# 22 SPI

## 22.1 Overview

Serial Peripheral Interface (SPI) is a serial synchronous communication method for external devices to exchange data through 4 wires.

The chip provides two SPI interface modules, which can be configured as a master device or a slave device to achieve SPI communication with the outside.

Features:

ÿ Full-duplex 4-wire serial synchronous transceiver (SCLK, MOSI, MISO, SSN)

ÿ MISO and MOSI can swap pin order

ÿ Half-duplex 4-wire serial synchronous transceiver (SCLK, SDATA, SSN, DCN)

ÿ 2 independent channels

ÿ Master-slave mode

ÿ Programmable clock polarity and phase

ÿ Programmable bit rate

ÿ Programmable data word length (8/16/24/32 bits)

ÿThe maximum baud rate is FAPBCLK/2

ÿ Transmission end interrupt flag

ÿ Write conflict error flag

ÿ Master mode error detection, protection and interrupt flags

ÿ Support DMA

## 22.2 Block Diagram

The figure below is a schematic diagram of the structure of the SPI module.

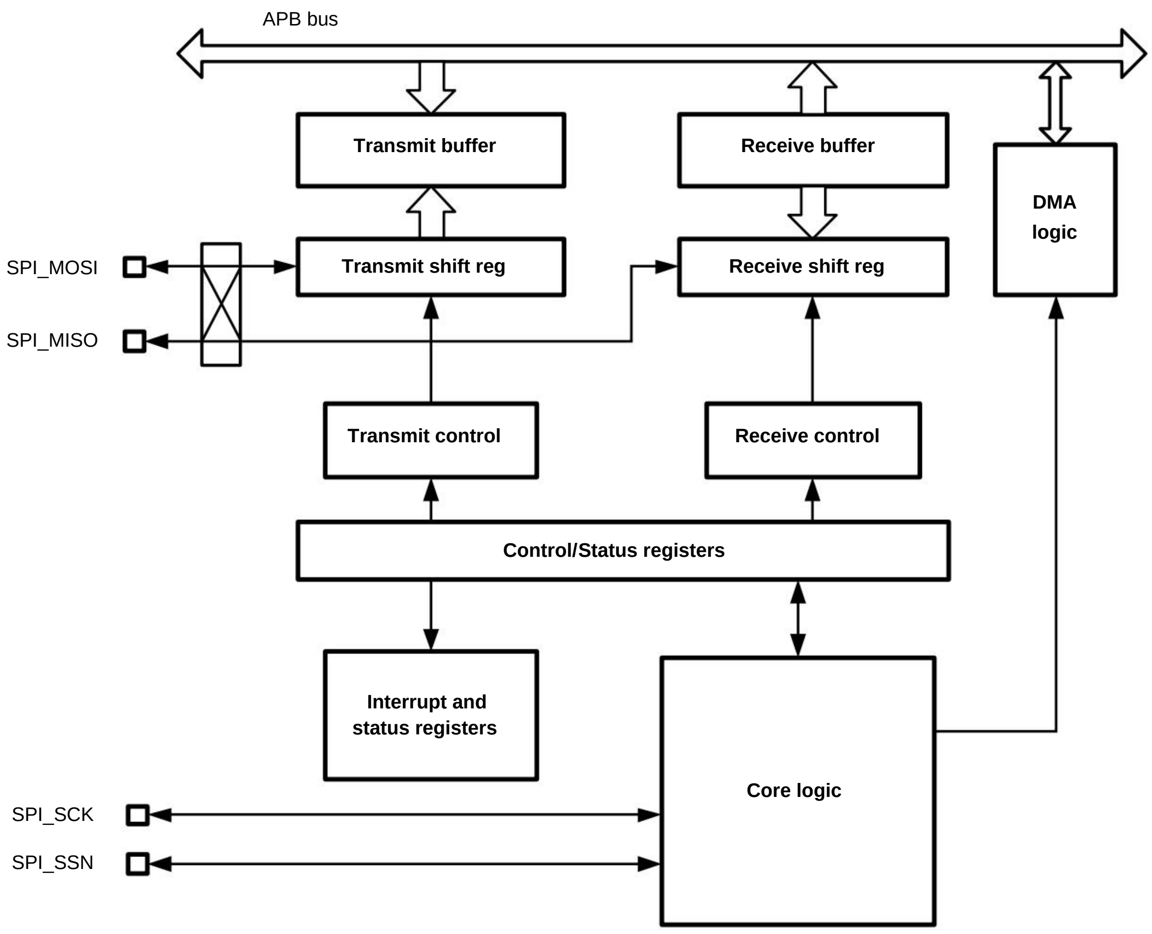

Figure **22-1 SPI** structure diagram

## **22.3** Pin Definition

The SPI module uses 4 pins to communicate with external devices. In full-duplex and half-duplex modes, the functional definitions of these four pins are different.

As shown in the following table:

| Pin **SPIx** full duplex | | | Function | Half-duplex | Function |
|---|---|---|---|---|---|
| PB8/PD2 | SPI1 | SSN | chip select | SSN | Chip select signal |
| PB9/PD3 | | SCLK | signal clock | SCLK | clock |
| PB10/PD4 | | MISO Master Input Slave Output | | DCN | Command/data identification |
| PB11/PD5 | | MOSI Master output Slave input SDATA Chip select signal clock | | | Bidirectional data |
| PC7 | SPI2 | SSN | | SSN | Chip select signal |
| PC8 | | SCLK | | SCLK | clock |
| PC9 | | MISO Master Input Slave Output | | DCN | Command/data identification |
| PC10 | | MOSI master output slave input SDATA | | | Bidirectional data |

## **22.4** Interface Timing

In order to be compatible with different SPI peripherals, the timing of the SPI serial clock can be controlled by the clock phase selection bit (CPHA) and the clock polarity selection bit

To ensure data transfer, the timing configuration of the master and slave devices must be consistent.

