SPI Controller (SPI)

30.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP32-S3 chip integrates four SPI controllers:

- SPI0,
- SPI1,
- General Purpose SPI2 (GP-SPI2),
- and General Purpose SPI3 (GP-SPI3).

SPI0 and SPI1 controllers are primarily reserved for internal use to communicate with external flash and PSRAM memory. This chapter mainly focuses on the GP-SPI controllers, i.e., GP-SPI2 and GP-SPI3. **In this chapter unless otherwise stated, GP-SPI refers to both GP-SPI2 and GP-SPI3.**

30.2 Glossary

To better illustrate the functions of GP-SPI, the following terms are used in this chapter.

Master Mode	GP-SPI acts as an SPI master and initiates SPI transactions.
Slave Mode	GP-SPI acts as an SPI slave and transfers data with its master when its CS is asserted.
MISO	Master in, slave out, data transmission from a slave to a master.
MOSI	Master out, slave in, data transmission from a master to a slave.
Transaction	One instance of a master asserting a CS line, transferring data to and from a slave, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
SPI Transfer	The whole process of an SPI master exchanges data with a slave. One SPI transfer consists of one or more SPI transactions.
Single Transfer	An SPI transfer consists of only one transaction.
CPU-Controlled Transfer	A data transfer happens between CPU configured buffer SPI_W0_REG ~ SPI_W15_REG and SPI peripheral.
DMA-Controlled Transfer	A data transfer happens between DMA and SPI peripheral, controlled by DMA engine.
Configurable Segmented Transfer	A data transfer controlled by DMA in SPI master mode. Such transfer consists of multiple transactions (segments), and each of transactions can be configured independently.

Slave Segmented Transfer	A data transfer controlled by DMA in SPI slave mode. Such transfer consists of multiple transactions (segments).
Full-duplex	The sending line and receiving line between the master and the slave are independent. Sending data and receiving data happen at the same time.
Half-duplex	Only one side, the master or the slave, sends data first, and the other side receives data. Sending data and receiving data can not happen at the same time.
4-line full-duplex	4-line here means: clock line, CS line, and two data lines. The two data lines can be used to send or receive data simultaneously.
4-line half-duplex	4-line here means: clock line, CS line, and two data lines. The two data lines can not be used simultaneously.
3-line half-duplex	3-line here means: clock line, CS line, and one data line. The data line is used to transmit or receive data.
1-bit SPI	In one clock cycle, one bit can be transferred.
(2-bit) Dual SPI	In one clock cycle, two bits can be transferred.
Dual Output Read	A data mode of Dual SPI. In one clock cycle, one bit of a command, or one bit of an address, or two bits of data can be transferred.
Dual I/O Read	Another data mode of Dual SPI. In one clock cycle, one bit of a command, or two bits of an address, or two bits of data can be transferred.
(4-bit) Quad SPI	In one clock cycle, four bits can be transferred.
Quad Output Read	A data mode of Quad SPI. In one clock cycle, one bit of a command, or one bit of an address, or four bits of data can be transferred.
Quad I/O Read	Another data mode of Quad SPI. In one clock cycle, one bit of a command, or four bits of an address, or four bits of data can be transferred.
QPI	In one clock cycle, four bits of a command, or four bits of an address, or four bits of data can be transferred.
(8-bit) Octal SPI	In one clock cycle, eight bits can be transferred.
Octal Output Read	A data mode of Octal SPI. In one clock cycle, one bit of a command, or one bit of an address, or eight bits of data can be transferred.
Octal I/O Read	Another data mode of Octal SPI. In one clock cycle, one bit of a command, or eight bits of an address, or eight bits of data can be transferred.
OPI	In one clock cycle, eight bits of a command, or eight bits of an address, or eight bits of data can be transferred.
FSPI	Fast SPI. The prefix of the signals for GP-SPI2. FSPI bus signals are routed to GPIO pins via either GPIO matrix or IO MUX.
SPI3	The prefix of the signals for GP-SPI3. SPI3 bus signals are routed to GPIO pins via GPIO matrix only.

30.3 Features

Some of the key features of GP-SPI are:

- Master and slave modes
- Half- and full-duplex communications
- CPU- and DMA-controlled transfers
- Various data modes:
  - **GP-SPI2**:
    * 1-bit SPI mode
    * 2-bit Dual SPI mode
    * 4-bit Quad SPI mode
    * QPI mode
    * 8-bit Octal SPI mode
    * OPI mode
  - **GP-SPI3**:
    * 1-bit SPI mode
    * 2-bit Dual SPI mode
    * 4-bit Quad SPI mode
    * QPI mode
- Configurable module clock frequency:
  - Master: up to 80 MHz
  - Slave: up to 60 MHz
- Configurable data length:
  - CPU-controlled transfer in master mode or in slave mode: 1 ~ 64 B
  - DMA-controlled single transfer in master mode: 1 ~ 32 KB
  - DMA-controlled configurable segmented transfer in master mode: data length is unlimited
  - DMA-controlled single transfer or segmented transfer in slave mode: data length is unlimited
- Configurable bit read/write order
- Independent interrupts for CPU-controlled transfer and DMA-controlled transfer
- Configurable clock polarity and phase
- Four SPI clock modes: mode 0 ~ mode 3
- Multiple CS lines in master mode:
  - **GP-SPI2**: CS0 ~ CS5
  - **GP-SPI3**: CS0 ~ CS2

- Able to communicate with SPI devices, such as a sensor, a screen controller, as well as a flash or RAM chip

