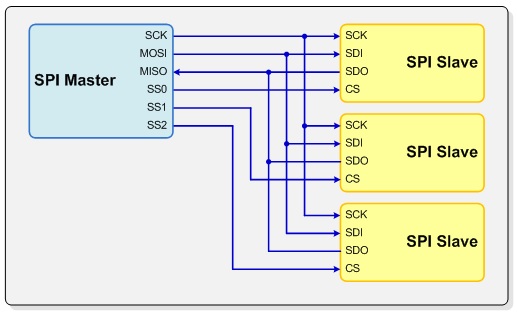
# SPI Interface

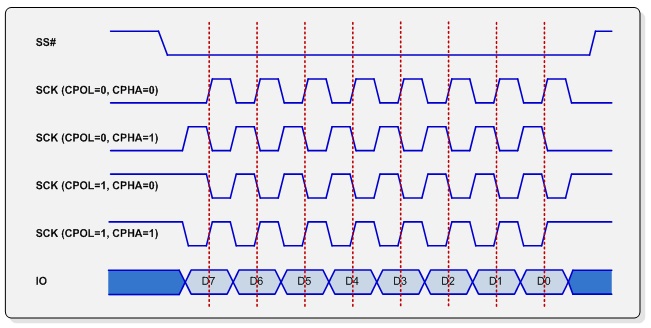
The Serial Peripheral Interface (SPI) bus was developed by Motorola to provide full-duplex synchronous serial communication between master and slave devices. The SPI bus is commonly used for communication with flash memory, sensors, real-time clocks (RTCs), analog-to-digital converters, and more.

As shown in Figure 1, standard SPI masters communicate with slaves using the serial clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS) lines. The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.

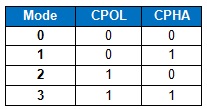
**  
Figure 1. 4-wire SPI bus configuration with multiple slaves**

**Polarity and Clock Phase**

The SPI interface defines no protocol for data exchange, limiting overhead and allowing for high speed data streaming. Clock polarity (CPOL) and clock phase (CPHA) can be specified as ‘0’ or ‘1’ to form four unique modes to provide flexibility in communication between master and slave as shown in Figure 2.

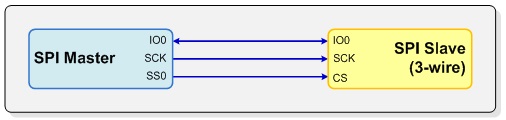
**  
Figure 2. SPI bus timing**

If CPOL and CPHA are both ‘0’ (defined as Mode 0) data is sampled at the leading rising edge of the clock. Mode 0 is by far the most common mode for SPI bus slave communication. If CPOL is ‘1’ and CPHA is ‘0’ (Mode 2), data is sampled at the leading falling edge of the clock. Likewise, CPOL = ‘0’ and CPHA = ‘1’ (Mode 1) results in data sampled at on the trailing falling edge and CPOL = ‘1’ with CPHA = ‘1’ (Mode 3) results in data sampled on the trailing rising edge. Table 1 below summarizes the available modes.

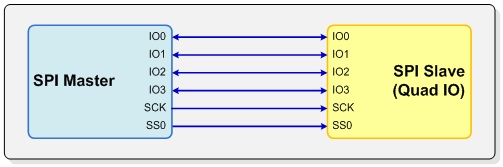
**  
Table 1. SPI mode definitions**

**SPI Bus 3-Wire and Multi-IO Configurations**

In addition to the standard 4-wire configuration, the SPI interface has been extended to include a variety of IO standards including 3-wire for reduced pin count and dual or quad I/O for higher throughput.   
  
In 3-wire mode, MOSI and MISO lines are combined to a single bidirectional data line as shown in Figure 3. Transactions are half-duplex to allow for bidirectional communication. Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput; many 3-wire devices have low performance requirements and are instead designed with low pin count in mind.

**  
Figure 3. 3-wire SPI configuration with one slave**

Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput. Components that utilize multi I/O modes can rival the read speed of parallel devices while still offering reduced pin counts. This performance increase enables random access and direct program execution from flash memory (execute-in-place).   
  
Quad I/O devices can, for example, offer up to 4 times the performance of a standard 4-wire SPI interface when communicating with a high speed device. Figure 4 shows an example of a single quad IO slave configuration.

