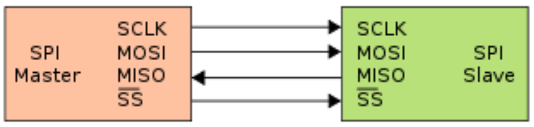
Serial Peripheral Interface Bus

From Wikipedia, the free encyclopedia

The **Serial Peripheral Interface Bus** or **SPI** (pronounced as either *ess-pee-eye* or *spy*) bus is a synchronous serial data link standard, named by Motorola, that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a *four-wire* serial bus, contrasting with three-, two-, and one-wire serial buses. SPI is often referred to as SSI (Synchronous Serial Interface).



SPI bus: single master and single slave

Contents

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, output from master).

Alternative naming conventions are also widely used:

- SCLK: SCK, CLK: serial clock (output from master)
- MOSI: SIMO, SDI, DI, DIN, SI, MTSR: serial data in; data in, serial in, master transmit slave receive
- MISO: SOMI, SDO, DO, DOUT, SO, MRST: serial data out; data out, serial out, master receive slave transmit
- SS: nCS, CS, CSB, CSN, nSS, STE: chip select, slave transmit enable (active low, output from master)

The SDI/SDO (DI/DO, SI/SO) convention requires that SDO on the master be connected to SDI 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.

SPI port pin names for particular IC products may differ from those depicted in these illustrations.

Operation

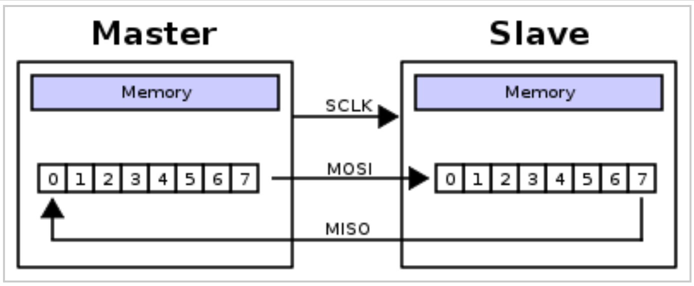
The SPI bus can operate with a single master device and with one or more slave devices.

If a single slave device is used, the SS pin *may* be fixed to logic low if the slave permits it. Some slaves require the falling edge (high→low transition) of the chip select to initiate an action such as the Maxim MAX1242 ADC, which starts conversion on said 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 (*logically disconnected*) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

Data transmission

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 MHz.



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

The master then transmits the logic 0 for the desired chip over chip select line. A logic 0 is transmitted because the chip select line is active low, meaning its *off* state is a logic 1; *on* is asserted with a logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line; the slave reads it from that same line
- the slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be *meaningful* but they do happen.

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 ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data[1] and the process repeats.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. However, other word sizes are also common, such as 16-bit words for touchscreen controllers or audio codecs, like the TSC2101 from Texas Instruments; or 12-bit words for many digital-to-analog or analog-to-digital converters.

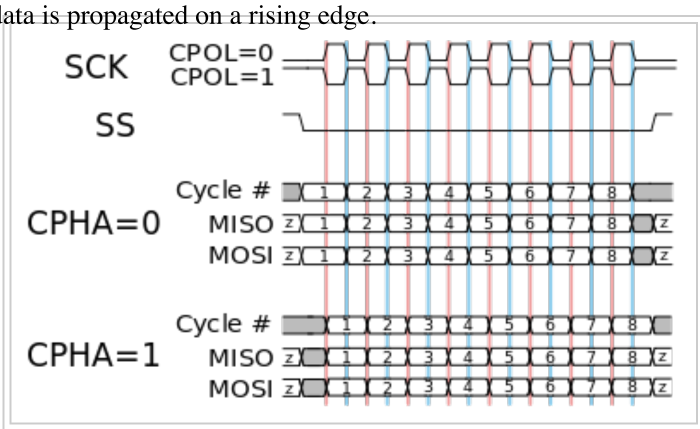
Every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

Clock polarity and phase

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Freescale's SPI Block Guide[2] names these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The 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
  - For CPHA=0, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).
  - For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
- At CPOL=1 the base value of the clock is one (inversion of CPOL=0)
  - For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
  - For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.