When in slave mode or the SPI system enable bit (SPE) is 0, the SCK pin of the SPI has no serial clock output.

## 22.4.1 CPHA=0

When CPHA=0, the SPI module samples data at the first transition edge of the serial clock, that is:

If CPOL=1, SCK stays at high level when the bus is IDLE, SPI samples data at the falling edge of the serial clock and

The rising edge sends data;

If CPOL=0, SCK stays at low level when the bus is IDLE, SPI samples data at the rising edge of the serial clock and samples data at the falling edge of the serial clock.

The falling edge sends data.

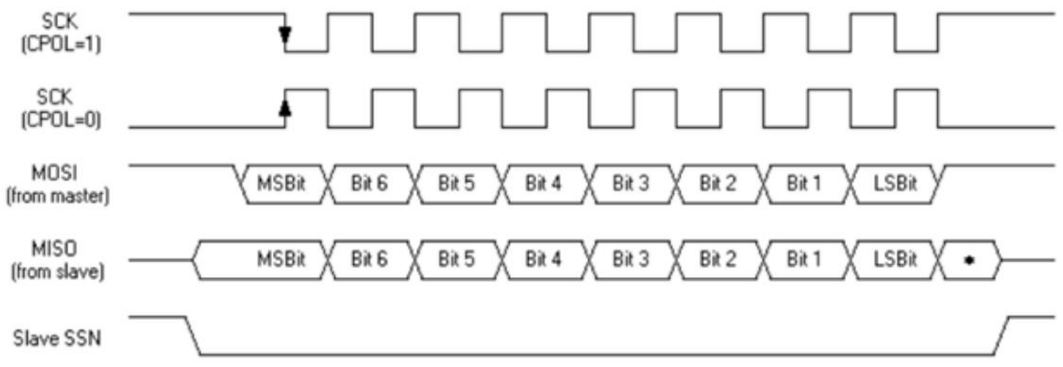

Figure **22-2 SPI** data/clock timing diagram (CPHA=0)

## 22.4.2 CPHA=1

When CPHA=1, the SPI module samples data at the second transition edge of the serial clock, that is:

If CPOL=1, SCK stays at a high level when the bus is IDLE, data is sampled on the rising edge of the serial clock, and data is sampled on the falling edge of the serial clock.

Send data along the

If CPOL=0, SCK stays at a low level when the bus is IDLE, data is sampled on the falling edge of the serial clock, and data is sampled on the rising edge of the serial clock.

Send data.

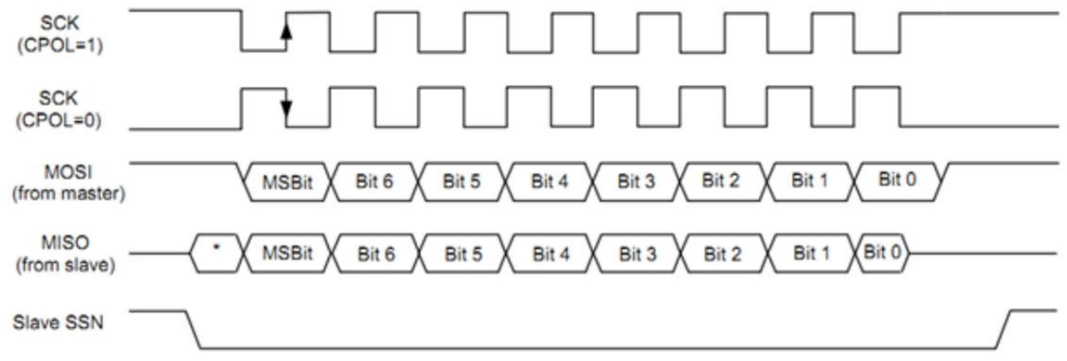

Figure **22-3 SPI** data/clock timing diagram (CPHA=1)

## 22.4.1 4- wire half-duplex mode (host)

4-wire half-duplex mode can support interactive communication with dot matrix LCD or TFT screen. In this mode, the DCN signal is used to distinguish

The current frame is a command frame or a data frame. Both bidirectional data are sent and received through the SDATA (MOSI) pin, which is automatically completed by the hardware.

Data direction switching. The SPI of FM33LC0XX only supports 4-wire half-duplex master mode, not slave mode.

All communications are initiated by the host, which first sends a command frame and then transmits a data frame. The command frame and data frame are transmitted through the DCN.

Signal line distinction. The host can write data to the slave or read data from the slave through a 4-wire half-duplex interface.

4-wire half-duplex write operation

The software clears the HD_RW register to indicate that the current host is about to initiate a write operation.

Before the host initiates a write operation, it first sends a write command frame. After the write command frame is sent, if the send buffer is empty, the hardware will

Pull up SSN and stop SCLK transmission; if new data has been written into the transmit buffer, the hardware will continuously send subsequent data frames.

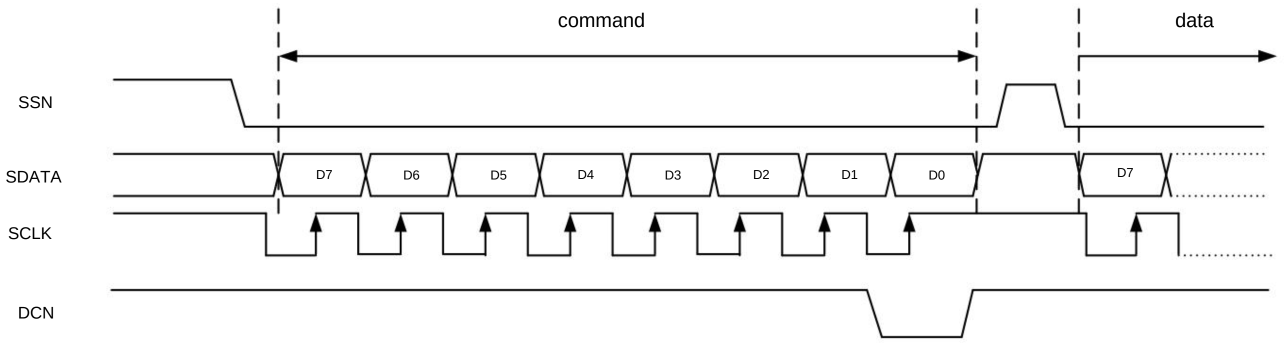

Figure **22-4 4** -wire half-duplex write operation

DCN is sampled and judged at the rising edge of the 8th clock. If it is 0, it means that the current frame is a command frame. Before sending a command frame, the software needs to set DCN