30.4 Architectural Overview

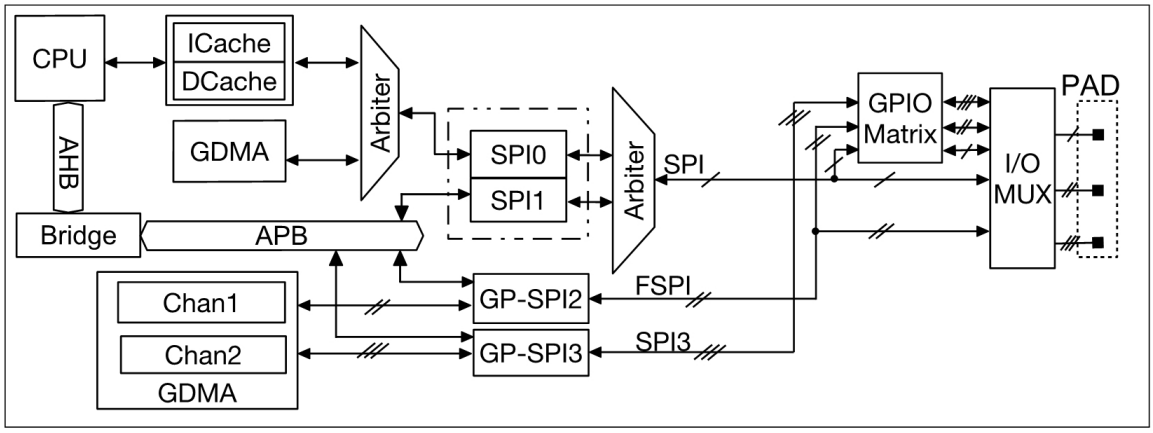


Figure 30.4-1. SPI Module Overview

Figure 30.4-1 shows an overview of SPI module. GP-SPI2 and GP-SPI3 exchange data with SPI devices in the following ways:

- CPU-controlled transfer: CPU <-> GP-SPI2 (GP-SPI3) <-> SPI devices
- DMA-controlled transfer: GDMA <-> GP-SPI2 (GP-SPI3) <-> SPI devices

The signals for GP-SPI2 and GP-SPI3 are prefixed with "FSPI" (Fast SPI) and "SPI3", respectively. FSPI bus signals are routed to GPIO pins via either GPIO matrix or IO MUX. SPI3 bus signals are routed to GPIO pins via GPIO matrix only. For more information, see Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

The functionalities of GP-SPI3 are nearly the same as those of GP-SPI2. GP-SPI2's functionalities are described in Section 30.5. The differences between GP-SPI2 and GP-SPI3 are described in Section 30.5.1 and Section 30.9.

30.5 Functional Description

30.5.1 Data Modes

GP-SPI can be configured as either a master or a slave to communicate with other SPI devices in the following data modes, see Table 30.5-1. As a GP-SPI master, the data modes listed in this table are defined in Section 30.5.8; as a GP-SPI slave, the data modes listed in this table are defined in Section 30.5.9.

Table 30.5-1. Data Modes Supported by GP-SPI2 and GP-SPI3

Supported Mode		CMD Phase	Address Phase	Data Phase	GP-SPI2	GP-SPI3
1-bit SPI		1-bit	1-bit	1-bit	Y	Y
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit	Y	Y
	Dual I/O Read	1-bit	2-bit	2-bit	Y	Y

Quad SPI	Quad Output Read	1-bit	1-bit	4-bit	Y	Y
	Quad I/O Read	1-bit	4-bit	4-bit	Y	Y
Octal SPI	Octal Output Read	1-bit	1-bit	8-bit	Y	—
	Octal I/O Read	1-bit	8-bit	8-bit	Y	—
QPI		4-bit	4-bit	4-bit	Y	Y
OPI		8-bit	8-bit	8-bit	Y	—

30.5.2 Introduction to FSPI Bus and SPI3 Bus Signals

Functional description of FSPI/SPI3 bus signals is shown in Table 30.5-2. Table 30.5-3 and Table 30.5-4 list the signals used in various SPI modes.

Table 30.5-2. Functional Description of FSPI/SPI3 Bus Signals

FSPI Bus Signal	SPI3 Bus Signal	Function
FSPICLK	SPI3_CLK	Input and output clock in master/slave mode
FSPICS0	SPI3_CS0	Input and output CS signal in master/slave mode
FSPICS1 ~ 5	SPI3_CS1 ~ 2	Output CS signal in master mode
FSPID	SPI3_D	MOSI/SIO0 (serial data input and output, bit0)
FSPIQ	SPI3_Q	MISO/SIO1 (serial data input and output, bit1)
FSPIWP	SPI3_WP	SIO2 (serial data input and output, bit2)
FSPIHD	SPI3_HD	SIO3 (serial data input and output, bit3)
FSPIIO4 ~ 7	—	SIO4 ~ 7 (serial data input and output, bit4 ~ 7)
FSPIDQS	—	Output data mask signal in master mode

Table 30.5-3. FSPI bus Signals Used in Various SPI Modes

FSPI Signal	Master Mode								Slave Mode					
	1-bit SPI			Dual SPI	Quad SPI	QPI	Octal SPI	OPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD						FD	3-line HD	4-line HD			
FSPICLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS1	Y	Y	Y	Y	Y	Y	Y	Y						
FSPICS2	Y	Y	Y	Y	Y	Y	Y	Y						
FSPICS3	Y	Y	Y	Y	Y	Y	Y	Y						
FSPICS4	Y	Y	Y	Y	Y	Y	Y	Y						
FSPICS5	Y	Y	Y	Y	Y	Y	Y	Y						
FSPID	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
FSPIQ	Y		(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
FSPIWP					Y[5]	Y	Y	Y					Y[8]	Y
FSPIHD					Y[5]	Y	Y	Y					Y[8]	Y
FSPII04 ~ 7							Y	Y						
FSPIDQS							Y	Y						

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

[4] The two signals are used in parallel.

[5] The four signals are used in parallel.

[6] Only one of the two signals is used at a time.

[7] The two signals are used in parallel.

[8] The four signals are used in parallel.

Table 30.5-4. SPI3 bus Signals Used in Various SPI Modes

SPI3 Signal	Master Mode						Slave Mode					
	1-bit SPI			Dual SPI	Quad SPI	QPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
SPI3_CLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS1	Y	Y	Y	Y	Y	Y						
SPI3_CS2	Y	Y	Y	Y	Y	Y						
SPI3_D	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
SPI3_Q	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
SPI3_WP					Y[5]	Y					Y[8]	Y
SPI3_HD					Y[5]	Y					Y[8]	Y

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

[4] The two signals are used in parallel.

[5] The four signals are used in parallel.

[6] Only one of the two signals is used at a time.

[7] The two signals are used in parallel.

[8] The four signals are used in parallel.