A timing diagram showing clock polarity and phase

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (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. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active.

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.

Some products use different naming conventions. For example, the TI MSP430 uses the name UCCKPL instead of CPOL, and its UCCKPH is the inverse of CPHA. When connecting two chips together, carefully examine the clock phase initialization values to be sure of using the right settings.

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:

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 SPI configuration

In the independent slave configuration, there is an independent chip select line for each slave. 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.

Daisy chain SPI configuration

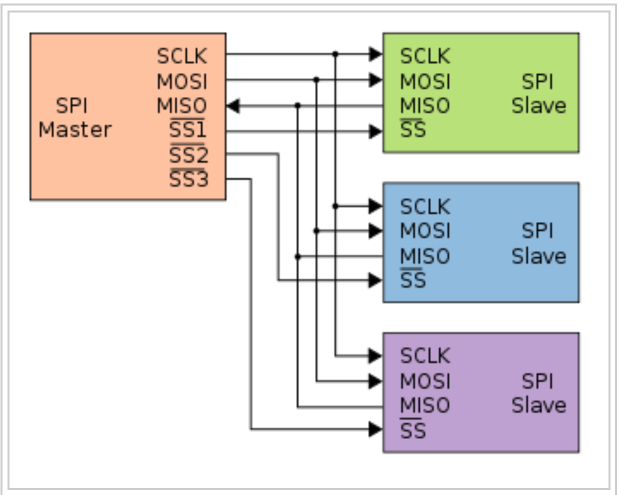
Some products with SPI bus are designed to be capable of being connected in a daisy chain configuration, the first slave output being connected to the second slave input, etc. The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of what it received during the first group of clock pulses. The whole chain acts as an SPI communication 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.[3]

Applications (discussed later) that require a daisy chain configuration include SGPIO and JTAG.

Valid SPI communications

Some slave devices are designed to ignore any SPI communications in which the number of clock pulses is greater than specified.

Others don't 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).



Typical SPI bus: master and three independent slaves

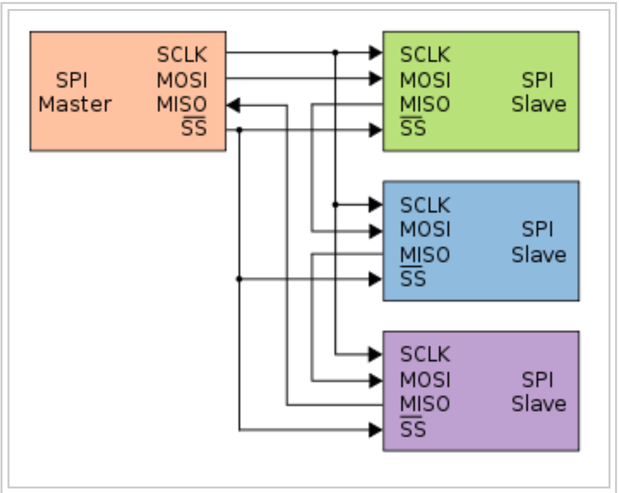
Interrupts

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, 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 SPI master protocol

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 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.

```
/* return: 0 is OK, */
unsigned char SPIWriteData(unsigned char byte)
{
    unsigned char bit;

    /* Negative clock polarity */
    SETGPHA(0);
    /* Data are propagated on a falling edge (high->low clock transitic
    SETGPOL(0);

    for (bit = 0; bit < 8; bit++) {
        /* delay between raise of clock */
        SPIDELAY(SPISPEED/2);

        SETCLK();

        /* delay between fall of clock */
        /* gives the h/w time to setup MISO line */
        SPIDELAY(SPISPEED - (SPISPEED/2));

        CLRCLK();

        /* write MOSI on falling edge of previous clock */
        if (byte & 0x80)
            SETMOSI();
        else
            CLRMOSI();
        byte <<= 1;
    }

    return byte;
}
```