The register is written to 0, and the hardware automatically sets the DCN register after the command frame is sent.

4-wire half-duplex read operation

The software sets the HD_RW register to indicate that the current host wants to initiate a read operation.

4-wire half-duplex read operations support 8-bit, 24-bit, and 32-bit reads. When the host initiates a read operation, it first sends a read command frame.

After the transmission is completed, a dummy cycle can be sent according to the register configuration. During the dummy cycle, the SCLK clock is normally sent.

However, the host does not drive SDATA and does not accept SDATA input.

After completing the command frame and dummy cycle (optional), the 4-wire half-duplex SPI automatically enters the receiving state, and the SDATA signal is driven by the slave.

The data frame received by the host will be written into the receive buffer. After each data frame is received, the RXBF interrupt flag register will be set.

The software should read the data in the receive buffer in time. If the receive buffer and the receive shift register are both full, the hardware will stop.

Stop SCLK transmission and pause reading data from the slave until the software or DMA reads the receive buffer.

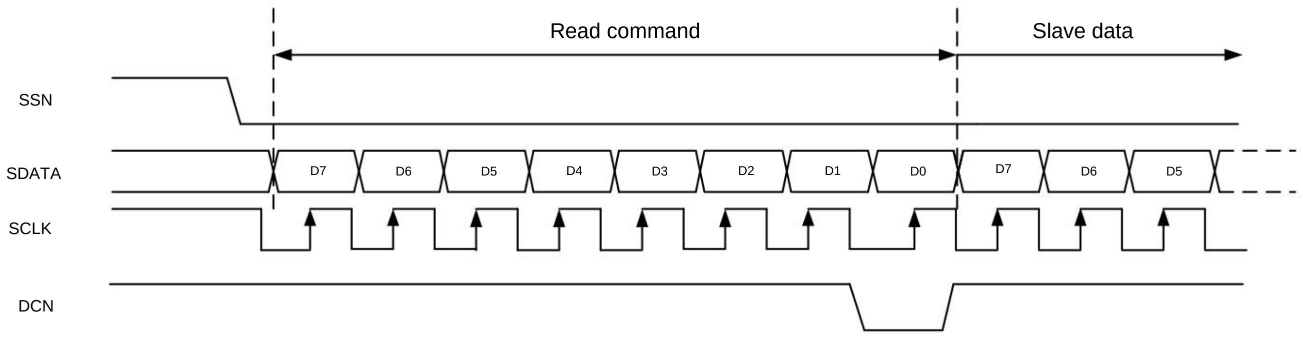

Figure **22-5 4** -wire half-duplex read operation (without **dummy cycle)**

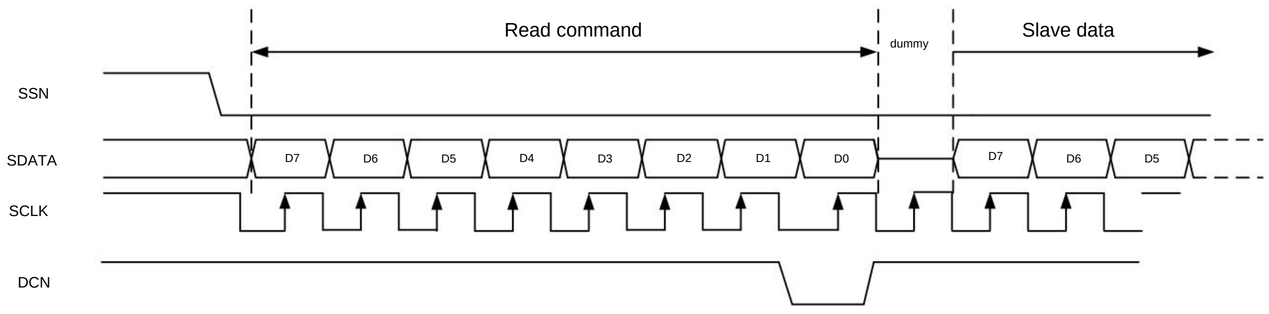

Figure **22-6 4** -wire half-duplex read operation (with **dummy cycle)**

## **22.5** Functional Description

### **22.5.1 I/O** Configuration

Master output, slave input (MOSI)

The Master Out Slave In (MOSI) pin is the output of the master device and the input of the slave device, and is used for serial data transmission from the master device to the slave device.

When the SPI is configured as a master, this pin is an output, and when the SPI is configured as a slave, this pin is an input. Data is transferred MSB first.

Master input, slave output (MISO)

The Master-In-Slave-Out (MISO) pin is an output from the slave and an input to the master, and is used for serial data transfer from the slave to the master.

When the SPI is configured as a master, this pin is an input, and when the SPI is configured as a slave, this pin is an output. Data is transferred MSB first.

Serial Clock (SCK)

The serial clock (SCK) pin is an output from the master device and an input from the slave device. It is used to synchronize the serial clock between the master and slave devices on MOSI and

When the SPI is configured as a master device, this pin outputs the clock. When the SPI is configured as a slave device, this pin outputs the clock.

