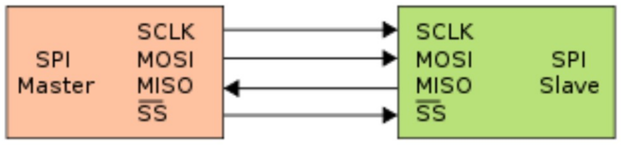
en.wikipedia.org

Serial Peripheral Interface - Wikipedia

31–40 minuuttia



Single Main to single Secondary: basic SPI bus example

The **Serial Peripheral Interface** (**SPI**) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid-1980s and has become a *de facto* standard. Typical applications include Secure Digital cards and liquid crystal displays.

SPI devices communicate in full duplex mode using a master-slave architecture (alternate terminology being main and secondary) 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,[1] 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.

Interface[edit]

The SPI bus specifies four logic signals:

- SCLK: Serial Clock (output from master)
- MOSI: Master Output Slave Input, or Master Out Slave In (data output from master)
- MISO: Master Input Slave Output, or Master In Slave Out (data output from slave)
- SS: Slave Select (often active low, output from master)

While the above pin names are the most popular, in the past alternative pin-naming conventions were sometimes used, and so SPI port pin-names for older IC products may differ from those depicted in these illustrations:

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: $\overline{\text{SS}}$, SSEL, CS, $\overline{\text{CS}}$, 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.

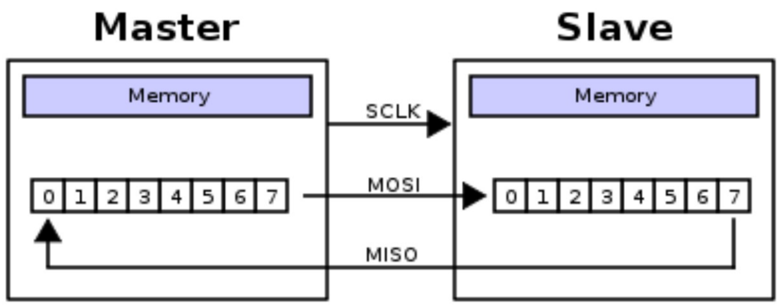
Operation[edit]

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.

Data transmission[edit]



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.

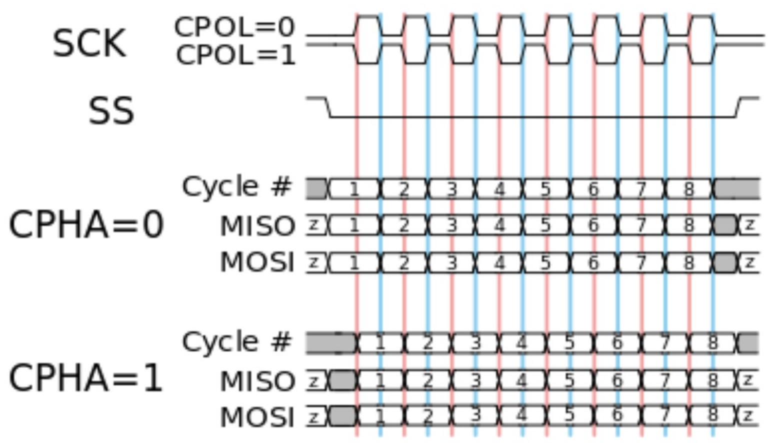
During each SPI clock cycle, a full-duplex data transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

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.

Clock polarity and phase[edit]



A timing diagram showing clock polarity and phase. Red lines denote clock leading edges, and blue lines, trailing edges.

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[2] names these two options as CPOL and CPHA (for clock "pol"arity and "pha"se) respectively, a convention most vendors have also adopted.

The timing diagram is shown to the right. The timing is further described below and applies to both the master and the slave device.

- CPOL determines the polarity of the clock. The polarities can be converted with a simple inverter.
- CPOL=0 is a clock which idles at 0, and each cycle consists of a pulse of 1. That is, the leading edge is a rising edge, and the trailing edge is a falling edge.
- CPOL=1 is a clock which idles at 1, and each cycle consists of a pulse of 0. That is, the leading edge is a falling edge, and the trailing edge is a rising edge.
- CPHA determines the timing (i.e. phase) of the data bits relative to the clock pulses. Conversion between these two forms is non-trivial.
- For CPHA=0, the "out" side changes the data on the trailing edge of the preceding clock cycle, while the "in" side captures the data on (or shortly after) the leading edge of the clock cycle. The out side holds the data valid until the trailing edge of the current clock cycle. For the first cycle, the first bit must be on the MOSI line before the leading clock edge.
- An alternative way of considering it is to say that a CPHA=0 cycle consists of a half cycle with the clock idle, followed by a half cycle with the clock asserted.
- For CPHA=1, the "out" side changes the data on the leading edge of the current clock cycle, while the "in" side captures the data on

(or shortly after) the trailing edge of the clock cycle. The out side holds the data valid until the leading edge of the following clock cycle. For the last cycle, the slave holds the MISO line valid until slave select is deasserted.