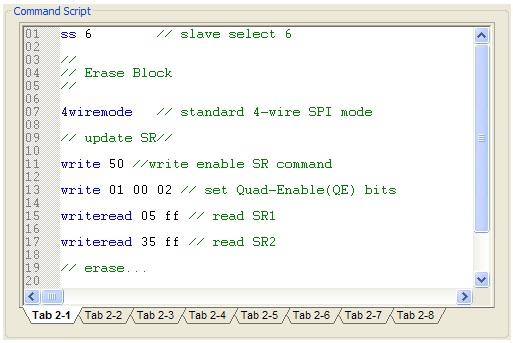
**  
Figure 4. Quad IO SPI configuration with one slave**

## SPI Bus Transactions

The SPI protocol does not define the structure of the data stream; the composition of data is completely up to the component designer. However, many devices follow the same basic format for sending and receiving data, allowing interoperability between parts from different vendors.

**Corelis BusPro-S Host Adapter**

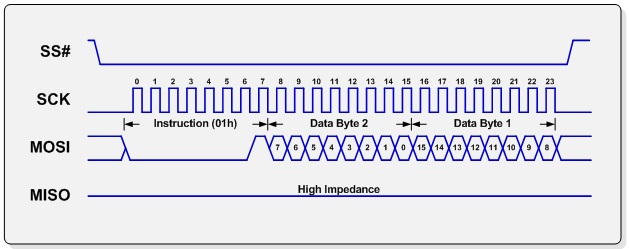
In this document, we’ll refer to Corelis SPI Exerciser Debugger code for creating SPI transactions. The Debugger module command script interface depicted in Figure 5 supports a simple command language for communication with SPI slave devices.

**  
 Figure 5. SPI Exerciser Debugger command script interface**

The BusPro-S includes automatic slave select (SS) signal handling—once a slave select signal has been selected with the “SS” command or defined in the application interface, the software will assert that slave select signal at the beginning of each command and de-assert the signal to complete the transaction. In this document, we will always explicitly specify the slave select state with “SSON” and “SSOFF” commands. In practical use, this is only required for grouping multiple commands into a single continuous transaction.

**Simple SPI Write Transaction**

Most SPI flash memories have a write status register command that writes one or two bytes of data, as shown in Figure 6. To write to the status register, the SPI host first enables the slave select line for the current device. The master then outputs the appropriate instruction followed by two data bytes that define the intended status register contents. Since the transaction does not need to return any data, the slave device keeps the MISO line in a high impedance state and the master masks any incoming data. Finally, slave select is de-asserted to complete the transaction.

**  
 Figure 6. Write command using a single-byte instruction and two-byte data word**

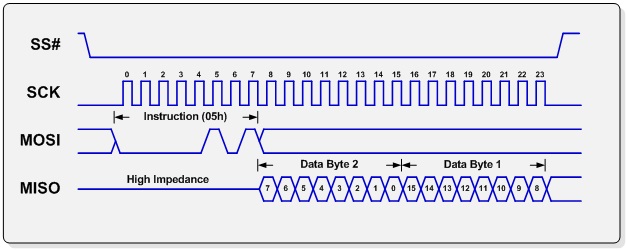
Using the Corelis SPI Exerciser command language, this transaction can be accomplished with the following code, where Data Byte 2 is “55” and Data Byte 1 is “AA”. For brevity we are using the short version of commands; the command script language supports both full commands such as “write, read” as well as abbreviated versions “wt, rd”. Figure 7 lists example code for a simple write transaction.

|  |
| --- |
| sson //Activate slave select  wt 01 55 AA // Write instruction 01h and data bytes 55h, AAh  ssoff // Deactivate slave select |

**Figure 7. Example code for a simple write transaction**

**Simple SPI Read Transaction**

A status register read transaction would be similar to the write transaction, but now takes advantage of data returned from the slave as shown in Figure 8. After sending the read status register instruction, the slave begins transmitting data on the MISO line at a rate of one byte per eight clock cycles. The host receives the bitstream and completes the transaction by de-asserting SS#.

