

18. SPI – Serial Peripheral Interface

18.1 Features

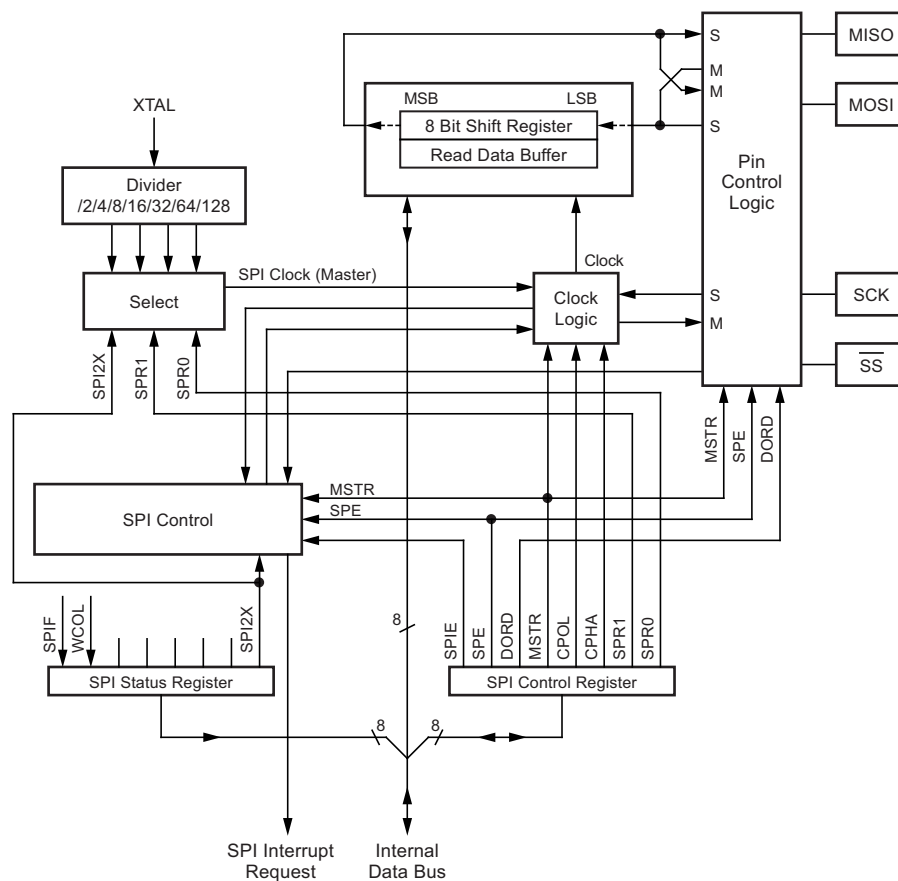
- Full-duplex, three-wire synchronous data transfer
- Master or slave operation
- LSB first or MSB first data transfer
- Seven programmable bit rates
- End of transmission interrupt flag
- Write collision flag protection
- Wake-up from idle mode
- Double speed (CK/2) master SPI mode

18.2 Overview

The serial peripheral interface (SPI) allows high-speed synchronous data transfer between the ATmega328P and peripheral devices or between several AVR® devices.

The USART can also be used in master SPI mode, see [Section 20. “USART in SPI Mode” on page 166](#). The PRSPI bit in [Section 9.10 “Minimizing Power Consumption” on page 36](#) must be written to zero to enable SPI module.

Figure 18-1. SPI Block Diagram⁽¹⁾



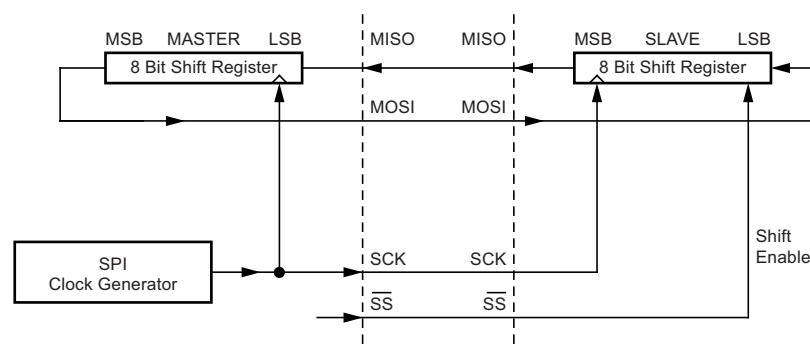
Note: 1. Refer to [Figure 1-1 on page 3](#), and [Table 13-3 on page 65](#) for SPI pin placement.

The interconnection between master and slave CPUs with SPI is shown in [Figure 18-2](#). The system consists of two shift registers, and a master clock generator. The SPI master initiates the communication cycle when pulling low the slave select \overline{SS} pin of the desired slave. master and slave prepare the data to be sent in their respective shift registers, and the master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from master to slave on the master out – slave In, MOSI, line, and from slave to master on the master In – slave out, MISO, line. After each data packet, the master will synchronize the Slave by pulling high the slave select, \overline{SS} , line.

