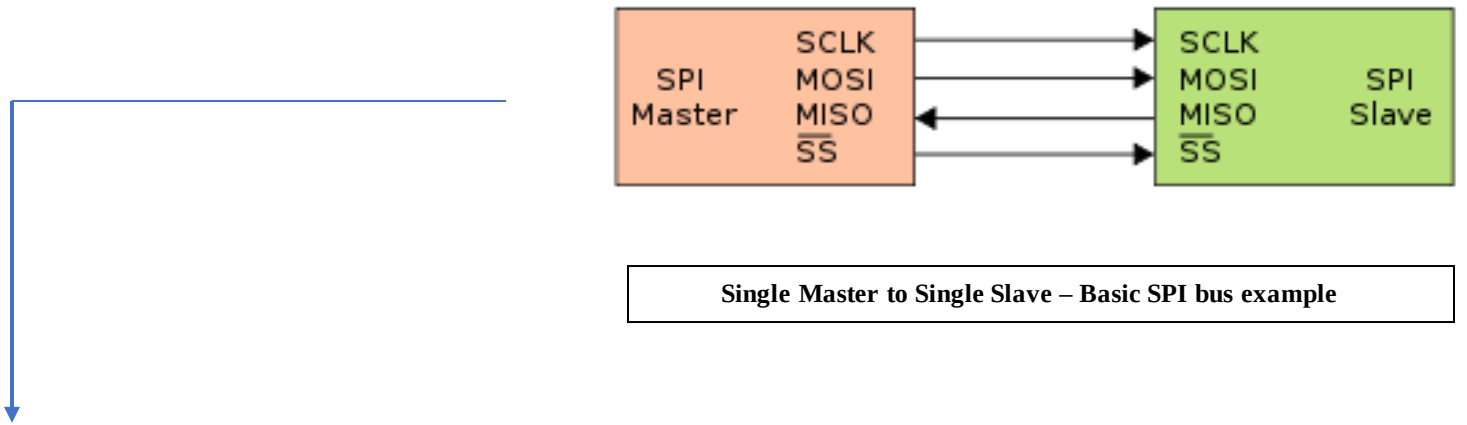
The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems.

The SPI is a synchronous serial interface in which data in an 8-bit byte can be shifted in and/or out one bit at a time. It can be used to communicate with a serial peripheral device or with another microcontroller with an SPI interface.



SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS), sometimes called chip select (CS), lines.

Sometimes SPI is called a *four-wire* serial bus, contrasting with three-, two-, and one-wire serial buses. The SPI may be accurately described as a synchronous serial interface, but it is different from the Synchronous Serial Interface (SSI) protocol, which is also a four-wire synchronous serial communication protocol. The SSI protocol employs differential signaling and provides only a single simplex communication channel. SPI is one master and multi slave communication.



Single Master to Single Slave – Basic SPI bus example

Bus wires:

- Master-Out, Slave-In (MOSI) - carries data out of master to slave
- Master-In, Slave-Out (MISO) - carries data out of slave to master
  (Both MOSI and MISO are active during every transmission)
- System Clock (SCLK) - produced by master to synchronize transfers
- SS (or CS) – unique line to select each slave chip

SPI Basics:

- Master asserts slave/chip select line
- Master generates clock signal
- Shift registers shift data in and out

The SPI bus specifies four logic signals:

Serial Clock:

- SCLK: SCK

Master Output → Slave Input (MOSI):

- SIMO, MTSR - correspond to MOSI on both master and slave devices, connects to each other
- SDI, DI, DIN, SI - on slave devices; connects to MOSI on master, or to below connections
- SDO, DO, DOUT, SO - on master devices; connects to MOSI on slave, or to above connections

Master Input ← Slave Output (MISO):

- SOMI, MRST - correspond to MISO on both master and slave devices, connects to each other
- SDO, DO, DOUT, SO - on slave devices; connects to MISO on master, or to below connections
- SDI, DI, DIN, SI - on master devices; connects to MISO on slave, or to above connections