30.5.3 Bit Read/Write Order Control

In master mode:

- The bit order of the command, address and data sent by the GP-SPI master is controlled by SPI_WR_BIT_ORDER.
- The bit order of the data received by the master is controlled by SPI_RD_BIT_ORDER.

In slave mode:

- The bit order of the data sent by the GP-SPI slave is controlled by SPI_WR_BIT_ORDER.
- The bit order of the command, address and data received by the slave is controlled by SPI_RD_BIT_ORDER.

Table 30.5-5 shows the function of SPI_RD/WR_BIT_ORDER. In Table 30.5-5, FSPI Bus signals are used for description. The bit order of SPI3 Bus signals can be referred to Table 30.5-5.

Table 30.5-5. Bit Order Control in GP-SPI Master and Slave Modes

Bit Mode	FSPI Bus Signal	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit mode	FSPID or FSPIQ	B7→B6→B5→B4→B3→B2→B1→B0	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7	B0→B1→B2→B3→B4→B5→B6→B7
2-bit mode	FSPIQ	B7→B5→B3→B1	B6→B4→B2→B0	B1→B3→B5→B7	B0→B2→B4→B6
	FSPID	B6→B4→B2→B0	B7→B5→B3→B1	B0→B2→B4→B6	B1→B3→B5→B7
4-bit mode	FSPIHD	B7→B3	B4→B0	B3→B7	B0→B4
	FSPIWP	B6→B2	B5→B1	B2→B6	B1→B5
	FSPIQ	B5→B1	B6→B2	B1→B5	B2→B6
	FSPID	B4→B0	B7→B3	B0→B4	B3→B7
8-bit mode	FSPIO7	B7	B7	B0	B0
	FSPIO6	B6	B6	B1	B1
	FSPIO5	B5	B5	B2	B2
	FSPIO4	B4	B4	B3	B3
	FSPIHD	B3	B3	B4	B4
	FSPIWP	B2	B2	B5	B5
	FSPIQ	B1	B1	B6	B6
	FSPID	B0	B0	B7	B7

30.5.4 Transfer Modes

GP-SPI supports the following transfers when working as a master or a slave.

Table 30.5-6. Supported Transfers in Master and Slave Modes

Mode		CPU-Controlled Single Transfer	DMA-Controlled Single Transfer	DMA-Controlled Configurable Segmented Transfer*	DMA-Controlled Slave Segmented Transfer
Master	Full-Duplex	Y	Y	Y	—
	Half-Duplex	Y	Y	Y	—
Slave	Full-Duplex	Y	Y	–	Y
	Half-Duplex	Y	Y	—	Y

* DMA-Controlled Configurable Segmented Transfer is not supported on GP-SPI3.

The following sections provide detailed information about the transfer modes listed in the table above.

30.5.5 CPU-Controlled Data Transfer

GP-SPI provides 16 x 32-bit data buffers, i.e., SPI_W0_REG ~ SPI_W15_REG, see Figure 30.5-1. CPU-controlled transfer indicates the transfer, in which the data to send is from GP-SPI data buffer and the received data is stored to GP-SPI data buffer. In such transfer, every single transaction needs to be triggered by the CPU, after its related registers are configured. For such reason, the CPU-controlled transfer is always single transfers (consisting of only one transaction). CPU-controlled transfer supports full-duplex communication and half-duplex communication.

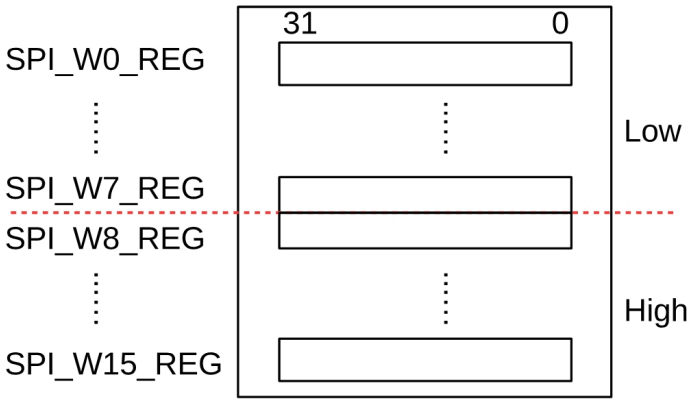


Figure 30.5-1. Data Buffer Used in CPU-Controlled Transfer

30.5.5.1 CPU-Controlled Master Mode

In a CPU-controlled master full-duplex or half-duplex transfer, the RX or TX data is saved to or sent from SPI_W0_REG ~ SPI_W15_REG. The bits SPI_USR_MOSI_HIGHPART and SPI_USR_MISO_HIGHPART control which buffers are used, see the list below.

- TX data
  - When SPI_USR_MOSI_HIGHPART is cleared, i.e., high part mode is disabled, TX data is from SPI_W0_REG ~ SPI_W15_REG and the data address is incremented by 1 on each byte transferred. If

the data byte length is larger than 64, the data in SPI_W0_REG ~ SPI_W15_REG may be sent more than once. For instance, 66 bytes (byte0 ~ byte65) need to send out, the address of byte65 is the result of (65 % 64 = 1), i.e., byte65 is from SPI_W0_REG[15:8], and byte64 is from SPI_W0_REG[7:0]. For this case, the content of SPI_W0_REG[15:0] may be sent more than once.

  - When SPI_USR_MOSI_HIGHPART is set, i.e., high part mode is enabled, TX data is from SPI_W8_REG ~ SPI_W15_REG and the data address is incremented by 1 on each byte transferred. If the data byte length is larger than 32, the data in SPI_W8_REG ~ SPI_W15_REG may be sent more than once.

- RX data

  - When SPI_USR_MISO_HIGHPART is cleared, i.e., high part mode is disabled, RX data is saved to SPI_W0_REG ~ SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. If the data byte length is larger than 64, the data in SPI_W0_REG ~ SPI_W15_REG may be overwritten. For instance, 66 bytes (byte0 ~ byte65) are received, byte65 and byte64 will be stored to the addresses of (65 % 64 = 1) and (64 % 64 = 0), i.e., SPI_W0_REG[15:8] and SPI_W0_REG[7:0]. For this case, the content of SPI_W0_REG[15:0] may be overwritten.