The pin is an input.

**From Select (SSN)**

The SSN pin is used to control the selection of the slave device. As shown in Figure 22-2, when the SPI is configured as a master device, the SSN pin must be connected to

High level, when SPI is configured as a slave device, the SSN pin must be connected to a low level.

The connection of SPI **master and slave devices is shown in Error! Reference source not found.**

The MOSI, MISO and SCK of the master and slave devices are connected together respectively. The SSN of the master device must be connected to a high level, and the SSN of the slave device must be connected to a high level.

The master and slave devices are connected into a loop through MOSI and MISO. The master device outputs the clock. When data is transmitted, the master device

The master and slave devices output data through MOSI and MISO respectively. After a byte of data is transmitted, the master and slave devices will exchange 8 bits.

Register value.

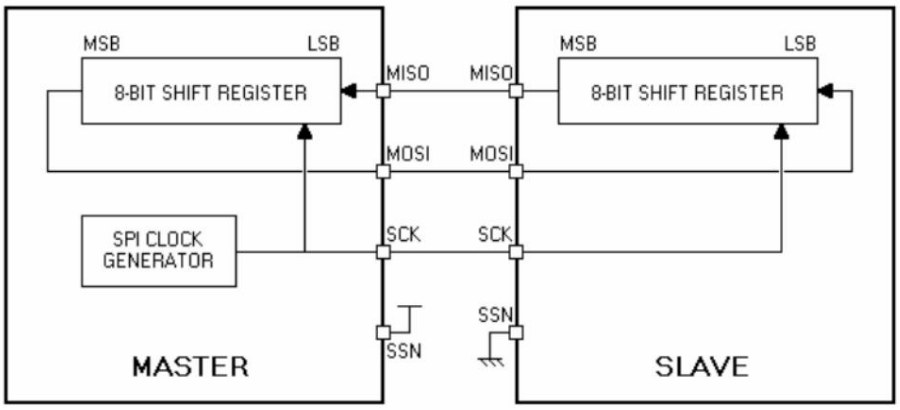

Figure **22-7 SPI Master/SPI Slave** Interconnection

## 22.5.2 Full-duplex Data Communication

The SPI module defaults to full-duplex communication. If continuous and uninterrupted data communication is required, the software needs to ensure that the TX BUFFER is not empty.

Even if the software only uses SPI for data reception, due to the full-duplex property of SPI, the software still needs to write to the TX BUFFER.

What is written at this time is invalid data, and all 0s or all Fs can be written according to the MOSI invalid state configuration.

Send Buffer

The software or DMA writes the data to be sent into the transmit buffer (SPIxTXBUF register). When the transmission starts, the hardware writes the data from the transmit buffer to the SPIxTXBUF register.

The send buffer is copied to the shift register and starts sending. After the data is transferred from the send buffer to the shift register, the send buffer empty flag is

(TXBE) is set, indicating that new data can be written to TXBUF; if the TXIE register is set, an interrupt is generated.

Writing data can clear the TXBE register.

If new data is written into the transmit buffer before the shift register is completed, continuous data transmission can be guaranteed.

If TXBE is 0 and TXBUF is written, data conflict will occur. See 22.5.6 Data Conflict.

Receive Buffer

When the SPI completes receiving a frame of data, the received data will be copied from the shift register to the receive buffer (SPIxRXBUF register).

At the same time, the RXBF flag is set, indicating that there is data to be processed in RXBUF. If the RXIE register is set, an interrupt is generated.

Reading RXBUF can clear the RXBF flag.

Reading RXBUF when RXBF is not set will return the last received data; if the application does not process RXBF in time,

If new data is received while RXBF is set, a data conflict occurs. See 22.5.6 Data Conflict.

**BUSY logo**

When SPI is sending or receiving data, the BUSY register is set. This register can be used to determine the last frame of data in some scenarios.

For example, TXBE only indicates that the data has entered the shift transmission, but the actual transmission is completed, and the BUSY mark must be waited for.

Ambition cleared.

**How to start SPI communication**

In host mode, it is recommended to follow the following steps to start SPI communication:

ÿ Application configuration SPI module

ÿ Set SPIEN

ÿ Write data to TXBUF, the SPI module automatically starts sending SCK and receiving data

In slave mode, it is recommended that the application complete configuration and enable before the host starts sending SCK, and write the first frame of data to be sent into

TXBUF, waiting for the host to send SCK to start communication.

**How to end SPI communication**

In host mode, it is recommended to follow the following steps to end SPI communication:

ÿ Wait for the RXBF and TXBE flags to be set. At this time, the last frame of data is being sent in the shift register.

ÿ Check the BUSY flag until BUSY is 0, and the last frame of data is sent and received.

ÿ Turn off the SPI module and read the last frame of received data if necessary

In slave mode, the application can shut down the SPI module after reading any frame of data. The data that has been shifted into the shift register before shutting down will be

is ignored.

## 22.5.3 TX-ONLY Mode

Sometimes SPI communication is half-duplex. When the host only needs to send, it can enter TX-ONLY by setting the TXO register.

Mode, the data received by MISO will not be written into the RX Buffer, and the RXBF interrupt flag will not be set accordingly.

By setting TXO_AC, the TXO automatic clearing function can be realized. In TX-ONLY mode, if the TX buffer is empty (TXBE is set

bit) and the transmit shift register is empty, the TXO register is automatically cleared to exit the TX-ONLY state.

## 22.5.4 RX-ONLY Mode

When the SPI master only needs to receive, it can enter the RX-ONLY mode by setting the RXO register. At this time, the SPI module does not need software

By writing to the TX Buffer, data can be continuously received. At this time, MOSI will maintain the IDLE level and will not be set.

Bit TXBE interrupt flag register.

### **22.5.5** Host **SSN** Control

The SPI module host supports hardware or software control of the SSN signal.