Slave Select:

- SS: $\overline{SS}$, SSEL, CS, $\overline{CS}$, CE, nSS, /SS, SS#

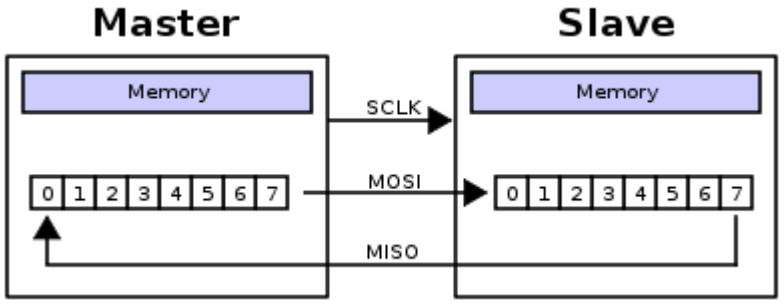
In other words, MOSI (or SDO on a master) connects to MOSI (or SDI on a slave). MISO (or SDI on a master) connects to MISO (or SDO on a slave). Slave Select is the same functionality as chip select and is used instead of an addressing concept. Pin names are always capitalized as in Slave Select, Serial Clock, and Master Output Slave Input.

Operation:

The SPI bus can operate with a single master device and with one or more slave devices.

If a single slave device is used, the SS pin *may* be fixed to logic low if the slave permits it. Some slaves require a falling edge of the chip select signal to initiate an action. An example is the Maxim MAX1242 ADC, which starts conversion on a high→low transition. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (*electrically disconnected*) when the device is not selected. Devices without tri-state outputs cannot share SPI bus segments with other devices without using an external tri-state buffer.



A typical hardware setup using two shift registers to form an inter-chip circular buffer

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz. The master then selects the slave device with a logic level 0 on the select line. If a waiting period is required, such as for an analog-to-digital conversion, the master must wait for at least that period of time before issuing clock cycles.

During each SPI clock cycle, a full-duplex data transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

Transmissions normally involve two shift registers of some given word-size, such as eight bits, one in the master and one in the slave; they are connected in a virtual ring topology. Data is usually shifted out with the most significant bit first. On the clock edge, both master and slave shift out a bit and output it on the transmission line to the counterpart. On the next clock edge, at each receiver the bit is sampled from the transmission line and set as a new least-significant bit of the shift register. After the register bits have been shifted out and in, the master and slave have exchanged register values. If more data needs to be exchanged, the shift registers are reloaded and the process repeats. Transmission may continue for any number of clock cycles. When complete, the master stops toggling the clock signal, and typically deselects the slave.

Transmissions often consist of eight-bit words. However, other word-sizes are also common, for example, sixteen-bit words for touch-screen controllers or audio codecs, such as the TSC2101 by Texas Instruments, or twelve-bit words for many digital-to-analog or analog-to-digital converters.

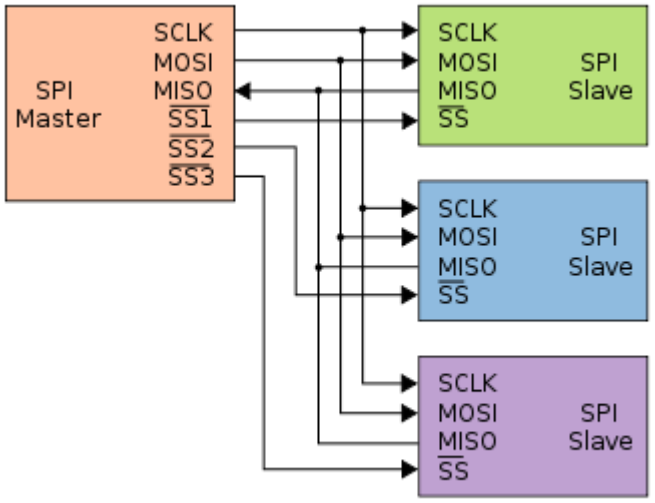
Every slave on the bus that has not been activated using its chip select line must disregard the input clock and MOSI signals and should not drive MISO (I.E. must have a tristate output) although some devices need external tristate buffers to implement this.