  - When SPI_USR_MISO_HIGHPART is set, i.e., high part mode is enabled, the RX data is saved to SPI_W8_REG ~ SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. If the data byte length is larger than 32, the content of SPI_W8_REG ~ SPI_W15_REG may be overwritten.

Note:

- TX/RX data address mentioned above both are byte-addressable. Address 0 stands for SPI_W0_REG[7:0], and Address 1 for SPI_W0_REG[15:8], and so on. The largest address is SPI_W15_REG[31:24].
- To avoid any possible error in TX/RX data, such as TX data being sent more than once or RX data being over-written, please make sure the registers are configured correctly.

30.5.5.2 CPU-Controlled Slave Mode

In a CPU-controlled slave full-duplex or half-duplex transfer, the RX data or TX data is saved to or sent from SPI_W0_REG ~ SPI_W15_REG, which are byte-addressable.

- In full-duplex communication, the address of SPI_W0_REG ~ SPI_W15_REG starts from 0 and is incremented by 1 on each byte transferred. If the data address is larger than 63, the content of SPI_W15_REG[31:24] is overwritten.
- In half-duplex communication, the ADDR value in transmission format is the start address of the RX or TX data, corresponding to the registers SPI_W0_REG ~ SPI_W15_REG. The RX or TX address is incremented by 1 on each byte transferred. If the address is larger than 63 (the highest byte address, i.e., SPI_W15_REG[31:24]), the address of overflowing data is always 63 and only the content of SPI_W15_REG[31:24] is overwritten.

According to your applications, the registers SPI_W0_REG ~ SPI_W15_REG can be used as:

- data buffers only
- data buffers and status buffers

- status buffers only

30.5.6 DMA-Controlled Data Transfer

DMA-controlled transfer refers to the transfer, in which GDMA RX module receives data and GDMA TX module sends data. This transfer is supported both in master mode and in slave mode.

A DMA-controlled transfer can be

- a single transfer, consisting of only one transaction. GP-SPI supports this transfer both in master and slave modes.
- a configurable segmented transfer, consisting of several transactions (segments). Only GP-SPI2 supports this transfer in master mode. For more information, see Section 30.5.8.5.
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI supports this transfer only in slave mode. For more information, see Section 30.5.9.3.

A DMA-controlled transfer only needs to be triggered once by CPU. When such transfer is triggered, data is transferred by the GDMA engine from or to the DMA-linked memory, without CPU operation.

DMA-controlled transfer supports full-duplex communication, half-duplex communication and functions described in Section 30.5.8 and Section 30.5.9. Meanwhile, the GDMA RX module is independent from the GDMA TX module, which means that there are four kinds of full-duplex communications:

- Data is received in DMA-controlled mode and sent in DMA-controlled mode.
- Data is received in DMA-controlled mode but sent in CPU-controlled mode.
- Data is received in CPU-controlled mode but sent in DMA-controlled mode.
- Data is received in CPU-controlled mode and sent in CPU-controlled mode.

30.5.6.1 GDMA Configuration

- Select a GDMA channel*n*, and configure a GDMA TX/RX descriptor, see Chapter 3 *GDMA Controller (GDMA)*.
- Set the bit GDMA_INLINK_START_CH*n*/GDMA_OUTLINK_START_CH*n* to start GDMA RX/TX engine.
- Before all the GDMA TX buffer is used or the GDMA TX engine is reset, if GDMA_OUTLINK_RESTART_CH*n* is set, a new TX buffer will be added to the end of the last TX buffer in use.
- GDMA RX buffer is linked in the same way as the GDMA TX buffer, by setting GDMA_INLINK_START_CH*n* or GDMA_INLINK_RESTART_CH*n*.
- The TX and RX data lengths are determined by the configured GDMA TX and RX buffer respectively, both of which can be unlimited.
- Initialize GDMA inlink and outlink before GDMA starts. The bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA in register SPI_DMA_CONF_REG should be set, otherwise the read/write data will be stored to/sent from the registers SPI_WO_REG ~ SPI_W15_REG.

In master mode, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one single transfer or one configurable segmented transfer is finished.

In slave mode, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one of conditions listed in Table 30.5-7 are met.

Table 30.5-7. Interrupt Trigger Condition on GP-SPI Data Transfer in Slave Mode

Transfer Type	Control Bit[1]	Control Bit[2]	Condition
Slave Single Transfer	0	0	A single transfer is done.
	1	0	A single transfer is done. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1)
Slave Segmented Transfer	0	1	(CMD7 or End_SEG_TRANS) is received correctly.
	1	1	(CMD7 or End_SEG_TRANS) is received correctly. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1)

[1] SPI_RX_EOF_EN

[2] SPI_DMA_SLV_SEG_TRANS_EN

30.5.6.2 GDMA TX/RX Buffer Length Control

It is recommended that the length of configured GDMA TX/RX buffer is equal to the length of real transferred data.

- If the length of configured GDMA TX buffer is shorter than that of real transferred data, the extra data will be the same as the last transferred data. SPI_OUTFIFO_EMPTY_ERR_INT and GDMA_OUT_EOF_CH*n*_INT are triggered.
- If the length of configured GDMA TX buffer is longer than that of real transferred data, the TX buffer is not fully used, and the remaining buffer is available for following transaction even if a new TX buffer is linked later. Please keep it in mind. Or save the unused data and reset DMA.
- If the length of configured GDMA RX buffer is shorter than that of real transferred data, the extra data will be lost. The interrupts SPI_INFIFO_FULL_ERR_INT and SPI_TRANS_DONE_INT are triggered. But GDMA_IN_SUC_EOF_CH*n*_INT interrupt is not generated.
- If the length of configured GDMA RX buffer is longer than that of real transferred data, the RX buffer is not fully used, and the remaining buffer is discarded. In the following transaction, a new linked buffer will be used directly.

30.5.7 Data Flow Control in GP-SPI Master and Slave Modes

CPU-controlled and DMA-controlled transfers are supported in GP-SPI master and slave modes. CPU-controlled transfer means that data transfers between registers SPI_W0_REG ~ SPI_W15_REG and the SPI device. DMA-controlled transfer means that data transfers between the configured GDMA TX/RX buffer and the SPI device. To select between the two transfer modes, configure SPI_DMA_RX_ENA and SPI_DMA_TX_ENA before the transfer starts.