Daisy-chained SPI bus: master and cooperative slaves

Pros and cons of SPI

Advantages

- Full duplex communication
- Higher throughput than I²C or SMBus
- 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
  - Slaves use the master's clock, and don't need precision oscillators
  - Slaves don't need a unique address — unlike I²C or GPIB or 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
- Not limited to any maximum clock speed, enabling potentially high throughput

Disadvantages

- 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)
- Supports only one master device
- No error-checking protocol is defined
- Generally prone to noise spikes causing faulty communication
- Without a formal standard, validating conformance is not possible
- Only handles short distances compared to RS-232, RS-485, or CAN-bus
- Many existing variations, making it difficult to find development tools like host adapters that support those variations
- SPI does not support hot plugging (dynamically adding nodes).

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 an SPI interface.[4]

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 displays, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant)

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.

JTAG is essentially an application stack for a three-wire SPI flavor, using different signal names[*citation needed*]: TCK not SCK, TDI not MOSI, TDO not MISO. It defines a state machine (driven by a TMS signal instead of a chip select line), protocol messages, a core command set, the ability to daisy-chain devices in a "scan chain", and how vendors define new commands. The devices in a scan chain are initially treated as a single device, and transitions on TMS update their state machines; once the individual devices are identified, commands may be issued that affect only one device in that scan chain. Different vendors use different JTAG connectors. Bit strings used in JTAG are often long and not multiples of 8 bit words; for example, a boundary scan reports signal state on each of several hundred pins.

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

Standards

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 or not 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 don't 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; this isn't usually called three-wire though. 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.

Development tools

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

Host adapters

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

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

The key parameters of SPI adapters 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-wires SPI (for example, support of quad-SPI protocol or other custom serial protocol that derive from SPI[5]).

Examples of SPI adapters *(manufacturers in alphabetical order):*

Manufacturer	SPI host adapter	Host bus	Bus protocol support	Max frequency
Byte Paradigm	SPI Storm (http://www.byteparadigm.com/product-spi-storm-39.html)	USB	SPI, dual/quad, custom	100 MHz
Corelis	BusPro-S (http://www.corelis.com/products-bus-analyzers/SPI_Exerciser_BusPro-S.htm)	USB	SPI, dual/quad	60 MHz
Microchip	MCP2210 Kit (http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en556988&part=ADM00421)	USB	SPI	12 MHz
National Instruments	USB-8541 (http://sine.ni.com/nips/cds/view/p/lang/en/nid/202368)	USB	I²C, SPI	12 MHz
Total Phase	Cheetah (http://www.totalphase.com/products/cheetah_spi/)	USB	SPI	40 MHz

Protocol analyzers

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

Oscilloscopes

Every major oscilloscope vendor offers 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.[6]

Logic analyzers

When developing and/or troubleshooting the SPI bus, examination of hardware signals can be very important. Logic analyzers are tools which collect, analyze, decode, 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.

Related terms

Intelligent SPI controllers

The **queued serial peripheral interface** (**QSPI**) is one type of SPI controller, not another bus type. It uses a data queue with programmable queue pointers allowing some data transfers without CPU intervention.[7] It also has a wrap-around mode allowing continuous transfers to and from the queue with no CPU intervention. 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**).[8] Most SPI master controllers integrate support for up to four chip selects,[9] although some require chip selects to be managed separately through GPIO lines.

Microwire

Microwire is essentially a predecessor of SPI. It's a strict subset: half duplex, and using SPI mode 0. (Microwire-Plus supports other SPI modes.) 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 (see below), which fits neatly with the restriction to half duplex.

Three-wire serial buses

As mentioned above, one variant of SPI uses single bidirectional data line (slave out/slave in, called SISO) instead of two unidirectional ones (MOSI and MISO). Clearly, 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. As of this writing, few SPI master controllers support this mode; although it can often be easily bit-banged in software.