**  
Figure 8. Read command using a single-byte instruction and two-byte data word**

This sequence is really a single-byte write followed by a two byte read; it can be considered a three-byte combined write/read command or a single byte write and two-byte read. We can create this transaction with the Corelis SPI Exerciser debugger command sequence shown in Figure 9.

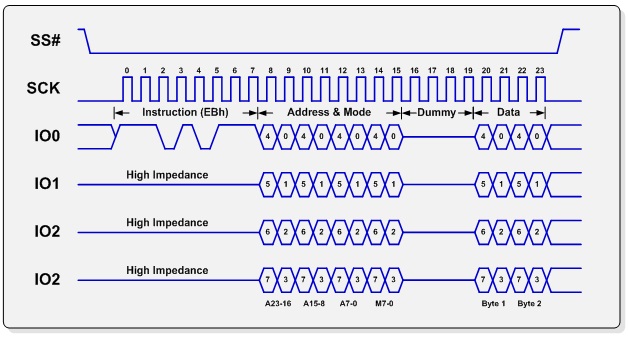
|  |
| --- |
| sson // Activate slave select  wt 05 // Write instruction 05h  rd 2 // Read two data bytes  ssoff // Deactivate slave select |

**Figure 9. Example code for a simple read transaction**

The data values from the read command will be displayed in the transaction log. The transaction log displays the data being transferred between the BusPro-S host and slave when performing read, write, or combined write/read commands.

**Quad IO SPI Transaction**

Quad IO is gaining popularity with flash memories for its increased performance. Instead of using a single output and single input interface, Quad IO utilizes 4 separate half-duplex data lines for both transmitting and receiving data for up to four times the performance of standard 4-wire SPI.   
  
Figure 10 shows an example read command for a Spansion S25FL016K serial NOR flash device. To read from the device, a fast read command (EBh) is first sent by the master on the first IO line while all others are tristated. Next, the host sends the address; since the interface now has 4 bidirectional data lines, it can utilize these to send a complete 24-bit address along with 8 mode bits in just 8 clock cycles. The address is then followed with 2 dummy bytes (4 clock cycles) to allow the device additional time to set up the initial address.

**  
 Figure 10. Quad mode fast read sequence for Spansion S25FL016K or equivalent**

After the address cycle and dummy bytes have been sent by the host, the component begins sending data bytes; each clock cycle consists of a data nibble spread across the 4 IO lines, for a total of two clock cycles per byte of data. Compare this to the 16 clock cycles required for our simple read transaction and it’s easy to see why quad mode is gaining popularity for high speed flash memory applications! To create this sequence in the SPI Exerciser command language, we would use the example code shown in Figure 11.

|  |
| --- |
| 4m // Start in 4-wire mode  sson // Activate slave select  wt EB // Write instruction EBh  qm // Switch to quad mode  wt AA AA AA 00 // Write 3-byte address and 8 read mode bits  wt 55 55 // Write 2 dummy bytes  rd 2 // Read two data bytes  ssoff // Deactivate slave select |

**Figure 11. Example code for a quad mode read transaction**

Note that we’re changing from 4-wire mode to quad mode in the middle of the transaction. In quad mode, the software automatically distributes the data bytes among the IO lines using the same bit pattern depicted in Figure 10 above.

**Conclusion**

The SPI interface bus is straightforward and versatile, enabling simple and fast communication with a variety of peripherals. A high speed multi-IO mode host adapter like the Corelis BusPro-S can be an invaluable tool in debugging as well as adding SPI communication capabilities to any test system. For more information about Corelis serial bus products, visit the Corelis website at[**http://www.corelis.com/products-bus-analyzers/**](http://www.corelis.com/products-bus-analyzers/).

Serial Peripheral Interface Bus

From Wikipedia, the free encyclopedia

[](https://en.wikipedia.org/wiki/File:SPI_single_slave.svg)

Single Master to Single Slave : basic SPI bus example.

The **Serial Peripheral Interface** (**SPI**) [bus](https://en.wikipedia.org/wiki/Bus_(computing)) is a [synchronous](https://en.wikipedia.org/wiki/Synchronous_circuit) [serial communication](https://en.wikipedia.org/wiki/Serial_communication) interface specification used for short distance communication, primarily in [embedded systems](https://en.wikipedia.org/wiki/Embedded_systems). The interface was developed by [Motorola](https://en.wikipedia.org/wiki/Motorola) in the late eighties and has become a [*de facto* standard](https://en.wikipedia.org/wiki/De_facto_standard). Typical applications include [Secure Digital](https://en.wikipedia.org/wiki/Secure_Digital) cards and [liquid crystal displays](https://en.wikipedia.org/wiki/Liquid_crystal_display).