When the SSNSEN register is cleared, SSN is controlled by the hardware circuit; if the SSNM register is set, the SPI will

Pull SSN high, and the SSN high time is configured by the WAIT register (several SCK clock cycles);

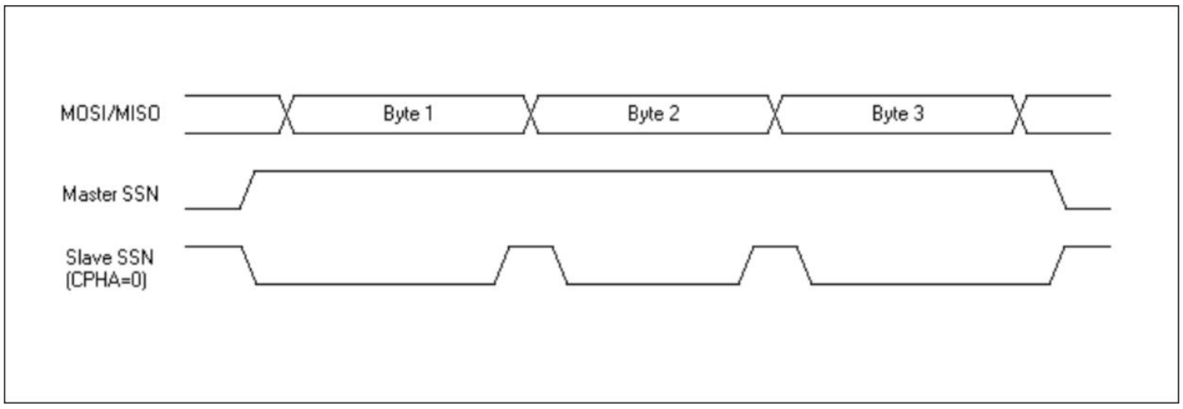

Figure **22-8 SPI SSN timing diagram (SSNM=1, CPHA=0)**

If the SSNM register is reset, the SPI will not pull up SSN after sending each frame of data, but will directly start sending the next frame of data.

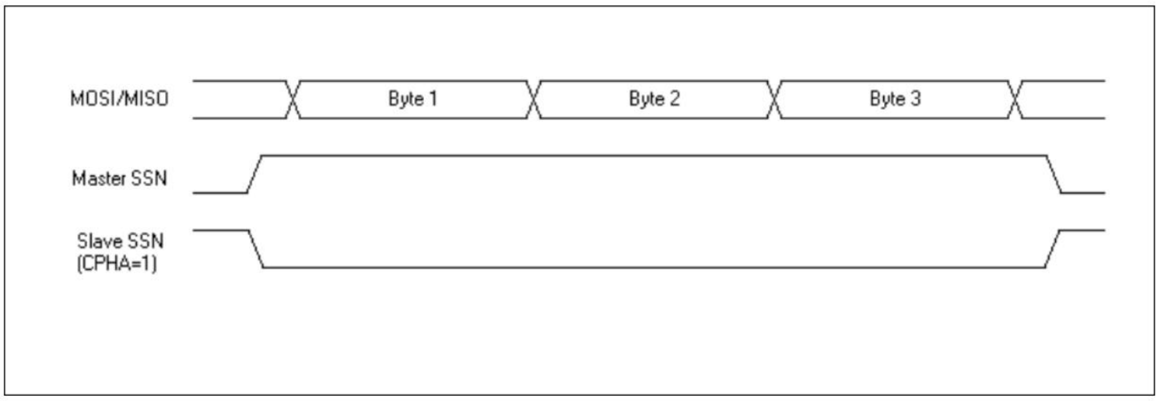

Figure **22-9 SPI SSN timing diagram (SSNM=0)**

When the SSNSEN register is set, SSN is directly controlled by software. Software can directly operate the SPIxCR2.SSN register bit by writing

As the SSN level sent by the SPI master.

### **22.5.6** Data Conflict

When the SPI TX Buffer data has not been read into the shift register, or the data in the SPI RX Buffer has not been read by software or DMA

When reading, a write operation to the TX Buffer or RX Buffer will generate a corresponding conflict error, and the TXCOL/RXCOL bit will be set.

Generates an interrupt. The write data that causes the conflict will be ignored. Data conflict errors will occur in both master and slave modes.

The write operation to the TX Buffer is initiated by the Master module inside the chip, including the CPU, DMA, etc.

Write operations are initiated by external SPI devices.

When a data conflict occurs, the original data in the TX Buffer and RX Buffer will not be refreshed, and the newly written data will be lost.

### **22.5.7** Using **DMA** for **SPI** Transmission and Reception

When the SPI module is enabled, the SPI module will automatically generate corresponding DMA requests when the transmit buffer is empty or the receive buffer is full.

The application software needs to configure the DMA channel connection in advance, point the specific channel to the SPI peripheral, set the pointer address for RAM access, and

Enable DMA channel. After that, DMA will automatically respond to SPI request and complete the data transfer between RAM and SPI.

Note: If you use *DMA* To perform full-duplex communication, the software should first enable *DMA* Send channel, then enable *DMA* The receiving

On the contrary, it may lead to *SPI* Send an extra byte *dummy* data.

**SPI** reception using **DMA**

ÿ Configure DMA channel 3 or 5 as SPI_RX

ÿ Set RAM pointer address, address increment and decrement, channel priority, transfer length and interrupt settings, etc.