When someone says a part supports SPI or Microwire, you can normally assume that means the four-wire version.

However, when someone talks about a part supporting a three-wire serial bus you should always find out what it means: standard four-wire SPI, without the chip select pin from that count, since most buses use chip selects but only three wires carry "real" signals; (More, sometimes with an unshared SPI bus segment the device's chip select will be hard-wired as "always selected".) "real" three-wire SPI; or even a RS-232 cable with RXD, TXD, and shield/ground, or an application-specific signaling scheme.

Multi I/O SPI

As opposed to three-wire serial buses, multi I/O SPI uses multiple parallel data lines (e.g., IO0 to IO3) to increase throughput. Dual I/O SPI using two data lines has comparable throughput to fast single I/O (MISO/MOSI). Quad I/O SPI using four data lines has approximately double the throughput.[10] Multi I/O SPI devices tend to be half duplex similar to three-wire devices to avoid adding too many pins. These serial memory devices combine the advantage of more speed with reduced pin count as compared to parallel memory.

See also

- Related buses
  - List of network buses
- Related connectors
  - Secure Digital - SD, SDHC, SDXC cards.
  - UEXT Connector.
- Universal Metering Interface (UMI)

References

1. ^ Note: on chips where a same shift register is used for both input and output (typical hardware), if one of the devices does not update in time the byte to be sent (especially the slave which has no control over the clock), the last data received will be sent back (like an "echo"). This may be taken into consideration when designing a new communication protocol over SPI, which has no flow control.
2. ^ SPI Block Guide V03.06 (http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf) , Freescale Semiconductor
3. ^ Maxim-IC application note 3947: "Daisy-Chaining SPI Devices" (http://www.maxim-ic.com/appnotes.cfm/an_pk/3947)
4. ^ AVR910 - In-system programming (http://www.atmel.com/dyn/resources/prod_documents/DOC0943.PDF)
5. ^ SPI Adapter (http://www.byteparadigm.com/product-spi-storm-39.html) with support of custom serial protocols, Byte Paradigm.
6. ^ "N5391B" (http://www.home.agilent.com/agilent/product.jspx?nid=-35491.898639.00&lc=eng&cc=US) . http://www.home.agilent.com/agilent/product.jspx?nid=-35491.898639.00&lc=eng&cc=US.
7. ^ Queued Serial Module Reference Manual (http://www.freescale.com/files/microcontrollers/doc/ref_manual/QSMRM.pdf) , Freescale Semiconductor
8. ^ Such as with the MultiChannel Serial Port Interface (http://focus.ti.com/docs/prod/folders/print/omap3530.html#technicaldocuments) , or McSPI, used in Texas Instruments OMAP chips. Download just the chapter about the SPI controller, if you want a good example of a highly engineered modern controller.
9. ^ Such at the SPI controller on Atmel AT91 chips (http://www.atmel.com/products/at91/) like the at91sam9G20, which is much simpler than TI's McSPI.
10. ^ Serial Peripheral Interface (SPI) Flash Memory Backgrounder (http://www.spansion.com/Products/Documents/Spansion_MirrorBit_SPI_MIO_backgrounder.pdf) , Spansion

External links

- Introduction to SPI and I2C protocols (http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html)
- Serial buses information page (http://www.epanorama.net/links/serialbus.html)
- SPI Introduction (http://www.mct.net/faq/spi.html) with helpful diagrams
- Serial Flash (http://www.serialflash.com/) Lots of good information on SPI part manufacturers and models.
- SPI - PICmicro Serial Peripheral Interface (http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf) , Microchip (company) tutorial on SPI.
- OpenCores (http://www.opencores.org/project,spi) Open Source implementation of SPI written in Verilog; simpler MC68HC11 version (http://www.opencores.org/?do=project&who=simple_spi)
- SPI flash memory controller (http://www.hdl-dh.com/prodbroch/HIP3100.01.04.2009.pdf) , HDL Design House

Navigation menu