SPI devices communicate in [full duplex](https://en.wikipedia.org/wiki/Full_duplex) mode using a [master-slave](https://en.wikipedia.org/wiki/Master-slave_(technology)) architecture with a single master. The master device originates the [frame](https://en.wikipedia.org/wiki/Frame_(networking)) for reading and writing. Multiple slave devices are supported through selection with individual [slave select](https://en.wikipedia.org/wiki/Slave_select) (SS) lines.

Sometimes SPI is called a *four-wire* serial bus, contrasting with [three-](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Three-wire_serial_buses), [two-](https://en.wikipedia.org/wiki/I%C2%B2C), and [one-wire](https://en.wikipedia.org/wiki/1-Wire) serial buses. The SPI may be accurately described as a synchronous serial interface,[[1]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-1) but it is different to the [Synchronous Serial Interface](https://en.wikipedia.org/wiki/Synchronous_Serial_Interface) (SSI) protocol, which is also a four-wire synchronous serial communication protocol, but employs [differential signaling](https://en.wikipedia.org/wiki/Differential_signaling) and provides only a single [simplex](https://en.wikipedia.org/wiki/Simplex_communication)

Interface[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=1" \o "Edit section: Interface)]

The SPI bus specifies four logic signals:

* SCLK : Serial Clock (output from master).
* MOSI : Master Output, Slave Input (output from master).
* MISO : Master Input, Slave Output (output from slave).
* SS : Slave Select ([active low](https://en.wikipedia.org/wiki/Logic_level), output from master).

Alternative naming conventions are also widely used, and SPI port pin names for particular IC products may differ from those depicted in these illustrations:

Serial Clock:

* SCLK : SCK, CLK.

Master Output {\displaystyle \rightarrow } Slave Input:

* MOSI : SIMO, SDI,[[2]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-forslavedevices-2) DI, DIN, SI, MTST.

Master Input {\displaystyle \leftarrow } Slave Output:

* MISO : SOMI, SDO,[[2]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-forslavedevices-2) DO, DOUT, SO, MRSR.

Slave Select:

* SS : nCS, CS, CSB, CSN, EN, nSS, STE, SYNC, SSQ.

The MOSI/MISO convention requires that, on devices using the alternate names, SDI on the master be connected to SDO on the slave, and vice versa. Chip select polarity is rarely active high, although some notations (such as SS or CS instead of nSS or nCS) suggest otherwise. Slave select is used instead of an addressing concept.

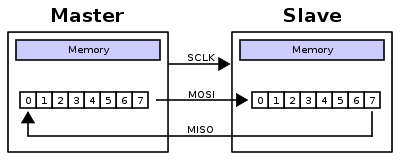
Operation[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=2" \o "Edit section: Operation)]

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](https://en.wikipedia.org/wiki/Logic_level) if the slave permits it. Some slaves require a falling [edge](https://en.wikipedia.org/wiki/Signal_edge) of the chip select signal to initiate an action. An example is the [Maxim](https://en.wikipedia.org/wiki/Maxim_Integrated_Products) MAX1242 [ADC](https://en.wikipedia.org/wiki/Analog-to-digital_converter), 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](https://en.wikipedia.org/wiki/Tri-state_output) so their MISO signal becomes [high impedance](https://en.wikipedia.org/wiki/High_impedance) (*logically disconnected*) when the device is not selected. Devices without tri-state outputs cannot share SPI bus segments with other devices; only one such slave could talk to the master.

**Data transmission**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=3" \o "Edit section: Data transmission)]

[](https://en.wikipedia.org/wiki/File:SPI_8-bit_circular_transfer.svg)

A typical hardware setup using two [shift registers](https://en.wikipedia.org/wiki/Shift_register) to form an inter-chip[circular buffer](https://en.wikipedia.org/wiki/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 analog-to-digital conversion, the master must wait for at least that period of time before issuing clock cycles.