When configured as a master, the SPI interface has no automatic control of the \overline{SS} line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI data register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI interrupt enable bit (SPIE) in the SPCR register is set, an interrupt is requested. The master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the slave select, \overline{SS} line. The last incoming byte will be kept in the buffer register for later use.

When configured as a slave, the SPI interface will remain sleeping with MISO tri-stated as long as the \overline{SS} pin is driven high. In this state, software may update the contents of the SPI data register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the \overline{SS} pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI interrupt enable bit, SPIE, in the SPCR register is set, an interrupt is requested. The slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the buffer register for later use.

Figure 18-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI data register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI data register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be:

Low periods: Longer than 2 CPU clock cycles.

High periods: Longer than 2 CPU clock cycles.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and \overline{SS} pins is overridden according to [Table 18-1](#). For more details on automatic port overrides, refer to [Section 13.3 “Alternate Port Functions” on page 63](#).

Table 18-1. SPI Pin Overrides⁽¹⁾

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User defined	Input
MISO	Input	User defined
SCK	User defined	Input
\overline{SS}	User defined	Input

Note: 1. See [Section 13.3.1 “Alternate Functions of Port B” on page 65](#) for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual data direction register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. E.g. if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

Assembly Code Example ⁽¹⁾
<pre> SPI_MasterInit: ; Set MOSI and SCK output, all others input ldi r17, (1<<DD_MOSI) (1<<DD_SCK) out DDR_SPI, r17 ; Enable SPI, Master, set clock rate fck/16 ldi r17, (1<<SPE) (1<<MSTR) (1<<SPR0) out SPCR, r17 ret SPI_MasterTransmit: ; Start transmission of data (r16) out SPDR, r16 Wait_Transmit: ; Wait for transmission complete in r16, SPSR sbrs r16, SPIF rjmp Wait_Transmit ret </pre>
C Code Example ⁽¹⁾
<pre> void SPI_MasterInit(void) { /* Set MOSI and SCK output, all others input */ DDR_SPI = (1<<DD_MOSI) (1<<DD_SCK); /* Enable SPI, Master, set clock rate fck/16 */ SPCR = (1<<SPE) (1<<MSTR) (1<<SPR0); } void SPI_MasterTransmit(char cData) { /* Start transmission */ SPDR = cData; /* Wait for transmission complete */ while(!(SPSR & (1<<SPIF))) ; } </pre>

Note: 1. See [Section 5. "About Code Examples" on page 8](#).

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example ⁽¹⁾
<pre> SPI_SlaveInit: ; Set MISO output, all others input ldi r17,(1<<DD_MISO) out DDR_SPI,r17 ; Enable SPI ldi r17,(1<<SPE) out SPCR,r17 ret SPI_SlaveReceive: ; Wait for reception complete sbis SPSR,SPIF rjmp SPI_SlaveReceive ; Read received data and return in r16,SPDR ret </pre>
C Code Example ⁽¹⁾
<pre> void SPI_SlaveInit(void) { /* Set MISO output, all others input */ DDR_SPI = (1<<DD_MISO); /* Enable SPI */ SPCR = (1<<SPE); } char SPI_SlaveReceive(void) { /* Wait for reception complete */ while(!(SPSR & (1<<SPIF))) ; /* Return Data Register */ return SPDR; } </pre>

Note: 1. See [Section 5. "About Code Examples" on page 8](#).

18.3 \overline{SS} Pin Functionality

18.3.1 Slave Mode

When the SPI is configured as a slave, the slave select (\overline{SS}) pin is always input. When \overline{SS} is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When \overline{SS} is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the \overline{SS} pin is driven high.

The \overline{SS} pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the \overline{SS} pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the shift register.

18.3.2 Master Mode

When the SPI is configured as a master (MSTR in SPCR is set), the user can determine the direction of the \overline{SS} pin.

If \overline{SS} is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the \overline{SS} pin of the SPI slave.

If \overline{SS} is configured as an input, it must be held high to ensure master SPI operation. If the \overline{SS} pin is driven low by peripheral circuitry when the SPI is configured as a master with the \overline{SS} pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCK pins become inputs.
2. The SPIF flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in master mode, and there exists a possibility that \overline{SS} is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI master mode.

18.4 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 18-3 and Figure 18-4 on page 140. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 18-3 on page 140 and Table 18-4 on page 141, as done in Table 18-2.