- An alternative way of considering it is to say that a CPHA=1 cycle consists of a half cycle with the clock asserted, followed by a half cycle with the clock idle.

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]

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

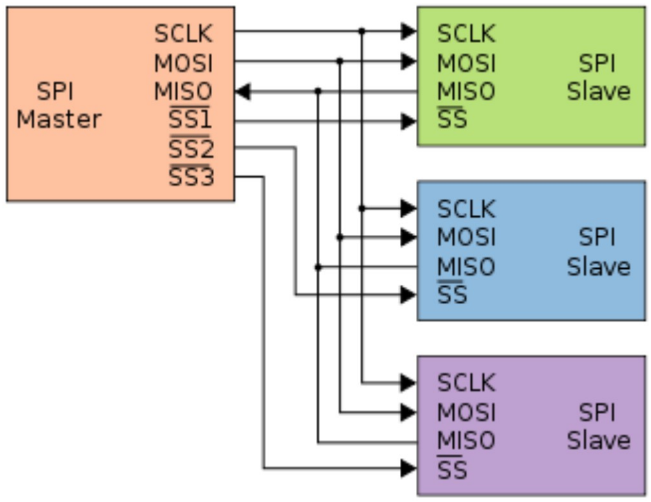
For PIC32MX: SPI mode configure CKP, CKE and SMP bits. Set SMP bit and CKP, CKE two bits configured as above table.

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]



Typical SPI bus: master and three independent slaves

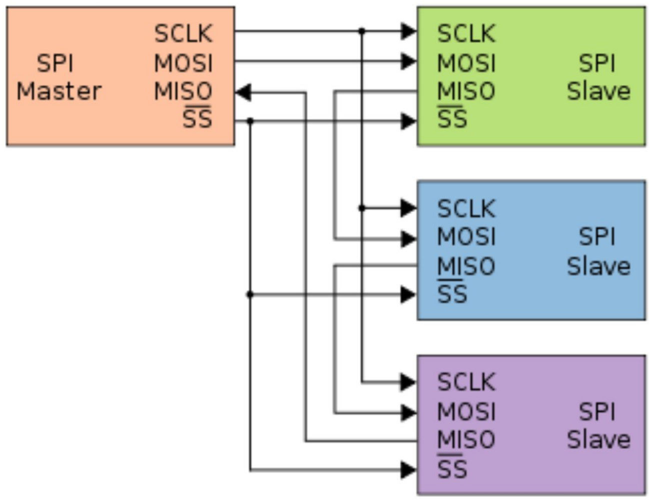
In the independent slave configuration, there is an independent chip select line for each slave. This is the way SPI is normally used.

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.

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.)

Daisy chain configuration[edit]



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.[4]

Other applications that can potentially interoperate with SPI that require a daisy chain configuration include SGPIO, JTAG,[5] and Two Wire Interface.

Valid communications[edit]

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 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]

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,[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. In other words, interrupts are outside the scope of the SPI standard and are optionally implemented independently from it.

Example of bit-banging the master protocol[edit]

Below is an example of 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 afterward. 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]

Advantages[edit]

- Full duplex communication in the default version of this protocol
- Push-pull drivers (as opposed to open drain) provide good signal integrity and high speed
- Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speed
- 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 or SMBus due to less circuitry (including pull up resistors)
- No arbitration or associated failure modes - unlike CAN-bus
- Slaves use the master's clock and do not need precision oscillators
- Slaves do not need a unique address – unlike I²C or GPIB or SCSI
- Transceivers are not needed - unlike CAN-bus
- 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

- Simple software implementation

Disadvantages[edit]

- Requires more pins on IC packages than I²C, even in the *three-wire* variant
- No in-band addressing; out-of-band chip select signals are required on shared buses
- No hardware flow control 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, RS-485, or CAN-bus. (Its distance can be extended with the use of transceivers like RS-422.)
- Opto-isolators in the signal path limit the clock speed for MISO transfer because of the added delays between clock and data
- Many existing variations, making it difficult to find development tools like host adapters that support those variations
- SPI does not support 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 dual SPI, quad SPI, and three-wire serial buses

defined below are half-duplex.