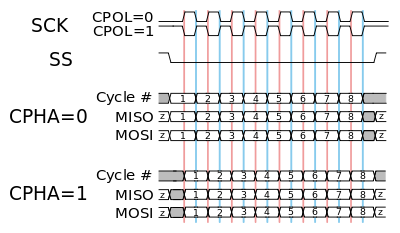
During each SPI clock cycle, a full duplex data transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

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, while shifting a new least-significant bit into the same register. At the same time, Data from the counterpart is shifted into the least-significant bit 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 8-bit words. However, other word sizes are also common, for example, 16-bit words for touchscreen controllers or audio codecs, such as the TSC2101 by[Texas Instruments](https://en.wikipedia.org/wiki/Texas_Instruments), or 12-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 must not drive MISO.

**Clock polarity and phase**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=4" \o "Edit section: Clock polarity and phase)]

[](https://en.wikipedia.org/wiki/File:SPI_timing_diagram2.svg)

A timing diagram showing clock polarity and phase. The red vertical line represents CPHA=0 and the blue vertical line represents CPHA=1

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Motorola SPI Block Guide[[3]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus" \l "cite_note-3) names these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The [timing diagram](https://en.wikipedia.org/wiki/Digital_timing_diagram) is shown to the right. The timing is further described below and applies to both the master and the slave device.

* At CPOL=0 the base value of the clock is zero, i.e. the idle state is 0 and active state is 1.
  + For CPHA=0, data are captured on the clock's rising edge (low→high transition) and data is output on a falling edge (high→low clock transition).
  + For CPHA=1, data are captured on the clock's falling edge and data is output on a rising edge.
* At CPOL=1 the base value of the clock is one (inversion of CPOL=0), i.e. the idle state is 1 and active state is 0.
  + For CPHA=0, data are captured on clock's falling edge and data is output on a rising edge.
  + For CPHA=1, data are captured on clock's rising edge and data is output on a falling edge.

That is, CPHA=0 means sampling on the first clock edge, while CPHA=1 means sampling on the second clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle.

In other words, CPHA=0 means transmitting data on the active to idle state and CPHA=1 means that data is transmitted on the idle to active state edge. Note that if transmission happens on a particular edge, then capturing will happen on the opposite edge(i.e. if transmission happens on falling, then reception happens on rising and vice versa). The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle.

This adds more flexibility to the communication channel between the master and slave.

**Mode numbers**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=5" \o "Edit section: Mode numbers)]

The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

For "Microchip PIC" / "ARM-based" microcontrollers (note that NCPHA is the inversion of CPHA):

|  |  |  |  |
| --- | --- | --- | --- |
| **SPI Mode** | **Clock Polarity (CPOL/CKP)** | **Clock Phase (CPHA)** | **Clock Edge (CKE/NCPHA)** |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 |

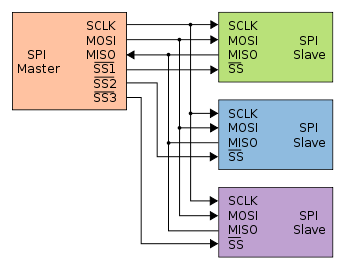
For PIC32MX : SPI mode configure CKP,CKE and SMP bits.Set SMP bit,and CKP,CKE two bits configured as above table.

For other microcontrollers:

|  |  |  |
| --- | --- | --- |
| **Mode** | **CPOL** | **CPHA** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

Another commonly used notation represents the mode as a (CPOL, CPHA) tuple; e.g., the value '(0, 1)' would indicate CPOL=0 and CPHA=1.

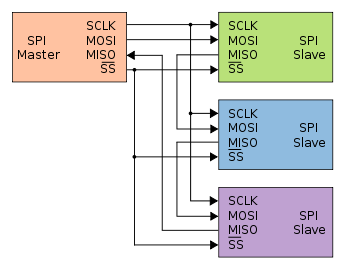
**Independent slave configuration**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=6" \o "Edit section: Independent slave configuration)]

[](https://en.wikipedia.org/wiki/File:SPI_three_slaves.svg)

Typical SPI bus: master and three independent slaves

In the independent slave configuration, there is an independent chip select line for each slave. A pull-up resistor between power source and chip select line is highly recommended for each independent device to reduce cross-talk between devices.[[4]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-dorkbotSPI-4) This is the way SPI is normally used. Since the MISO pins of the slaves are connected together, they are required to be tri-state pins (high, low or high-impedance).