Table 18-2. SPI Modes

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

Figure 18-3. SPI Transfer Format with CPHA=0

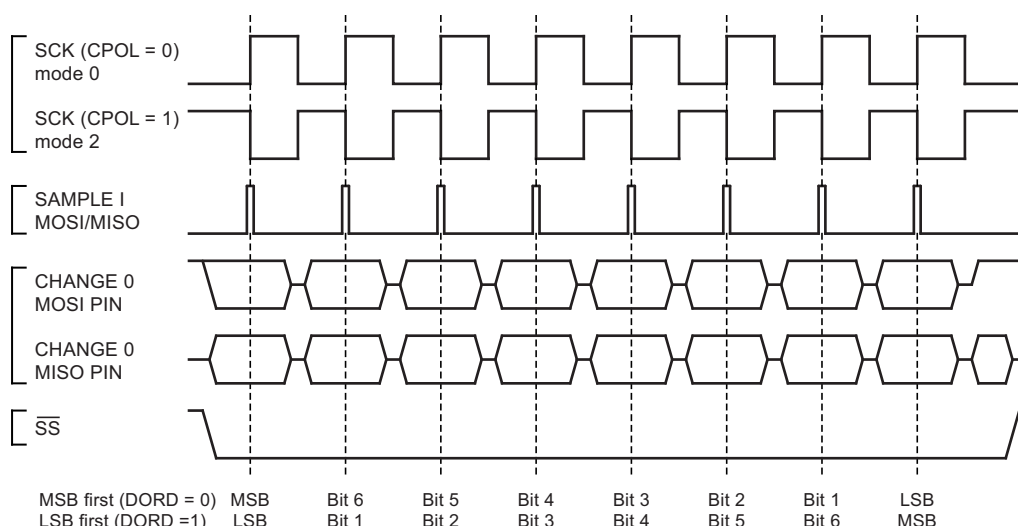
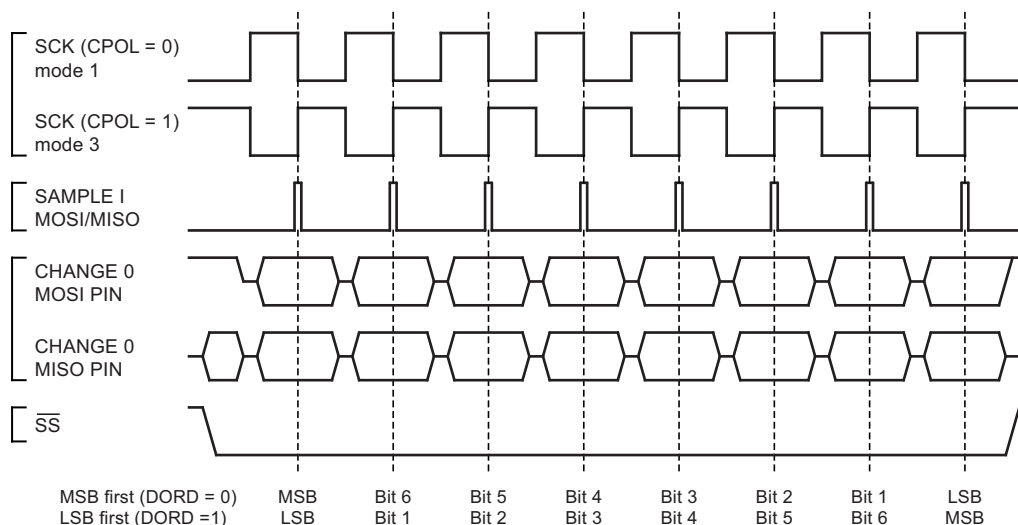


Figure 18-4. SPI Transfer Format with CPHA=1



18.5 Register Description

18.5.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects master SPI mode when written to one, and slave SPI mode when written logic zero. If \overline{SS} is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to [Figure 18-3 on page 139](#) and [Figure 18-4](#) for an example. The CPOL functionality is summarized below.

Table 18-3. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to [Figure 18-3 on page 139](#) and [Figure 18-4 on page 140](#) for an example. The CPOL functionality is summarized below:

Table 18-4. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a master. SPR1 and SPR0 have no effect on the slave. The relationship between SCK and the oscillator clock frequency f_{osc} is shown in [Table 18-5](#).

Table 18-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

18.5.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If \overline{SS} is an input and is driven low when the SPI is in master mode, this will also set the SPIF flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI data register (SPDR).

- **Bit 6 – WCOL: Write COLLision Flag**

The WCOL bit is set if the SPI data register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI status register with WCOL set, and then accessing the SPI data register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved bits in the Atmel® ATmega328P and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK frequency) will be doubled when the SPI is in master mode (see [Table 18-5 on page 141](#)). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

The SPI interface on the Atmel ATmega328P is also used for program memory and EEPROM downloading or uploading. See [Section 27.8 “Serial Downloading” on page 254](#) for serial programming and verification.

18.5.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI data register is a read/write register used for data transfer between the register file and the SPI shift register. Writing to the register initiates data transmission. Reading the register causes the shift register Receive buffer to be read.