Serial Peripheral Interface

The Serial Peripheral Interface (SPI) is a synchronous interface allowing several SPI-type devices to be interconnected. The SPI is a full-duplex, synchronous, character-oriented communication channel that employs a four-wire interface. The SPI block consists of a transmitter, receiver, baud rate generator, and control unit. During an SPI transfer, data is sent and received simultaneously by both the master and the slave SPI devices.

In a serial peripheral interface, separate signals are required for data and clock. The SPI is configured either as a master or as a slave. The connection of two SPI devices (one master and one slave) and the direction of data transfer is demonstrated in Figures 40 and 41.

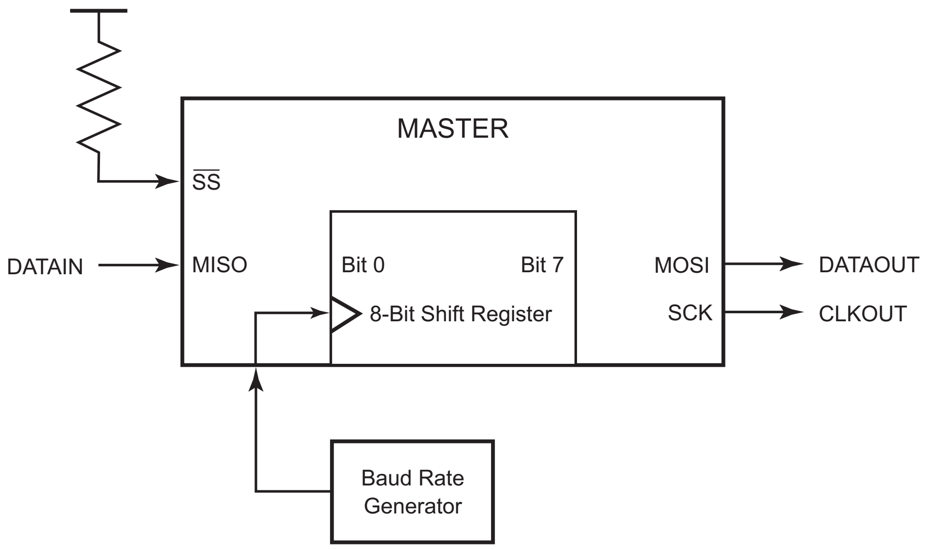

Figure 40. SPI Master Device

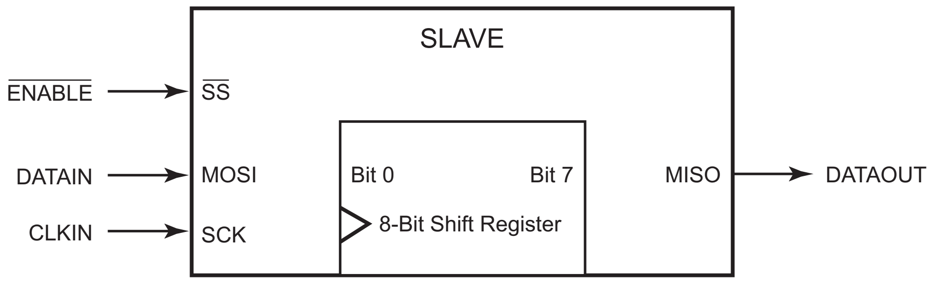

Figure 41. SPI Slave Device

SPI Signals

The four basic SPI signals are:

- MISO (Master In, Slave Out)
- MOSI (Master Out, Slave In)
- SCK (SPI Serial Clock)
- $\overline{\text{SS}}$ (Slave Select)

These SPI signals are discussed in the following paragraphs. Each signal is described in both MASTER and SLAVE modes.

Master In, Slave Out

The Master In, Slave Out (MISO) pin is configured as an input in a master device and as an output in a slave device. It is one of the two lines that transfer serial data, with the most-significant bit sent first. The MISO pin of a slave device is placed in a high-impedance state if the slave is not selected. When the SPI is not enabled, this signal is in a high-impedance state.

Master Out, Slave In

The Master Out, Slave In (MOSI) pin is configured as an output in a master device and as an input in a slave device. It is one of the two lines that transfer serial data, with the most-significant bit sent first. When the SPI is not enabled, this signal is in a high-impedance state.