ÿ Enable the corresponding DMA channel

ÿ Configure SPI module parameters

ÿ Enable the SPI module and wait for data reception

ÿ After receiving data, SPI automatically generates DMA request

ÿ DMA responds to the request, reads the SPI receive buffer register, and writes to the specified RAM address

ÿ When the DMA transfer of the specified length is completed, the DMA will ignore subsequent requests and generate a transfer completion interrupt. The software should handle the interrupt

And turn off SPI

ÿ If data is received before closing SPI, the software can clear RXBUF by writing RXBFC

**SPI** Transmit using **DMA**

The DMA sending process is similar to the receiving process described above. The main difference is that when the DMA transfer of the specified length is completed, the software cannot immediately close the

Close SPI, because the last frame of data is still being shifted and sent, so the software needs to query the BUSY flag until the shift transmission is completed.

Then turn off the SPI module.

Data frame length and **RAM** data organization

The SPI transmission frame length can be configured to 8, 16, 24, or 32 bits.

When the data frame length is 8 bits, DMA moves 1 byte each time, and 4 moves fill up one address in RAM. The word is stored in little endian:

RAM word: { data3ÿdata2ÿdata1ÿdata0 }

When the data frame length is 16 bits, DMA transfers 2 bytes each time, and two transfers fill up one address in RAM.

Storage:

RAM word: { data1ÿdata0 }

When the data frame length is 24 bits, DMA transfers 1 word at a time, and one transfer fills up one address in RAM, but the valid data is only

Occupies the lower 24 bits of the RAM word:

RAM word: { 8'h0ÿdata0 }

When the data frame length is 32 bits, DMA transfers 1 word at a time, and one transfer fills up one address in RAM:

RAM word: { data0 }

## 22.6 Registers

| **offset** address | name | symbol |
|---|---|---|
| | SPI1 register (module start address: 0x40018C00) | |
| 0x00000000 | SPI1 Control Register 1 ÿSPI1 Control Register1ÿ | SPI1_CR1 |
| 0x00000004 | SPI1 Control Register 2 ÿSPI1 Control Register2ÿ | SPI1_CR2 |
| 0x00000008 | SPI1 Control Register 3 ÿSPI1 Control Register3ÿ | SPI1_CR3 |
| 0x0000000C | SPI1 interrupt enable register ÿSPI1 Interrupt Enable Registerÿ | SPI1_IER |
| 0x00000010 | SPI1 Interrupt Status Register ÿSPI1 Status Registerÿ | SPI1_ISR |
| 0x00000014 | SPI1 transmit data buffer register ÿSPI1 Transmit Bufferÿ | SPI1_TXBUF |
| 0x00000018 | SPI1 receive data buffer register ÿSPI1 Receive Bufferÿ | SPI1_RXBUF |
| | SPI2 register (module start address: 0x40010800) | |
| 0x00000000 | SPI2 Control Register 1 ÿSPI2 Control Register1ÿ | SPI2_CR1 |
| 0x00000004 | SPI2 Control Register 2 ÿSPI2 Control Register2ÿ | SPI2_CR2 |
| 0x00000008 | SPI2 Control Register 3 ÿSPI2 Control Register3ÿ | SPI2_CR3 |
| 0x0000000C | SPI2 Interrupt Enable Register ÿSPI2 Interrupt Enable Registerÿ | SPI2_IER |
| 0x00000010 | SPI2 Interrupt Status Register ÿSPI2 Status Registerÿ | SPI2_ISR |
| 0x00000014 | SPI2 transmit data buffer register ÿSPI2 Transmit Bufferÿ | SPI2_TXBUF |
| 0x00000018 | SPI2 receive data buffer register ÿSPI2 Receive Bufferÿ | SPI2_RXBUF |

### 22.6.1 SPIx Control Register 1 (SPIx_CR1)

| name | SPIx_CR1(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000000 | | | | | | | |
| bit Bit31 | Bit name | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| bit | · | | | | | | | |
| | U-0 | | | | | | | |
| permission | Bit23 Bit | Bit22 | Bit21 | Bit20 | Bit19 | Bit18 | Bit17 | Bit16 |
| name | · | | | | | | | |
| bit | U-0 | | | | | | | |
| permission | Bit15 | Bit14 | Bit13 | Bit12 | Bit11 | Bit10 | Bit9 | Bit8 |
| Bit | · | | | | IOSWAP MSPA SSPA MM | | | |
| name | U-0 | | | | R/W-0 | R/W-0 | R/W-0 | R/W-1 |
| bit | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| | WAIT | | BAUD | | | LSBF CPOL CPHA | | |
| permission | Bit name Bit permission R/W-00 | | R/W-000 | | | R/W-0 | R/W-0 | R/W-0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:12 | - | RFU: Unimplemented, read as 0 |
| 11 | **IOSWAP** | MOSI and MISO pin swapping (IO swapping)<br>0: Default pin order<br>1: Swap the pin order |
| 10 | **MSPA** | Master Sampling Position Adjustment, Master to MISO signal<br>Sampling position adjustment to compensate for PCB routing delays during high-speed communication<br>1: Sampling point delayed by half SCK cycle<br>0: No adjustment |
| 9 | **SSPA** | Slave Sending Position Adjustment, Slave MISO Sending Position Adjustment<br>1: The first half of the SCK cycle is sent<br>0: No adjustment |
| 8 | **MM** | Master/Slave mode selection.<br>1: Master mode<br>0: Slave mode |
| 7:6 | **WAIT** | In Master mode, add at least (1+WAIT) SCK cycles after sending each frame.<br>Waiting time, and then transmit the next frame of data. If SSN is controlled by hardware, and<br>If SSNM=1, the hardware will automatically pull SSN high. |
| 5:3 | **BAUD** | Master mode baud rate configuration bits:<br>000ÿ fAPBCLK/2<br>001ÿ fAPBCLK/4<br>010ÿ fAPBCLK/8<br>011ÿ fAPBCLK/16<br>100ÿ fAPBCLK/32<br>101ÿ fAPBCLK/64<br>110ÿ fAPBCLK/128<br>111ÿ fAPBCLK/256<br>These bits cannot be modified while communication is in progress. |
| 2 | **LSBF** | Frame format (LSB First) 0:<br>Send MSB first<br>1: Send LSB first<br>Note: The value of this bit cannot be changed while communication is in progress. |
| 1 | **CPOL** | Clock Polarity Selection<br>1: Serial clock stops at high level<br>0: Serial clock stops at low level<br>Note: The value of this bit cannot be changed while communication is in progress<br>Note: The value of this bit cannot be changed when SSN is low. |
| 0 | **CPHA** | Clock Phase Selection<br>1: The second clock edge is the first capture edge<br>0: The first clock edge is the first capture edge<br>Note: The value of this bit cannot be changed while communication is in progress. |