30.5.7.1 GP-SPI Functional Blocks

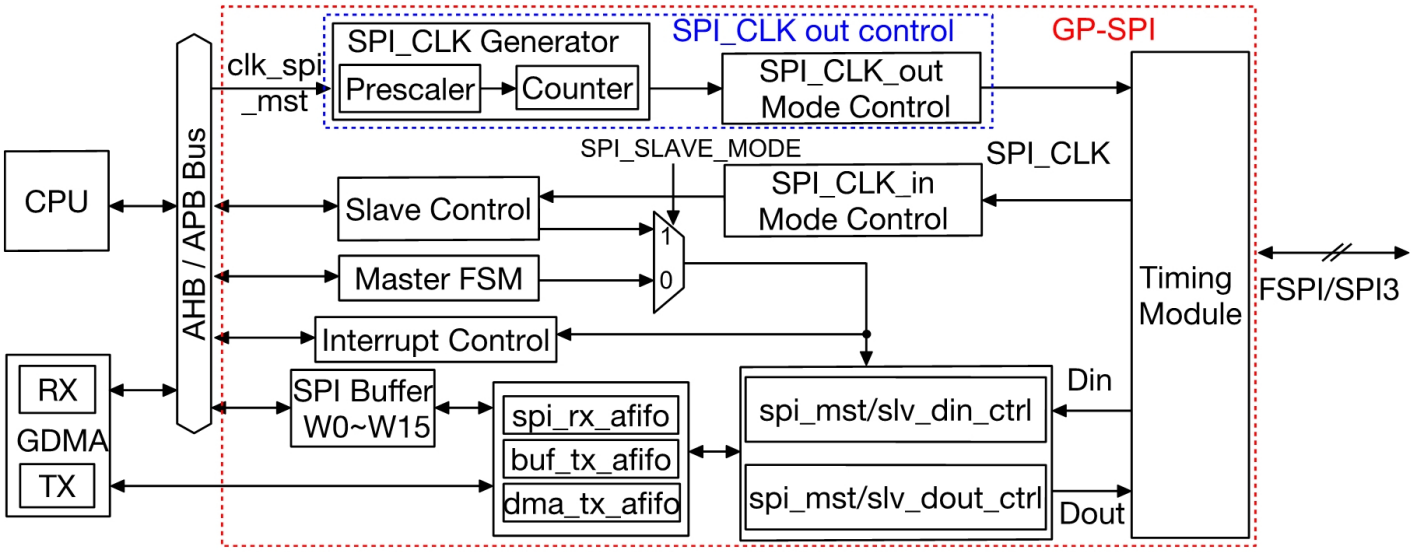


Figure 30.5-2. GP-SPI Block Diagram

Figure 30.5-2 shows main functional blocks in GP-SPI, including:

- Master FSM: all the features, supported in GP-SPI master mode, are controlled by this state machine together with register configuration.
- SPI Buffer: SPI_W0_REG ~ SPI_W15_REG, see Figure 30.5-1. The data transferred in CPU-controlled mode is prepared in this buffer.
- Timing Module: capture data on FSPI/SPI3 bus.
- spi_mst/slv_din/dout_ctrl: convert the TX/RX data into bytes.
- spi_rx_afifo: store the received data.
- buf_tx_afifo: store the data to send.
- dma_tx_afifo: store the data from GDMA.
- clk_spi_mst: this clock is the module clock of GP-SPI and derived from PLL_CLK. It is used in GP-SPI master mode, to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK Generator: generate SPI_CLK by dividing clk_spi_mst. The divider is determined by SPI_CLKCNT_N and SPI_CLKDIV_PRE.
- SPI_CLK_out Mode Control: output the SPI_CLK signal for data transfer and for slaves.
- SPI_CLK_in Mode Control: capture the SPI_CLK signal from SPI master when GP-SPI works as a slave.

30.5.7.2 Data Flow Control in Master Mode

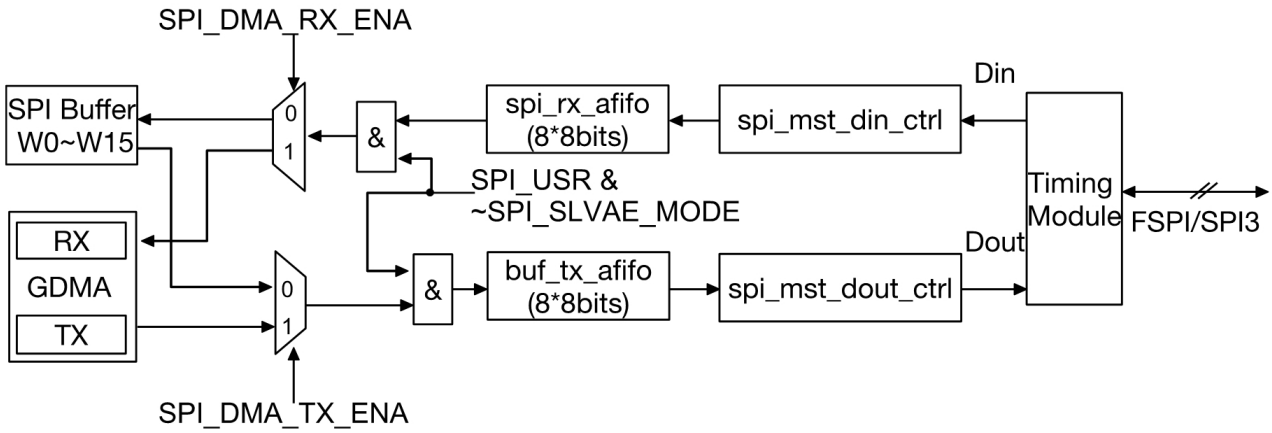


Figure 30.5-3. Data Flow Control in GP-SPI Master Mode

Figure 30.5-3 shows the data flow of GP-SPI in master mode. Its control logic is as follows:

- RX data: data in FSPI/SPI3 bus is captured by Timing Module, converted in units of bytes by spi_mst_din_ctrl module, then buffered in spi_rx_afifo, and finally stored in corresponding addresses according to the transfer modes.
  - CPU-controlled transfer: the data is stored to registers SPI_W0_REG ~ SPI_W15_REG.
  - DMA-controlled transfer: the data is stored to GDMA RX buffer.
- TX data: the TX data is from corresponding addresses according to transfer modes and is saved to buf_tx_afifo.
  - CPU-controlled transfer: TX data is from SPI_W0_REG ~ SPI_W15_REG.
  - DMA-controlled transfer: TX data is from GDMA TX buffer.