Slave Select

The active Low Slave Select ($\overline{\text{SS}}$) input signal is used to select the SPI as a slave device. It must be Low prior to all data communication and must stay Low for the duration of the data transfer.

The $\overline{\text{SS}}$ input signal must be High for the SPI to operate as a master device. If the $\overline{\text{SS}}$ signal goes Low in Master Mode, a Mode Fault error flag (MODF) is set in the SPI_SR Register. For more information, see the SPI Status Register section on page 206.

When the clock phase (CPHA) is set to 0, the shift clock is the logic OR of $\overline{\text{SS}}$ with SCK. In this clock phase mode, $\overline{\text{SS}}$ must go High between successive characters in an SPI message. When CPHA is set to 1, $\overline{\text{SS}}$ remains Low for several SPI characters. In cases in which there is only one SPI slave, its $\overline{\text{SS}}$ line could be tied Low as long as CPHA is set to 1. For more information about CPHA, see the SPI Control Register section on page 205.

Serial Clock

The Serial Clock (SCK) is used to synchronize data movement both in and out of the device via its MOSI and MISO pins. The master and slave are each capable of exchanging a byte of data during a sequence of eight clock cycles. Because SCK is generated by the master, the SCK pin becomes an input on a slave device. The SPI contains an internal divide-by-two clock divider. In MASTER Mode, the SPI serial clock is one-half the frequency of the clock signal created by the SPI Baud Rate Generator.

As demonstrated in Figure 42 and Table 111, four possible timing relations are chosen by using the clock polarity (CPOL) and clock phase CPHA control bits in the SPI Control Register. See the SPI Control Register section on page 205. Both the master and slave must operate with the identical timing, CPOL, and CPHA. The master device always places data on the MOSI line a half-cycle before the clock edge (SCK signal), for the slave device to latch the data.

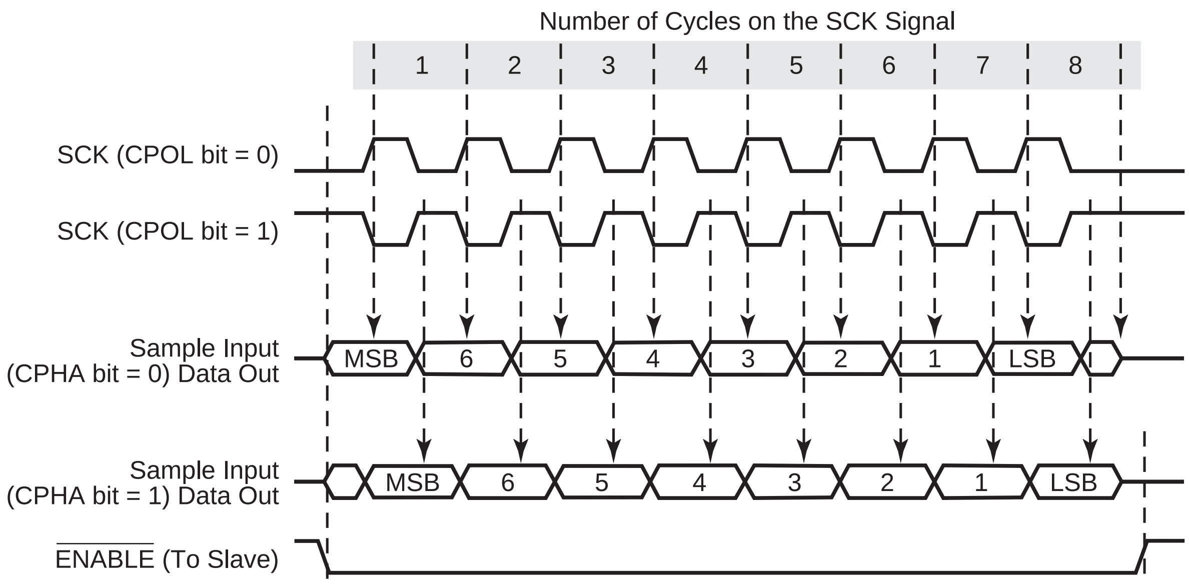

Figure 42. SPI Timing

Table 111. SPI Clock Phase and Clock Polarity Operation

CPHA	CPOL	SCK Transmit Edge	SCK Receive Edge	SCK Idle State	$\overline{\text{SS}}$ High Between Characters?
0	0	Falling	Rising	Low	Yes
0	1	Rising	Falling	High	Yes