SPI Bus Architecture:

1. **Independent slave configuration:**

In the independent slave configuration, there is an independent chip select line for each slave. This is the way SPI is normally used. The master asserts only one chip select at a time.

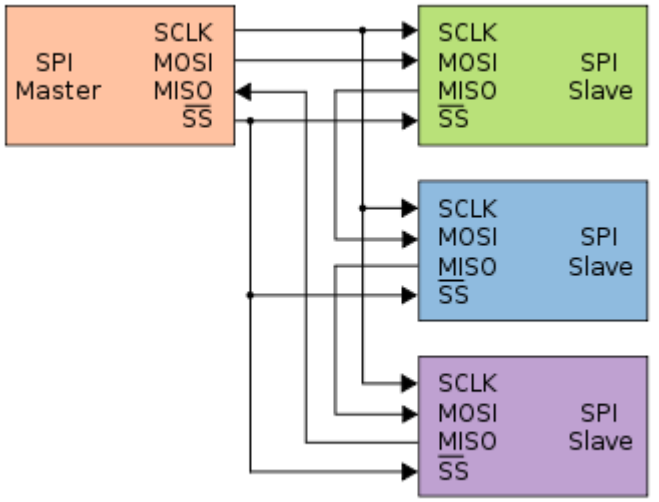
Pull-up resistors between power source and chip select lines are recommended for systems where the master's chip select pins may default to an undefined state.[3] When separate software routines initialize each chip select and communicate with its slave, pull-up resistors prevent other uninitialized slaves from responding.



Typical SPI bus: master and three independent slaves

Since the MISO pins of the slaves are connected together, they are required to be tri-state pins (high, low or high-impedance), where the high-impedance output must be applied when the slave is not selected. Slave devices not supporting tri-state may be used in independent slave configuration by adding a tri-state buffer chip controlled by the chip select signal.[3] (Since only a single signal line needs to be tristated per slave, one typical standard logic chip that contains four tristate buffers with independent gate inputs can be used to interface up to four slave devices to an SPI bus.)

2. Daisy chain configuration:



Daisy-chained SPI bus: master and cooperative slaves

Some products that implement SPI may be connected in a daisy chain configuration, the first slave output being connected to the second slave input, etc. The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of the data it received during the first group of clock pulses. The whole chain acts as a communication shift register; daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Each slave copies input to output in the next clock cycle until active low SS line goes high. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.

Other applications that can potentially interoperate with SPI that require a daisy chain configuration include SGPIO, JTAG, and Two Wire Interface.

Applications:

The board real estate savings compared to a parallel I/O bus are significant, and have earned SPI a solid role in embedded systems. That is true for most system-on-a-chip processors, both with higher end 32-bit processors such as those using ARM, MIPS, or PowerPC and with other microcontrollers such as the AVR, PIC, and MSP430. These chips usually include SPI controllers capable of running in either master or slave mode. In-system programmable AVR controllers (including blank ones) can be programmed using a SPI interface.

Chip or FPGA based designs sometimes use SPI to communicate between internal components; on-chip real estate can be as costly as its on-board cousin.

The full-duplex capability makes SPI very simple and efficient for single master/single slave applications. Some devices use the full-duplex mode to implement an efficient, swift data stream for applications such as digital audio, digital signal processing, or telecommunications channels, but most off-the-shelf chips stick to half-duplex request/response protocols.

SPI is used to talk to a variety of peripherals, such as

- Sensors: temperature, pressure, ADC, touchscreens, video game controllers
- Control devices: audio codecs, digital potentiometers, DAC
- Camera lenses: Canon EF lens mount
- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
- Memory: flash and EEPROM

- Real-time clocks
- LCD, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant[6])

For high-performance systems, FPGAs sometimes use SPI to interface as a slave to a host, as a master to sensors, or for flash memory used to bootstrap if they are SRAM-based.