**Daisy chain configuration**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=7" \o "Edit section: Daisy chain configuration)]

[](https://en.wikipedia.org/wiki/File:SPI_three_slaves_daisy_chained.svg)

Daisy-chained SPI bus: master and cooperative slaves

Some products that implement SPI may be connected in a [daisy chain](https://en.wikipedia.org/wiki/Daisy_chain_(electrical_engineering)) 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](https://en.wikipedia.org/wiki/Shift_register); daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.[[5]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-5)

Applications that require a daisy chain configuration include [SGPIO](https://en.wikipedia.org/wiki/SGPIO) and [JTAG](https://en.wikipedia.org/wiki/JTAG).

**Valid communications**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=8" \o "Edit section: Valid communications)]

Some slave devices are designed to ignore any SPI communications in which the number of clock pulses is greater than specified. Others do not care, ignoring extra inputs and continuing to shift the same output bit. It is common for different devices to use SPI communications with different lengths, as, for example, when SPI is used to access the [scan chain](https://en.wikipedia.org/wiki/Scan_chain) of a digital IC by issuing a command word of one size (perhaps 32 bits) and then getting a response of a different size (perhaps 153 bits, one for each pin in that scan chain).

**Interrupts**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=9" \o "Edit section: Interrupts)]

SPI devices sometimes use another signal line to send an interrupt signal to a host CPU. Examples include pen-down interrupts from touchscreen sensors, thermal limit alerts from temperature sensors, alarms issued by real time clock chips, [SDIO](https://en.wikipedia.org/wiki/Secure_Digital#SDIO),[[6]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-3wireSDI-6) and headset jack insertions from the sound codec in a cell phone. Interrupts are not covered by the SPI standard; their usage is neither forbidden nor specified by the standard.

**Example of bit-banging the master protocol**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=10" \o "Edit section: Example of bit-banging the master protocol)]

Below is an example of [bit-banging](https://en.wikipedia.org/wiki/Bit-banging) the SPI protocol as an SPI master with CPOL=0, CPHA=0, and eight bits per transfer. The example is written in the C programming language. Because this is CPOL=0 the clock must be pulled low before the chip select is activated. The chip select line must be activated, which normally means being toggled low, for the peripheral before the start of the transfer, and then deactivated afterwards. Most peripherals allow or require several transfers while the select line is low; this routine might be called several times before deselecting the chip.

*/\**

*\* Simultaneously transmit and receive a byte on the SPI.*

*\**

*\* Polarity and phase are assumed to be both 0, i.e.:*

*\* - input data is captured on rising edge of SCLK.*

*\* - output data is propagated on falling edge of SCLK.*

*\**

*\* Returns the received byte.*

*\*/*

uint8\_t SPI\_transfer\_byte(uint8\_t byte\_out)

{

uint8\_t byte\_in = 0;

uint8\_t bit;

**for** (bit = 0x80; bit; bit >>= 1) {

*/\* Shift-out a bit to the MOSI line \*/*

write\_MOSI((byte\_out & bit) ? HIGH : LOW);

*/\* Delay for at least the peer's setup time \*/*

delay(SPI\_SCLK\_LOW\_TIME);

*/\* Pull the clock line high \*/*

write\_SCLK(HIGH);

*/\* Shift-in a bit from the MISO line \*/*

**if** (read\_MISO() == HIGH)

byte\_in |= bit;

*/\* Delay for at least the peer's hold time \*/*

delay(SPI\_SCLK\_HIGH\_TIME);

*/\* Pull the clock line low \*/*

write\_SCLK(LOW);

}

**return** byte\_in;

}

Pros and cons[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=11" \o "Edit section: Pros and cons)]