## 22.6.2 SPIx Control Register 2 (SPIx_CR2)

| name | SPIx_CR2(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000004 | | | | | | | |
| Bit31 Bit name Bit | | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| | . | | | | | | | |
| permission | U-0 | | | | | | | |

| Bit 23 Bit name | | Bit22 | Bit21 | Bit20 | Bit19 | Bit18 | Bit17 | Bit16 |
|---|---|---|---|---|---|---|---|---|
| | - | | | | | | | |
| | U-0 | | | | | | | |
| Permission Bit 15 | | Bit14 | Bit13 | Bit12 | Bit11 | Bit10 | Bit9 | Bit8 |
| Position Name | DUMMY _EN | - | | | RXO | DLEN | | HALFDU PLEX |
| bit permission R/W-0 | | U-0 | | | R/W-0 | R/W-00 | | R/W-0 |
| bit | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| Bit Name HD_RW CMD8b SSNM TXO_AC TXO Bit Permission R/W-0 | | | | | | SSN SSNSEN SPIEN | | |
| | | R/W-1 | R/W-0 | R/W-1 | R/W-0 | R/W-1 | R/W-0 | R/W-0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:16 | - | RFU: Unimplemented, read as 0 |
| 15 | **DUMMY_EN** | Whether to insert a dummy cycle in the read operation in 4-wire half-duplex protocol<br>(Dummy cycle Enable)<br>0: Do not insert a dummy cycle<br>1: Insert a dummy cycle after the read command |
| 14:12 | - | RFU: Unimplemented, read as 0 |
| 11 | **RXO** | RXONLY control bit. When this register is set, SPI can receive continuously without software<br>Write TXBUF (Receive Only mode)<br>1: Start the single receiving mode of Master<br>0: Disable single-receive mode (full-duplex transmission and reception) |
| 10:9 | **DLEN** | Communication data word length configuration (Data Length)<br>00ÿ8bit<br>01ÿ16bit<br>10ÿ24bit<br>11ÿ32bit |
| 8 | **HALFDUPLEX** | Communication mode selection (Half-Duplex mode)<br>0: Standard SPI mode, 4-wire full-duplex<br>1: DCN mode, 4-wire half-duplex |
| 7 | **HD_RW** | Read/Write configuration for host read/write operation in half-duplex mode<br>Half-Duplex mode)<br>0: The master writes to the slave in 4-wire half-duplex protocol<br>1: Master reads slave data under 4-wire half-duplex protocol |
| 6 | **CMD8b** | Define the command frame length in half-duplex mode (Command 8 bits)<br>1: Command frame is fixed to 8 bits<br>0: Command frame length is defined by DLEN |
| 5 | **SSNM** | SSN control mode selection in Master mode (SSN mode)<br>1: After sending each frame, the Master pulls up SSN and maintains the high level time from WAIT<br>Register Control<br>0: Master keeps SSN low after sending each frame |
| 4 | **TXO_AC** | TXONLY hardware auto-clear enable (TXONLY auto-clear enable)<br>1: TXONLY hardware automatic clearing is effective, after software enables TXO, wait for sending<br>After completion, the hardware is cleared<br>0: Disable TXONLY, hardware automatically clears |
| 3 | **TXT** | TXONLY control bit (Transmit Only mode enable)<br>1: Start the single send mode of Master<br>0: Disable single-transmit mode (full-duplex transmission and reception) |
| 2 | **SSN** | In Master mode, if SSNSEN is 1, the software can control SSN through this bit<br>Output Level |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| | | 1: SSN output high level<br>0: SSN outputs low level |
| 1 | **SSNS** | In Master mode, software controls SSN enable (SSN Software Enable)<br>1: In Master mode, SSN output is controlled by software<br>0: In Master mode, SSN output is automatically controlled by hardware |
| 0 | **SPINE** | SPI enable<br>1: Enable SPI<br>0: Turn off SPI and clear the send and receive buffers |

## 22.6.3 SPIx Control Register 3 (SPIx_CR3)

| name | SPIx_CR3(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000008 | | | | | | | |
| Bit31 Bit | name bit | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| | - | | | | | | | |
| | U-0 | | | | | | | |
| permission | Bit23 Bit | Bit22 | Bit21 | Bit20 | Bit19 | Bit18 | Bit17 | Bit16 |
| name | - | | | | | | | |
| bit | U-0 | | | | | | | |
| permission | Bit15 | Bit14 | Bit13 | Bit12 | Bit11 | Bit10 | Bit9 | Bit8 |
| Bit | - | | | | | | | |
| name | U-0 | | | | | | | |
| bit | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| | - | | | | TXBFC | RXBFC | MERRC | SERRC |
| permission | Bit name bit permission U-0 | | | | W-0 | W-0 | W-0 | W-0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:4 | - | RFU: Unimplemented, read as 0 |
| 3 | **TXBFC** | Transmit Buffer Clear, software writes 1 to clear the transmit buffer, writes 0 to disable |
| 2 | **RXBFC** | Receive Buffer Clear, software writes 1 to clear the send buffer, writes 0 to have no effect |
| 1 | **MERRC** | Master Error Clear, software writes 1 to clear the HSPISTA.MERR register |
| 0 | **SERRC** | Slave Error Clear, software writes 1 to clear the HSPISTA.SERR register |