The data in buf_tx_afifo is sent out to Timing Module in 1/2/4/8-bit modes, controlled by GP-SPI state machine. The Timing Module can be used for timing compensation. For more information, see Section 30.8.

30.5.7.3 Data Flow Control in Slave Mode

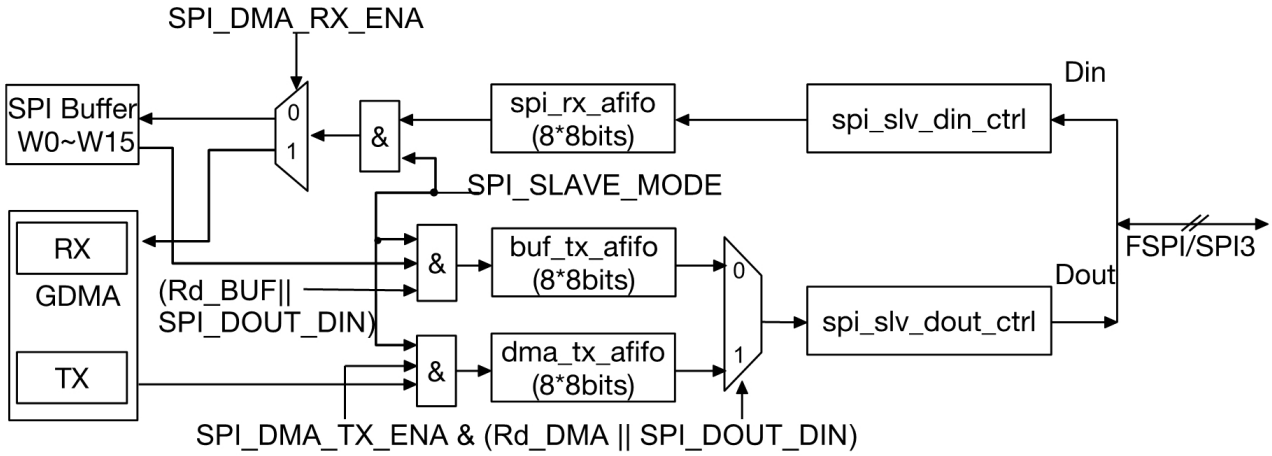


Figure 30.5-4. Data Flow Control in GP-SPI Slave Mode

Figure 30.5-4 shows the data flow in GP-SPI slave mode. Its control logic is as follows:

- In CPU/DMA-controlled full-duplex/half-duplex modes, when an external SPI master starts the SPI transfer, data on the FSPI/SPI3 bus is captured, converted into unit of bytes by spi_slv_din_ctrl module, and then is stored in spi_rx_afifo.
  - In CPU-controlled full-duplex transfer, the received data in spi_rx_afifo will be later stored into registers SPI_W0_REG ~ SPI_W15_REG, successively.
  - In half-duplex Wr_BUF transfer, when the value of address (SLV_ADDR[7:0]) is received, the received data in spi_rx_afifo will be stored in the related address of registers SPI_W0_REG ~ SPI_W15_REG.
  - In DMA-controlled full-duplex transfer or in half-duplex Wr_DMA transfer, the received data in spi_rx_afifo will be stored in the configured GDMA RX buffer.
- In CPU-controlled full-/half-duplex transfer, the data to send is stored in buf_tx_afifo. In DMA-controlled full-/half-duplex transfer, the data to send is stored in dma_tx_afifo. Therefore, Rd_BUF transaction controlled by CPU and Rd_DMA transaction controlled by DMA can be done in one slave segmented transfer. TX data comes from corresponding addresses according to the transfer modes.
  - In CPU-controlled full-duplex transfer, when SPI_SLAVE_MODE and SPI_DOUTDIN are set and SPI_DMA
    _TX_ENA is cleared, the data in SPI_W0_REG ~ SPI_W15_REG will be stored into buf_tx_afifo.
  - In CPU-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, Rd_BUF command and SLV_ADDR[7:0] are received, the data started from the related address of SPI_W0_REG ~ SPI_W15_REG will be stored into buf_tx_afifo.
  - In DMA-controlled full-duplex transfer, when SPI_SLAVE_MODE, SPI_DOUTDIN and SPI_DMA_TX_ENA are set, the data in the configured GDMA TX buffer will be stored into dma_tx_afifo.
  - In DMA-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, and Rd_DMA command is received, the data in the configured GDMA TX buffer will be stored into dma_tx_afifo.

  The data in buf_tx_afifo or dma_tx_afifo is sent out by spi_slv_dout_ctrl module in 1/2/4-bit modes.

30.5.8 GP-SPI Works as a Master

GP-SPI can be configured as a SPI master by clearing the bit SPI_SLAVE_MODE in SPI_SLAVE_REG. In this operation mode, GP-SPI provides clock signal (the divided clock from GP-SPI module clock) and six CS lines (CS0 ~ CS5).

Note:

- The length of transferred data must be in unit of bytes, otherwise the extra bits will be lost. The extra bits here means the result of total data bits % 8.
- To transfer bits not in unit of bytes, consider implementing it in CMD state or ADDR state.

30.5.8.1 State Machine

When GP-SPI works as a master, the state machine controls its various states during data transfer, including configuration (CONF), preparation (PREP), command (CMD), address (ADDR), dummy (DUMMY), data out (DOUT), and data in (DIN) states. GP-SPI is mainly used to access 1/2/4/8-bit SPI devices, such as flash and external RAM, thus the naming of GP-SPI states keeps consistent with the sequence naming of flash and external RAM. The meaning of each state is described as follows and Figure 30.5-5 shows the workflow of GP-SPI state machine.