Applications[edit]

SPI Serial Memory by Atmel

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.[7]

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.

Although there are some similarities between the SPI bus and the 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]

SGPIO is essentially another (incompatible) application stack for SPI designed for particular backplane management activities.[*citation needed*] SGPIO uses 3-bit messages.

Standards[edit]

The SPI bus is a *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 is 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 or 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, JTAG connector, Secure Digital card socket, etc. Signal levels depend entirely on the chips involved.

SafeSPI[9] is an industry standard for SPI in automotive applications. Its main focus is the transmission of sensor data between different devices.

Development tools[edit]

When developing or troubleshooting systems using SPI, visibility at the level of hardware signals can be important.

Host adapters[edit]

There are a number of USB hardware solutions to provide computers, running Linux, Mac, or Windows, SPI master or slave capabilities. Many of them also provide scripting or programming capabilities (Visual Basic, C/C++, VHDL, etc.).

An SPI host adapter lets the user play the role of a master on an SPI bus directly from a PC. They are used for embedded systems, chips (FPGA, ASIC, and SoC) and peripheral testing, programming and debugging.

The key parameters of SPI are: the maximum supported frequency for the serial interface, command-to-command latency and the maximum length for SPI commands. It is possible to find SPI adapters on the market today that support up to 100 MHz serial interfaces, with virtually unlimited access length.

SPI protocol being a de facto standard, some SPI host adapters also have the ability of supporting other protocols beyond the traditional 4-wire SPI (for example, support of quad-SPI protocol or other custom serial protocol that derive from SPI[10]).

Protocol analyzers[edit]

SPI protocol analyzers are tools which sample an SPI bus and decode the electrical signals to provide a higher-level view of the data being transmitted on a specific bus.

Oscilloscopes[edit]

Most oscilloscope vendors offer oscilloscope-based triggering and protocol decoding for SPI. Most support 2-, 3-, and 4-wire SPI. The triggering and decoding capability is typically offered as an optional extra. SPI signals can be accessed via analog oscilloscope channels or with digital MSO channels.[11]

Logic analyzers[edit]

When developing or troubleshooting the SPI bus, examination of hardware signals can be very important. Logic analyzers are tools which collect, analyze, decode, and store signals so people can view the high-speed waveforms at their leisure. Logic analyzers display time-stamps of each signal level change, which can help find protocol problems. Most logic analyzers have the capability to decode bus signals into high-level protocol data and show ASCII data.

[edit]

Intelligent SPI controllers[edit]

A **Queued Serial Peripheral Interface** (**QSPI**) is a type of SPI controller that uses a data queue to transfer data across the SPI bus.[12] It has a wrap-around mode allowing continuous transfers to and from the queue with only intermittent attention from the CPU. Consequently, the peripherals appear to the CPU as memory-mapped parallel devices. This feature is useful in applications such as control of an A/D converter. Other programmable features in QSPI are chip selects and transfer length/delay.

SPI controllers from different vendors support different feature sets; such DMA queues are not uncommon, although they may be associated with separate DMA engines rather than the SPI controller itself, such as used by **Multichannel Buffered Serial Port** (**MCBSP**).[13] Most SPI master controllers integrate support for up to four chip selects,[14] although some require chip selects to be managed separately through GPIO lines.

Microwire[edit]

Fairchild Serial EEPROM with Microwire bus

Microwire,[15] often spelled **μWire**, is essentially a predecessor of SPI and a trademark of National Semiconductor. It's a strict subset of SPI: half-duplex, and using SPI mode 0. Microwire chips tend to need slower clock rates than newer SPI versions; perhaps 2 MHz vs. 20 MHz. Some Microwire chips also support a three-wire mode.

Microwire/Plus[edit]

Microwire/Plus[16] is an enhancement of Microwire and features full-duplex communication and support for SPI modes 0 and 1. There was no specified improvement in serial clock speed.

Three-wire serial buses[edit]

As mentioned, one variant of SPI uses single bidirectional data line (slave out/slave in, called SISO or master out/master in, called MOMI) instead of two unidirectional ones (MOSI and MISO). This variant is restricted to a half duplex mode. It tends to be used for lower performance parts, such as small EEPROMs used only during system startup and certain sensors, and Microwire. Few SPI master controllers support this mode; although it can often be easily bit-banged in software.