Table 111. SPI Clock Phase and Clock Polarity Operation (Continued)

CPHA	CPOL	SCK Transmit Edge	SCK Receive Edge	SCK Idle State	$\overline{\text{SS}}$ High Between Characters?
1	0	Rising	Falling	Low	No
1	1	Falling	Rising	High	No

SPI Functional Description

When a master transmits to a slave device via the MOSI signal, the slave device responds by sending data to the master via the master's MISO signal. The result is a full-duplex transmission, with both *data out* and *data in* synchronized with the same clock signal. The byte transmitted is replaced by the byte received, eliminating the need for separate transmit-empty and receive-full status bits. A single status bit, SPIF, is used to signify that the I/O operation is complete. See the SPI Status Register section on page 206.

The SPI is double-buffered during reads, but not during writes. If a write is performed during data transfer, the transfer occurs uninterrupted, and the write is unsuccessful. This condition causes the write collision (WCOL) status bit in the SPI_SR Register to be set. After a data byte is shifted, the SPI flag of the SPI_SR Register is set to 1.

In SPI MASTER Mode, the SCK pin functions as an output. It idles High or Low depending on the CPOL bit in the SPI_CTL Register until data is written to the shift register. Data transfer is initiated by writing to the transmit shift register, SPI_TSR. Eight clocks are then generated to shift the eight bits of transmit data out via the MOSI pin while shifting in eight bits of data via the MISO pin. After transfer, the SCK signal becomes idle.

In SPI SLAVE Mode, the start logic receives a logic Low from the $\overline{\text{SS}}$ pin and a clock input at the SCK pin; as a result, the slave is synchronized to the master. Data from the master is received serially from the slave MOSI signal and is loaded into the 8-bit shift register. After the 8-bit shift register is loaded, its data is parallel-transferred to the read buffer. During a write cycle, data is written into the shift register. Next, the slave waits for the SPI master to initiate a data transfer, supply a clock signal, and shift the data out on the slave's MISO signal.

If the CPHA bit in the SPI_CTL Register is 0, a transfer begins when the $\overline{\text{SS}}$ pin signal goes Low. The transfer ends when $\overline{\text{SS}}$ goes High after eight clock cycles on SCK. When the CPHA bit is set to 1, a transfer begins the first time SCK becomes active while $\overline{\text{SS}}$ is Low. The transfer ends when the SPI flag is set to 1.

SPI Flags

This section describes the SPI Mode Fault and Write Collision flags.

Mode Fault

The Mode Fault flag (MODF) indicates that there is a multimaster conflict in the system control. The MODF bit is normally cleared to 0 and is only set to 1 when the master device's $\overline{\text{SS}}$ pin is pulled Low. When a mode fault is detected, the following sequence occurs:

1. The MODF flag (SPI_SR[4]) is set to 1.
2. The SPI device is disabled by clearing the SPI_EN bit (SPI_CTL[5]) to 0.
3. The MASTER_EN bit (SPI_CTL[4]) is cleared to 0, forcing the device into SLAVE Mode.
4. If the SPI interrupt is enabled by setting IRQ_EN (SPI_CTL[7]) High, an SPI interrupt is generated.

Clearing the Mode Fault flag is performed by reading the SPI Status Register. The other SPI control bits (SPI_EN and MASTER_EN) must be restored to their original states by user software after the Mode Fault Flag is cleared to 0.

Write Collision

The write collision flag, WCOL (SPI_SR[5]), is set to 1 when an attempt is made to write to the SPI Transmit Shift Register (SPI_TSR) while data transfer occurs. Clearing the WCOL bit is performed by reading SPI_SR with the WCOL bit set to 1.

SPI Baud Rate Generator

The SPI Baud Rate Generator (BRG) creates a lower frequency clock from the high-frequency system clock. The BRG output is used as the clock source by the SPI.