## 22.6.4 SPIx Interrupt Control Register (SPIx_IER)

| name | SPIx_IER(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x0000000C | | | | | | | |
| Bit31 Bit | name bit | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| | - | | | | | | | |
| | U-0 | | | | | | | |
| permission | Bit23 Bit | Bit22 | Bit21 | Bit20 | Bit19 | Bit18 | Bit17 | Bit16 |
| name | - | | | | | | | |
| bit | U-0 | | | | | | | |
| permission | Bit15 | Bit14 | Bit13 | Bit12 | Bit11 | Bit10 | Bit9 | Bit8 |
| Bit | - | | | | | | | |
| name | U-0 | | | | | | | |
| bit permission | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |

| Bit name | - | ERRIE | TXIE | RXIE |
|---|---|---|---|---|
| and permission | U-0 | R/W-0 | R/W-0 | R/W-0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:3 | - | RFU: Unimplemented, read as 0 |
| 2 | **ERRIE** | SPI Error Interrupt Enable |
| 1 | **LET'S GO** | Transmit Interrupt Enable |
| 0 | **RXIE** | Receive Interrupt Enable |

## 22.6.5 SPIx Interrupt Flag Register (SPIx_ISR)

| name | SPIx_ISR(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000010 | | | | | | | |
| Bit31 Bit | name bit | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| | - | | | | | | | |
| | U-0 | | | | | | | |
| permission | Bit23 Bit | Bit22 | Bit21 | Bit20 | Bit19 | Bit18 | Bit17 | Bit16 |
| name | - | | | | | | | |
| bit | U-0 | | | | | | | |
| permission | Bit15 Bit | Bit14 | Bit13 | Bit12 | Bit11 | Bit10 | Bit9 | Bit8 |
| name | - | | | DCN_TX | - | RXCOL | TXCOL | BUSY |
| bit | U-0 | | | R/W-1 | U-0 | R/W-0 | R/W-0 | R-0 |
| | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| | - | GET | SERR | - | | | TXBE | RXBF |
| permission | Bit name bit permission U-0 | R-0 | R-0 | U-0 | | | R-1 | R-0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:13 | - | RFU: Unimplemented, read as 0 |
| 12 | **DCN_TX** | In half-duplex mode (HALFDUPLEX=1), the configuration is at the end of each data frame.<br>bit DCN signal level sent (Data/Command transmit config)<br>0: DCN=0, indicating command frame<br>1: DCN=1, indicating data frame<br>Software should set the DCN_TX register before sending. If DCN_TX=0, hardware<br>After completing a frame transmission, DCN_TX is automatically set to 1, that is, only one frame is transmitted by default.<br>The first frame is a command frame, followed by data frames. |
| 11 | - | RFU: Unimplemented, read as 0 |
| 10 | **RXCOL** | Receive buffer overflow, software writes 1 to clear (Receive Collision flag, write 1 to flag) |
| 9 | **TXCOL** | Transmit buffer overflow, software writes 1 to clear (Transmit Collision flag, write 1 to clear) |
| 8 | **BUSY** | SPI idle flag, read-only (busy flag)<br>1: SPI transfer in progress<br>0: SPI transmission idle |
| 7 | - | RFU: Unimplemented, read as 0 |
| 6 | **GET** | Master Error ÿÿ(Master Error flag)<br>When the Master transmits less than 8 bits of SSN and is pulled high, MERR is set. |
| 5 | **HEAD** | Slave Error flag<br>When the Slave transmits less than 8 bits and SSN is pulled high, SERR is set. |
| 4:2 | - | RFU: Unimplemented, read as 0 |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 1 | **TXBE** | TX Buffer Empty flag<br>1: The transmit buffer is empty, and the software writes TXBUF to clear it.<br>0: Send buffer is full |
| 0 | **RXBF** | RX Buffer Full flag<br>1: The receiving buffer is full, and the software reads RXBUF to clear it.<br>0: Receive buffer empty |

## 22.6.6 SPIx Transmit Buffer Register (SPIx_TXBUF)

| name | SPIx_TXBUF(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000014 | | | | | | | |
| bit Bit31 | Bit name | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| bit | TXBUF[31:24] | | | | | | | |
| | W-0000 0000 | | | | | | | |
| permission | Bit23 Bit | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 |
| name | TXBUF[23:16] | | | | | | | |
| bit | W-0000 0000 | | | | | | | |
| permissi | on Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 |
| Bit | TXBUF[15:8] | | | | | | | |
| name | W-0000 0000 | | | | | | | |
| bit | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 |
| | TXBUF[7:0] | | | | | | | |
| permission | Bit name bit permission W-0000 0000 | | | | | | | |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:0 | **TXBUF** | SPI Transmit Buffer |

## 22.6.7 SPIx Receive Buffer Register (SPIx_RXBUF)

| name | SPIx_RXBUF(x=1,2) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Offset** | 0x00000018 | | | | | | | |
| Bit31 Bit | name bit | Bit30 | Bit29 | Bit28 | Bit27 | Bit26 | Bit25 | Bit24 |
| | RXBUF[31:24] | | | | | | | |
| | R-0000 0000 | | | | | | | |
| permission | Bit23 Bit | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 | Bit23 |
| name | RXBUF[23:16] | | | | | | | |
| bit | R-0000 0000 | | | | | | | |
| permissi | on Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 | Bit15 |
| Bit | RXBUF[15:8] | | | | | | | |
| name | R-0000 0000 | | | | | | | |
| bit | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 | Bit7 |
| | RXBUF[7:0] | | | | | | | |
| permission | Bit name bit permission R-0000 0000 | | | | | | | |

| Position No. | Mnemonics | Functional Description |
|---|---|---|
| 31:0 | **RXBUF** | SPI Receive Buffer |