**Advantages**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=12" \o "Edit section: Advantages)]

* Full duplex communication in the default version of this protocol.
* [Push-pull drivers](https://en.wikipedia.org/wiki/Push-pull_output) (as opposed to open drain) provide good signal integrity and high speed
* Higher [throughput](https://en.wikipedia.org/wiki/Throughput) than [I²C](https://en.wikipedia.org/wiki/I%C2%B2C) or [SMBus](https://en.wikipedia.org/wiki/System_Management_Bus)
* Complete protocol flexibility for the bits transferred
  + Not limited to 8-bit words
  + Arbitrary choice of message size, content, and purpose
* Extremely simple hardware interfacing
  + Typically lower power requirements than [I²C](https://en.wikipedia.org/wiki/I%C2%B2C) or SMBus due to less circuitry (including pull up resistors)
  + No arbitration or associated failure modes
  + Slaves use the master's clock, and do not need precision oscillators
  + Slaves do not need a unique [address](https://en.wikipedia.org/wiki/Address_space) — unlike [I²C](https://en.wikipedia.org/wiki/I%C2%B2C) or [GPIB](https://en.wikipedia.org/wiki/GPIB) or [SCSI](https://en.wikipedia.org/wiki/SCSI)
  + Transceivers are not needed
* Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
* At most one unique bus signal per device (chip select); all others are shared
* Signals are unidirectional allowing for easy [Galvanic isolation](https://en.wikipedia.org/wiki/Galvanic_isolation)
* Not limited to any maximum clock speed, enabling potentially high speed
* Simple software implementation

**Disadvantages**[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=13" \o "Edit section: Disadvantages)]

* Requires more pins on IC packages than [I²C](https://en.wikipedia.org/wiki/I%C2%B2C), even in the [*three-wire*](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Three-wire_serial_buses) variant
* No in-band addressing; out-of-band chip select signals are required on shared buses
* No hardware [flow control](https://en.wikipedia.org/wiki/Flow_control_(data)) by the slave (but the master can delay the next clock edge to slow the transfer rate)
* No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
* Typically supports only one master device (depends on device's hardware implementation)
* No error-checking protocol is defined
* Without a formal standard, validating conformance is not possible
* Only handles short distances compared to [RS-232](https://en.wikipedia.org/wiki/RS-232), [RS-485](https://en.wikipedia.org/wiki/RS-485), or [CAN-bus](https://en.wikipedia.org/wiki/CAN-bus). (its distance can be extended with use of transceivers like [RS-422](https://en.wikipedia.org/wiki/RS-422))
* Many existing variations, making it difficult to find development tools like host adapters that support those variations
* SPI does not support [hot swapping](https://en.wikipedia.org/wiki/Hot_swapping) (dynamically adding nodes).
* Interrupts must either be implemented with out-of-band signals or be faked by using periodic polling similarly to USB 1.1 and 2.0
* Some variants like [Multi I/O SPI](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Multi_I.2FO_SPI) and [three-wire serial buses](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Three-wire_serial_buses) defined below are half-duplex.

Applications[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=14" \o "Edit section: 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](https://en.wikipedia.org/wiki/System-on-a-chip) processors, both with higher end 32-bit processors such as those using [ARM](https://en.wikipedia.org/wiki/ARM_architecture),[MIPS](https://en.wikipedia.org/wiki/MIPS_architecture), or [PowerPC](https://en.wikipedia.org/wiki/PowerPC) and with other microcontrollers such as the [AVR](https://en.wikipedia.org/wiki/Atmel_AVR), [PIC](https://en.wikipedia.org/wiki/PIC_microcontroller), and [MSP430](https://en.wikipedia.org/wiki/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 an SPI interface.[[7]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-7)