Baud Rate Generator Functional Description

The SPI BRG consists of a 16-bit downcounter, two 8-bit registers, and associated decoding logic. The BRG's initial value is defined by the two BRG Divisor Latch registers {SPI_BRG_H, SPI_BRG_L}. At the rising edge of each system clock, the BRG decrements until it reaches the value `0001h`. On the next system clock rising edge, the BRG reloads the initial value from {SPI_BRG_H, SPI_BRG_L) and outputs a pulse to indicate the end of the count.

The SPI Data Rate is calculated using the following equation:

$$\text{SPI Data Rate (bits/s)} = \frac{\text{System Clock Frequency}}{2 \times \text{SPI Baud Rate Generator Divisor}}$$

Upon RESET, the 16-bit BRG divisor value resets to `0002h`. When the SPI is operating as a Master, the BRG divisor value must be set to a value of `0003h` or greater. When the SPI is operating as a Slave, the BRG divisor value must be set to a value of `0004h` or greater. A software write to either the Low- or High-byte registers for the BRG Divisor Latch causes both the low and high bytes to load into the BRG counter, and causes the count to restart.

Data Transfer Procedure with SPI Configured as a Master

The following list describes the procedure for transferring data from a master SPI device to a slave SPI device.

1. Load the SPI BRG Registers, SPI_BRG_H and SPI_BRG_L. The external device must deassert the $\overline{\text{SS}}$ pin if currently asserted.
2. Load the SPI Control Register, SPI_CTL.
3. Assert the $\overline{\text{ENABLE}}$ pin of the slave device using a GPIO pin.
4. Load the SPI Transmit Shift Register, SPI_TSR.
5. When the SPI data transfer is complete, deassert the $\overline{\text{ENABLE}}$ pin of the slave device.

Data Transfer Procedure with SPI Configured as a Slave

The following list describes the procedure for transferring data from a slave SPI device to a master SPI device.

1. Load the SPI BRG Registers, SPI_BRG_H and SPI_BRG_L.
2. Load the SPI Transmit Shift Register, SPI_TSR. This load cannot occur while the SPI slave is currently receiving data.
3. Wait for the external SPI Master device to initiate the data transfer by asserting $\overline{\text{SS}}$.

SPI Registers

There are six registers in the Serial Peripheral Interface that provide control, status, and data storage functions. The SPI registers are described in the following paragraphs.

SPI Baud Rate Generator Low Byte and High Byte Registers

These registers hold the low and high bytes of the 16-bit divisor count loaded by the CPU for baud rate generation. The 16-bit clock divisor value is returned by {SPI_BRG_H, SPI_BRG_L}. Upon RESET, the 16-bit BRG divisor value resets to `0002h`. When configured as a Master, the 16-bit divisor value must be between `0003h` and `FFFFh`, inclusive. When configured as a Slave, the 16-bit divisor value must be between `0004h` and `FFFFh`, inclusive.

A write to either the Low- or High-byte registers for the BRG Divisor Latch causes both bytes to be loaded into the BRG counter and a restart of the count. See Tables 112 and 113.

Table 112. SPI Baud Rate Generator Low Byte Register (SPI_BRG_L)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	SPI_BRG_L							
Reset	0	0	0	0	0	0	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	00B8h							

Note: R/W = read/write.

Bit	**Description**
[7:0] SPI_BRG_L	**BRG Low Byte** 00h–FFh: These bits represent the low byte of the 16-bit BRG divider value. The complete BRG divisor value is returned by {SPI_BRG_H, SPI_BRG_L}.

Table 113. SPI Baud Rate Generator High Byte Register (SPI_BRG_H)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	SPI_BRG_H							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	00B9h							

Note: R/W = read/write.

Bit	**Description**
[7:0] SPI_BRG_H	**BRG High Byte** 00h–FFh: These bits represent the high byte of the 16-bit BRG divider value. The complete BRG divisor value is returned by {SPI_BRG_H, SPI_BRG_L}.

SPI Control Register

This register is used to control and setup the serial peripheral interface. The SPI must be disabled prior to making any changes to CPHA or CPOL. See Table 114.