1. IDLE: GP-SPI is not active or is in slave mode.
2. CONF: only used in DMA-controlled configurable segmented transfer (valid only for GP-SPI2). Set SPI_USR and SPI_USR_CONF to enable this state. If this state is not enabled, it means the current transfer is a single transfer.
3. PREP: prepare an SPI transaction and control SPI CS setup time. Set SPI_USR and SPI_CS_SETUP to enable this state.
4. CMD: send command sequence. Set SPI_USR and SPI_USR_COMMAND to enable this state.
5. ADDR: send address sequence. Set SPI_USR and SPI_USR_ADDR to enable this state.
6. DUMMY (wait cycle): send dummy sequence. Set SPI_USR and SPI_USR_DUMMY to enable this state.
7. DATA: transfer data.
    - DOUT: send data sequence. Set SPI_USR and SPI_USR_MOSI to enable this state.
    - DIN: receive data sequence. Set SPI_USR and SPI_USR_MISO to enable this state.
8. DONE: control SPI CS hold time. Set SPI_USR to enable this state.

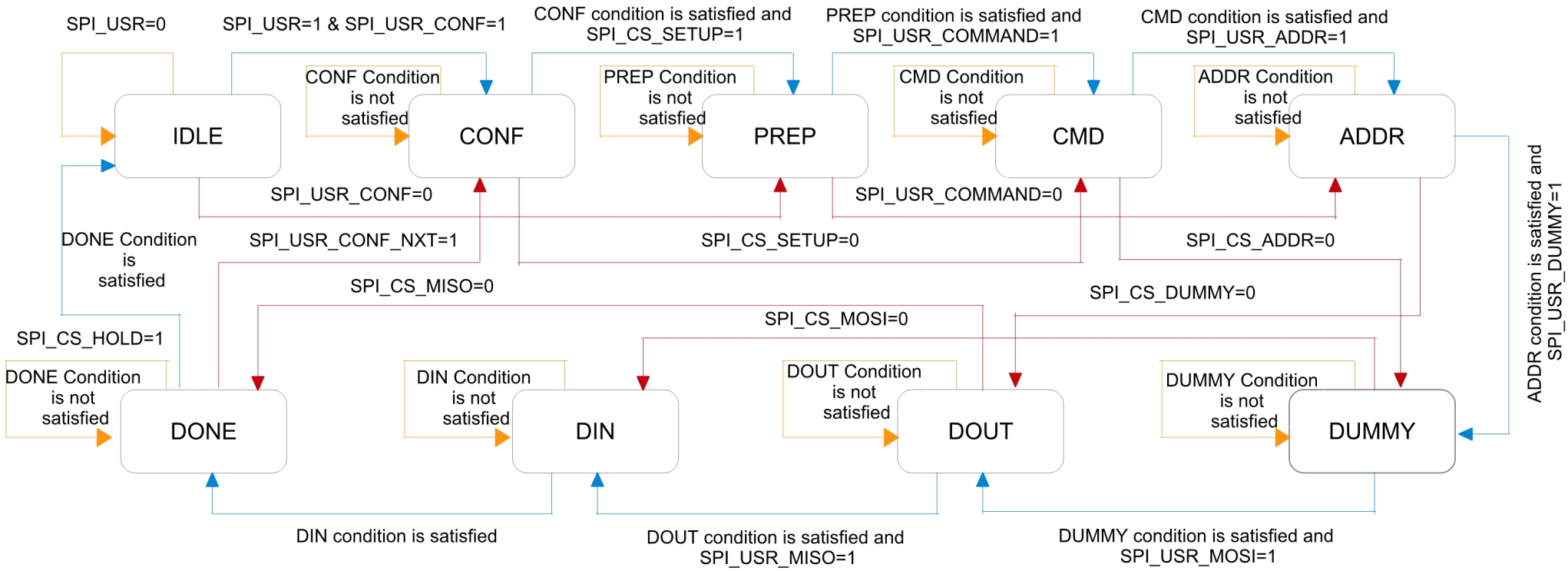


Figure 30.5-5. GP-SPI State Machine in Master Mode

Legend to state flow:

- —: indicates corresponding state condition is not satisfied; repeats current state.
- —: corresponding registers are set and conditions are satisfied; goes to next state.
- —: state registers are not set; skips one or more following states, depending on the registers of the following states are set or not.

Explanation of the conditions listed in the figure above:

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

A counter (gpc[17:0]) is used in the state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently. The cycle length of each state can also be configured independently.

30.5.8.2 Register Configuration for State and Bit Mode Control

Introduction

The registers, related to GP-SPI state control, are listed in Table 30.5-8. Users can enable QPI mode for GP-SPI by setting the bit SPI_QPI_MODE in register SPI_USER_REG.

Table 30.5-8. Registers Used for State Control in 1/2/4/8-bit Modes

State	Control Registers for 1-bit Mode FSPI/SPI3 Bus	Control Registers for 2-bit Mode FSPI/SPI3 Bus	Control Registers for 4-bit Mode FSPI/SPI3 Bus	Control Registers for 8-bit Mode FSPI/SPI3 Bus
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_OCT SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_OCT
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_OCT
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_OCT

As shown in Table 30.5-8, the registers in each cell should be configured to set the FSPI/SPI3 bus to corresponding bit mode, i.e., the mode shown in the table header, at a specific state (corresponding to the first column).

Configuration

For instance, when GP-SPI reads data, and

- CMD is in 1-bit mode
- ADDR is in 2-bit mode
- DUMMY is 8 clock cycles
- DIN is in 4-bit mode

The register configuration can be as follows:

1. Configure CMD state related registers.
    - Configure the required command value in SPI_USR_COMMAND_VALUE.
    - Configure command bit length in SPI_USR_COMMAND_BITLEN. SPI_USR_COMMAND_BITLEN = expected bit length - 1.
    - Set SPI_USR_COMMAND.
    - Clear SPI_FCMD_DUAL and SPI_FCMD_QUAD.
2. Configure ADDR state related registers.
    - Configure the required address value in SPI_USR_ADDR_VALUE.
    - Configure address bit length in SPI_USR_ADDR_BITLEN. SPI_USR_ADDR_BITLEN = expected bit length - 1.
    - Set SPI_USR_ADDR and SPI_FADDR_DUAL.
    - Clear SPI_FADDR_QUAD.
3. Configure DUMMY state related registers.
    - Configure DUMMY cycles in SPI_USR_DUMMY_CYCLELEN. SPI_USR_DUMMY_CYCLELEN = expected clock cycles - 1.
    - Set SPI_USR_DUMMY.
4. Configure DIN state related registers.
    - Configure read data bit length in SPI_MS_DATA_BITLEN. SPI_MS_DATA_BITLEN = bit length expected - 1.
    - Set SPI_FREAD_QUAD and SPI_USR_MISO.
    - Clear SPI_FREAD_DUAL.
    - Configure GDMA in DMA-controlled mode. In CPU controlled mode, no action is needed.
5. Clear SPI_USR_MOSI.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Set SPI_USR to start GP-SPI transfer.