Chip or [FPGA](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Digital_audio), [digital signal processing](https://en.wikipedia.org/wiki/Digital_signal_processing), or[telecommunications channels](https://en.wikipedia.org/wiki/Channel_(communications)), 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](https://en.wikipedia.org/wiki/Analog-to-digital_converter), touchscreens, video game controllers
* Control devices: [audio codecs](https://en.wikipedia.org/wiki/Audio_codec), digital potentiometers, [DAC](https://en.wikipedia.org/wiki/Digital-to-analog_converter)
* Camera lenses: [Canon EF lens mount](https://en.wikipedia.org/wiki/Canon_EF_lens_mount)
* Communications: [Ethernet](https://en.wikipedia.org/wiki/Ethernet), [USB](https://en.wikipedia.org/wiki/USB), [USART](https://en.wikipedia.org/wiki/USART), [CAN](https://en.wikipedia.org/wiki/CAN_bus), [IEEE 802.15.4](https://en.wikipedia.org/wiki/IEEE_802.15.4), [IEEE 802.11](https://en.wikipedia.org/wiki/IEEE_802.11), handheld video games
* Memory: [flash](https://en.wikipedia.org/wiki/Flash_memory#Serial_flash) and [EEPROM](https://en.wikipedia.org/wiki/EEPROM#Serial_bus_devices)
* Real-time clocks
* [LCD](https://en.wikipedia.org/wiki/LCD), sometimes even for managing image data
* Any [MMC](https://en.wikipedia.org/wiki/MultiMediaCard) or [SD](https://en.wikipedia.org/wiki/Secure_Digital) card (including [SDIO](https://en.wikipedia.org/wiki/Secure_Digital#SDIO) variant[[6]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-3wireSDI-6))

For high performance systems, [FPGAs](https://en.wikipedia.org/wiki/FPGA) 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.

Although there are some similarities between the SPI bus and the [JTAG](https://en.wikipedia.org/wiki/JTAG) (IEEE 1149.1-2013) protocol, they are not interchangeable. The SPI bus is intended for high speed, on board initialization of device peripherals, while the JTAG protocol is intended to provide reliable test access to the I/O pins from an off board controller with less precise signal delay and skew parameters. While not strictly a level sensitive interface, the JTAG protocol supports the recovery of both setup and hold violations between JTAG devices by reducing the clock rate or changing the clock's duty cycles. Consequently, the JTAG interface is not intended to support extremely high data rates.[[8]](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#cite_note-8)

[SGPIO](https://en.wikipedia.org/wiki/SGPIO) is essentially another (incompatible) application stack for SPI designed for particular backplane management activities.[*[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*] SGPIO uses 3-bit messages.

Standards[[edit](https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&action=edit&section=15)]

The SPI bus is a [*de facto* standard](https://en.wikipedia.org/wiki/De_facto_standard). However, the lack of a formal standard is reflected in a wide variety of protocol options. Different word sizes are common. Every device defines its own protocol, including whether it supports commands at all. Some devices are transmit-only; others are receive-only. Chip selects are sometimes active-high rather than active-low. Some protocols send the least significant bit first.

Some devices even have minor variances from the CPOL/CPHA modes described above. Sending data from slave to master may use the opposite clock edge as master to slave. Devices often require extra clock idle time before the first clock or after the last one, or between a command and its response. Some devices have two clocks, one to read data, and another to transmit it into the device. Many of the read clocks run from the chip select line.

Some devices require an additional flow control signal from slave to master, indicating when data are ready. This leads to a 5-wire protocol instead of the usual 4. Such a *ready* or *enable* signal is often active-low, and needs to be enabled at key points such as after commands or between words. Without such a signal, data transfer rates may need to be slowed down significantly, or protocols may need to have dummy bytes inserted, to accommodate the worst case for the slave response time. Examples include initiating an ADC conversion, addressing the right page of flash memory, and processing enough of a command that device firmware can load the first word of the response. (Many SPI masters do not support that signal directly, and instead rely on fixed delays.)

Many SPI chips only support messages that are multiples of 8 bits. Such chips can not interoperate with the [JTAG](https://en.wikipedia.org/wiki/JTAG) or [SGPIO](https://en.wikipedia.org/wiki/SGPIO) protocols, or any other protocol that requires messages that are not multiples of 8 bits.

There are also hardware-level differences. Some chips combine MOSI and MISO into a single data line (SI/SO); this is sometimes called 'three-wire' signaling (in contrast to normal 'four-wire' SPI). Another variation of SPI removes the chip select line, managing protocol state machine entry/exit using other methods. Anyone needing an external connector for SPI defines their own: [UEXT](https://en.wikipedia.org/wiki/UEXT), [JTAG connector](https://en.wikipedia.org/wiki/JTAG_connector), [Secure Digital](https://en.wikipedia.org/wiki/Secure_Digital) card socket, etc. Signal levels depend entirely on the chips involved.

[SafeSPI](http://safespi.org/) is an industry standard for SPI in automotive applications. Its main focus is the transmission of sensor data between different devices.