Table 114. SPI Control Register (SPI_CTL)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	IRQ_EN	Reserved	SPI_EN	MASTER_EN	CPOL	CPHA	Reserved	
Reset	0	0	0	0	0	1	0	0
R/W	R/W	R	R/W	R/W	R/W	R/W	R	R
Address	00BAh							

Note: R = read only; R/W = read/write.

Bit	**Description**
[7] IRQ_EN	**SPI Interrupt Request Enable** 0: SPI system interrupt is disabled. 1: SPI system interrupt is enabled.
[6]	**Reserved** This bit is reserved and must be programmed to 0.
[5] SPI_EN	**Serial Peripheral Interface Enable** 0: SPI is disabled. 1: SPI is enabled.
[4] MASTER_EN	**SPI Mode Enable** 0: When enabled, the SPI operates as a slave. 1: When enabled, the SPI operates as a master.
[3] CPOL	**Clock Polarity** 0: Master SCK pin idles in a Low (0) state. 1: Master SCK pin idles in a High (1) state.
[2] CPHA	**Clock Phase** 0: $\overline{SS}$ must go High after transfer of every byte of data. 1: $\overline{SS}$ remains Low to transfer any number of data bytes.
[1:0]	**Reserved** These bits are reserved and must be programmed to 00.

SPI Status Register

The read-only SPI Status Register returns the status of data transmitted using the serial peripheral interface. Reading the SPI_SR Register clears Bits 7, 6, and 4 to a logic 0. See Table 115.

Table 115. SPI Status Register (SPI_SR)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	SPIF	WCOL	Reserved	MODF	Reserved			
Reset	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R
Address	00BBh							

Note: R = read only.

Bit	Description
[7] SPIF	**SPI Flag** 0: SPI data transfer is not finished. 1: SPI data transfer is finished. If enabled, an interrupt is generated. This bit flag is cleared to 0 by a read of the SPI_SR Register.
[6] WCOL	**SPI Write Collision** 0: An SPI write collision is not detected. 1: An SPI write collision is detected. This bit Flag is cleared to 0 by a read of the SPI_SR registers.
[5]	**Reserved** This bit is reserved and must be programmed to 0.
[4] MODF	**SPI Mode Fault** 0: A mode fault (multimaster conflict) is not detected. 1: A mode fault (multimaster conflict) is detected. This bit Flag is cleared to 0 by a read of the SPI_SR Register.
[3:0]	**Reserved** These bits are reserved and must be programmed to 0000.

SPI Transmit Shift Register

The SPI Transmit Shift Register (SPI_TSR) is used by the SPI master to transmit data over an SPI serial bus to a slave device. A write to the SPI_TSR Register places data directly into the shift register for transmission. A write to this register within an SPI device configured as a master initiates transmission of the byte of the data loaded into the register. At the completion of transmitting a byte of data, the SPI Flag (SPI_SR[7]) is set to 1 in both the master and slave devices.

The write-only SPI Transmit Shift Register shares the same address space as the read-only SPI Receive Buffer Register. See Table 116.

Table 116. SPI Transmit Shift Register (SPI_TSR)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	Tx_DATA							
Reset	U	U	U	U	U	U	U	U
R/W	W	W	W	W	W	W	W	W
Address	00BCh							

Note: U = undefined; W = write only.

Bit	Description
[7:0] Tx_DATA	**SPI Transmit Data** 00h–FFh: SPI transmit data.

SPI Receive Buffer Register

The SPI Receive Buffer Register (SPI_RBR), shown in Table 117, is used by the SPI slave to receive data from the serial bus. The SPIF bit must be cleared prior to a second transfer of data from the shift register; otherwise, an overrun condition exists. In the event of an overrun, the byte that causes the overrun is lost.

The read-only SPI Receive Buffer Register shares the same address space as the write-only SPI Transmit Shift Register.

Table 117. SPI Receive Buffer Register (SPI_RBR)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	Rx_DATA							
Reset	U	U	U	U	U	U	U	U
R/W	R	R	R	R	R	R	R	R
Address	00BCh							

Note: U = undefined; R = read only.

Bit	**Description**
[7:0] Rx_DATA	00h–FFh: SPI received data.