SPI library from Arduino:

This library allows you to communicate with SPI devices, with the Arduino as the master device.

To use this library
#include <SPI.h>

A Brief Introduction to the Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically, there are three lines common to all the devices:

- 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

and one line specific for every device:

- SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

When a device's Slave Select pin is low, it communicates 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.

To write code for a new SPI device you need to note a few things:

- What is the maximum SPI speed your device can use? This is controlled by the first parameter in SPISettings. If you are using a chip rated at 15 MHz, use 15000000. Arduino will automatically use the best speed that is equal to or less than the number you use with SPISettings.
- Is data shifted in Most Significant Bit (MSB) or Least Significant Bit (LSB) first? This is controlled by second SPISettings parameter, either MSBFIRST or LSBFIRST. Most SPI chips use MSB first data order.
- Is the data clock idle when high or low? Are samples on the rising or falling edge of clock pulses? These modes are controlled by the third parameter in SPISettings.

The SPI standard is loose and each device implements it a little differently. This means you have to pay special attention to the device's datasheet when writing your code.

Generally speaking, there are four modes of transmission. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal (called the clock phase), and whether the clock is idle when high or low (called the clock polarity). The four modes combine polarity and phase according to this table:

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	Output Edge	Data Capture
SPI_MODE0	0	0	Falling	Rising
SPI_MODE1	0	1	Rising	Falling
SPI_MODE2	1	0	Rising	Falling
SPI_MODE3	1	1	Falling	Rising

Once you have your SPI parameters, use SPI.beginTransaction() to begin using the SPI port. The SPI port will be configured with your all of your settings. The simplest and most efficient way to use SPISettings is directly inside SPI.beginTransaction(). For example:

SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));

If other libraries use SPI from interrupts, they will be prevented from accessing SPI until you call SPI.endTransaction(). The SPI settings are applied at the begin of the transaction and SPI.endTransaction() doesn't change SPI settings. Unless you, or some library, calls beginTransaction a second time, the setting are maintained. You should attempt to minimize the time between before you call SPI.endTransaction(), for best compatibility if your program is used together with other libraries which use SPI.

With most SPI devices, after SPI.beginTransaction(), you will write the slave select pin LOW, call SPI.transfer() any number of times to transfer data, then write the SS pin HIGH, and finally call SPI.endTransaction().

For more on SPI, see Wikipedia's page on SPI.

Connections:

The following table display on which pins the SPI lines are broken out on the different Arduino boards:

Arduino / Genuino Board	MOSI	MISO	SCK	SS (slave)	SS (master)	Level
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	-	5V
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	-	5V
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-	5V
Due	SPI-4	SPI-1	SPI-3	-	4, 10, 52	3,3V
Zero	ICSP-4	ICSP-1	ICSP-3	-	-	3,3V
101	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	10	3,3V
MKR1000	8	10	9	-	-	3,3V

Note that MISO, MOSI, and SCK are available in a consistent physical location on the ICSP header; this is useful, for example, in designing a shield that works on every board.

Note about Slave Select (SS) pin on AVR based boards

All AVR based boards have an SS pin that is useful when they act as a slave controlled by an external master. Since this library supports only master mode, this pin should be set always as OUTPUT otherwise the SPI interface could be put automatically into slave mode by hardware, rendering the library inoperative.

It is, however, possible to use any pin as the Slave Select (SS) for the devices. For example, the Arduino Ethernet shield uses pin 4 to control the SPI connection to the on-board SD card, and pin 10 to control the connection to the Ethernet controller.

Examples

- Barometric Pressure Sensor: Read air pressure and temperature from a sensor using the SPI protocol.
- Digital Pot Control: Control a AD5206 digital potentiometer using the SPI protocol.

Some good references for more data on SPI:

http://www.gammon.com.au/forum/?id=10892 **- Details about Program that uses SPI for Arduino**

http://www.arduino.cc/en/Tutorial/SPIEEPROM **- Info of Variables used in Arduino program for SPI**