Dual SPI[edit]

For instances where the full-duplex nature of SPI is not used, an extension uses both data pins in a half-duplex configuration to send two bits per clock cycle. Typically a command byte is sent requesting a response in dual mode, after which the MOSI line becomes SIO0 (serial I/O 0) and carries even bits, while the MISO line becomes SIO1 and carries odd bits. Data is still transmitted msbit-first, but SIO1 carries bits 7, 5, 3 and 1 of each byte, while SIO0 carries bits 6, 4, 2 and 0.

This is particularly popular among SPI ROMs, which have to send a large amount of data, and comes in two variants:[17][18]

- Dual read commands accept the send and address from the master in single mode, and return the data in dual mode.
- Dual I/O commands send the command in single mode, then send the address and return data in dual mode.

Quad SPI[edit]

While dual SPI re-uses the existing serial I/O lines, quad SPI adds two more I/O lines (SIO2 and SIO3) and sends 4 data bits per clock cycle. Again, it is requested by special commands, which enable quad mode after the command itself is sent in single mode.[17][18]

SQI Type 1: Commands sent on single line but addresses and data sent on four lines

SQI Type 2: Commands and addresses sent on a single line but data sent/received on four lines

QPI/SQI[edit]

Further extending quad SPI, some devices support a "quad everything" mode where *all* communication takes place over 4 data lines, including commands.[19] This is variously called "QPI"[18] (not to be confused with Intel QuickPath Interconnect) or "serial quad I/O" (SQI)[20]

This requires programming a configuration bit in the device and requires care after reset to establish communication.

Double data rate[edit]

In addition to using multiple lines for I/O, some devices increase the transfer rate by using double data rate transmission.[21][22]

Intel Enhanced Serial Peripheral Interface Bus[edit]

Intel has developed a successor to its Low Pin Count (LPC) bus that it calls the Enhanced Serial Peripheral Interface Bus, or eSPI for short. Intel aims to allow the reduction in the number of pins required on motherboards compared to systems using LPC, have more available throughput than LPC, reduce the working voltage to 1.8 volts to facilitate smaller chip manufacturing processes, allow eSPI peripherals to share SPI flash devices with the host (the LPC bus did not allow firmware hubs to be used by LPC peripherals), tunnel previous out-of-band pins through the eSPI bus, and allow system designers to trade off cost and performance.[23][24]

The eSPI bus can either be shared with SPI devices to save pins or be separate from the SPI bus to allow more performance, especially when eSPI devices need to use SPI flash devices.[23]

This standard defines an Alert# signal that is used by an eSPI slave to request service from the master. In a performance-oriented design or a design with only one eSPI slave, each eSPI slave will have its Alert# pin connected to an Alert# pin on the eSPI master that is dedicated to each slave, allowing the eSPI master to grant low-latency service because the eSPI master will know which eSPI slave needs service and will not need to poll all of the slaves to determine which device needs service. In a budget design with more than one eSPI slave, all of the Alert# pins of the slaves are connected to one Alert# pin on the eSPI master in a wired-OR connection, which will require the master to poll all the slaves to determine which ones need service when the Alert# signal is pulled low by one or more peripherals that need service. Only after all of the devices are serviced will the Alert# signal be pulled high due to none of the eSPI slaves needing service and therefore pulling the Alert# signal low.[23]

This standard allows designers to use 1-bit, 2-bit, or 4-bit communications at speeds from 20 to 66 MHz to further allow designers to trade off performance and cost.[23]

All communications that were out-of-band of the LPC bus like general-purpose input/output (GPIO) and System Management Bus (SMBus) are tunneled through the eSPI bus via virtual wire cycles and out-of-band message cycles respectively in order to remove those pins from motherboard designs using eSPI.[23]

This standard supports standard memory cycles with lengths of 1 byte to 4 kilobytes of data, short memory cycles with lengths of 1, 2, or 4 bytes that have much less overhead compared to standard memory cycles, and I/O cycles with lengths of 1, 2, or 4 bytes of data which are low overhead as well. This significantly reduces overhead compared to the LPC bus, where all cycles except for the 128-byte firmware hub read cycle spends more than one-half of all of the bus's throughput and time in overhead. The standard memory cycle allows a length of anywhere from 1 byte to 4 kilobytes in order to allow its larger overhead to be amortised over a large transaction. eSPI slaves are allowed to initiate bus master versions of all of the memory cycles. Bus master I/O cycles, which were introduced by the LPC bus specification, and ISA-style DMA including the 32-bit variant introduced by the LPC bus specification, are not present in eSPI. Therefore, bus master memory cycles are the only allowed DMA in this standard.[23]

eSPI slaves are allowed to use the eSPI master as a proxy to perform flash operations on a standard SPI flash memory slave on behalf of the requesting eSPI slave.[23]

64-bit memory addressing is also added, but is only permitted when there is no equivalent 32-bit address.[23]

The Intel Z170 chipset can be configured to implement either this bus or a variant of the LPC bus that is missing its ISA-style DMA capability and is underclocked to 24 MHz instead of the standard 33 MHz.[25]

See also[edit]

- List of network buses

References[edit]

1. ^ *"What is Serial Synchronous Interface (SSI)?". Retrieved 2015-01-28.*

2. ^ SPI Block Guide v3.06; Motorola/Freescale/NXP; 2003.
3. ^ Jump up to: *a* *b* Better SPI Bus Design in 3 Steps
4. ^ Maxim-IC application note 3947: "Daisy-Chaining SPI Devices"
5. ^ *Interfaces, pp. 80, 84*
6. ^ Jump up to: *a* *b* Not to be confused with the SDIO(Serial Data I/O) line of the half-duplex implementation of the SPI bus, sometimes also called "3-wire SPI-bus". Here e.g. MOSI (via a resistor) and MISO (no resistor) of a master is connected to the SDIO line of a slave.
7. ^ *"AVR910 - In-system programming" (PDF). Archived from the original (PDF) on 2011-03-02.*
8. ^ IEEE 1149.1-2013
9. ^ SafeSPI.org
10. ^ SPI Storm – Serial Protocol Host Adapter with support of custom serial protocols, Byte Paradigm.
11. ^ *"N5391B I²C and SPI Protocol Triggering and Decode for Infiniium scopes".*
12. ^ Queued Serial Module Reference Manual, Freescale Semiconductor (now NXP)
13. ^ Such as with the MultiChannel Serial Port Interface, or McSPI, used in Texas Instruments OMAP chips.
14. ^ Such as the SPI controller on Atmel AT91 chips like the at91sam9G20, which is much simpler than TI's McSPI.
15. ^ MICROWIRE Serial Interface National Semiconductor Application Note AN-452

16. ^ MICROWIRE/PLUS Serial Interface for COP800 Family National Semiconductor Application Note AN-579

17. ^ Jump up to: a b "W25Q16JV 3V 16M-bit serial flash memory with Dual/Quad SPI" (PDF) (data sheet). Revision D. Winbond. 12 August 2016. Retrieved 2017-02-10.

18. ^ Jump up to: a b c "D25LQ64 1.8V Uniform Sector Dual and Quad SPI Flash" (PDF) (data sheet). version 0.1. GigaDevice. 11 February 2011. Archived from the original (PDF) on 12 February 2017. Retrieved 2017-02-10.

19. ^ "QuadSPI flash: Quad SPI mode vs. QPI mode". NXP community forums. December 2014. Retrieved 2016-02-10.

20. ^ "SST26VF032B / SST26VF032BA 2.5V/3.0V 32 Mbit Serial Quad I/O (SQI) Flash Memory" (PDF) (Data sheet). version E. Microchip, Inc. 2017. Retrieved 2017-02-10.

21. ^ Patterson, David (May 2012). "Quad Serial Peripheral Interface (QuadSPI) Module Updates" (PDF) (Application note). Freescale Semiconductor. Retrieved September 21, 2016.

22. ^ Pell, Rich (13 October 2011). "Improving performance using SPI-DDR NOR flash memory". EDN.

23. ^ Jump up to: a b c d e f g h Enhanced Serial Peripheral Interface (eSPI) Interface Base Specification (for Client and Server Platforms) (PDF) (Report). Revision 1.0. Intel. January 2016. Document number 327432-004. Retrieved 2017-02-05.

24. ^ Enhanced Serial Peripheral Interface (eSPI) Interface Specification (for Client Platforms) (PDF) (Report). Revision 0.6. Intel. May 2012. Document Number 327432-001EN. Retrieved 2017-02-05.

25. ^ *"Intel® 100 Series Chipset Family PCH Datasheet, Vol. 1" (PDF). Retrieved April 15, 2015.*

External links[edit]

- Intel eSPI (Enhanced Serial Peripheral Interface)
- Introduction to SPI and I2C protocols
- Serial buses information page
- SPI Introduction
- SPI Tutorial