When writing data (DOUT state), SPI_USR_MOSI should be configured instead, while SPI_USR_MISO should be cleared. The output data bit length is the value of SPI_MS_DATA_BITLEN + 1. Output data should be configured in GP-SPI data buffer (SPI_W0_REG ~ SPI_W15_REG) in CPU-controlled mode, or GDMA TX buffer in DMA-controlled mode. The data byte order is incremented from LSB (byte 0) to MSB.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and to address value in SPI_USR_
ADDR_VALUE.

The configuration of command value is as follows:

Table 30.5-9. Sending Sequence of Command Value

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Command Value
0 - 7	[7:0]	1	COMMAND_VALUE[COMMAND_BITLEN:0] is sent first.
		0	COMMAND_VALUE[7:7 - COMMAND_BITLEN] is sent first.
8 - 15	[15:0]	1	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[COMMAND_BITLEN:8] is sent.
		0	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[15:15 - COMMAND_BITLEN] is sent.

[1] SPI_USR_COMMAND_BITLEN: this field is used to configure the bit length of the command.
[2] SPI_USR_COMMAND_VALUE: command value is written into this field. For which part of this field is used, see the table above.
[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

The configuration of address value is as follows:

Table 30.5-10. Sending Sequence of Address Value

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Address Value
0 - 7	[31:24]	1	COMMAND_VALUE[ADDR_BITLEN + 24:24] is sent first.
		0	ADDR_VALUE[31:31 - ADDR_BITLEN] is sent first.
8 - 15	[31:16]	1	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[ADDR_BITLEN + 8:16] is sent.
		0	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[23:31 - ADDR_BITLEN] is sent.
16 - 23	[31:8]	1	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 8:8] is sent.
		0	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[15:31 - ADDR_BITLEN] is sent.
24 - 31	[31:0]	1	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 24:0] is sent.
		0	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[7:31 - ADDR_BITLEN] is sent.

[1] SPI_USR_ADDR_BITLEN: this field is used to configure the bit length of the address.

[2] SPI_USR_ADDR_VALUE: address value is written into this field. For which part of this field is used, see the table above.

[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

30.5.8.3 Full-Duplex Communication (1-bit Mode Only)

Introduction

GP-SPI supports SPI full-duplex communication. In this mode, SPI master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode via MOSI (FSPID/SPI3_D, sending) and MISO (FSPIQ/SPI3_Q, receiving) at the same time. To enable this communication mode, set the bit SPI_DOUTDIN in register SPI_USER_REG. Figure 30.5-6 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

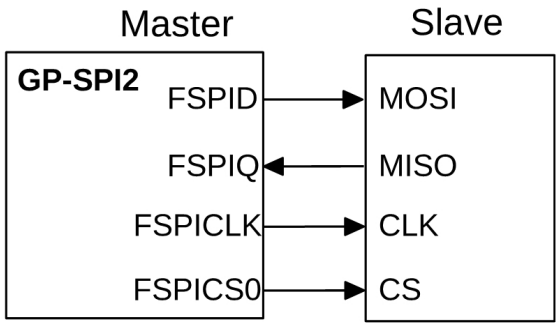


Figure 30.5-6. Full-Duplex Communication Between GP-SPI2 Master and a Slave

In full-duplex communication, the behavior of states CMD, ADDR, DUMMY, DOUT and DIN are configurable. Usually, the states CMD, ADDR and DUMMY are not used in this communication. The bit length of transferred

data is configured in SPI_MS_DATA_BITLEN. The actual bit length used in communication equals to (SPI_MS_DATA_BITLEN + 1).

Configuration (Take GP-SPI2 as an example)

To start a data transfer, follow the steps below:

- Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
- Configure APB clock (APB_CLK, see Chapter 7 *Reset and Clock*) and module clock (clk_spi_mst) for the GP-SPI2 module.
- Set SPI_DOUTDIN and clear SPI_SLAVE_MODE, to enable full-duplex communication in master mode.
- Configure GP-SPI2 registers listed in Table 30.5-8.
- Configure SPI CS setup time and hold time according to Section 30.6.
- Set the property of FSPICLK according to Section 30.7.
- Prepare data according to the selected transfer mode:
  - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
  - In DMA-controlled mode,
    * configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA
    * configure GDMA TX/RX link
    * start GDMA TX/RX engine, as described in Section 30.5.6 and Section 30.5.7.
- Configure interrupts and wait for SPI slave to get ready for transfer.
- Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
- Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

30.5.8.4 Half-Duplex Communication (1/2/4/8-bit Mode)

Introduction

In this mode, GP-SPI provides CLK and CS signals. Only one side (SPI master or slave) can send data at a time, while the other side receives the data. To enable this communication mode, clear the bit SPI_DOUTDIN in register SPI_USER_REG. The standard format of SPI half-duplex communication is CMD + [ADDR +] [DUMMY +] [DOUT or DIN]. The states ADDR, DUMMY, DOUT, and DIN are optional, and can be disabled or enabled independently.

As described in Section 30.5.8.2, the properties of GP-SPI states: CMD, ADDR, DUMMY, DOUT and DIN, such as cycle length, value, and parallel bus bit mode, can be set independently. For the register configuration, see Table 30.5-8.

The detailed properties of half-duplex GP-SPI are as follows:

1. CMD: 0 ~ 16 bits, master output, slave input.
2. ADDR: 0 ~ 32 bits, master output, slave input.
3. DUMMY: 0 ~ 256 FSPICLK/SPI3_CLK cycles, master output, slave input.

4. DOUT: 0 ~ 64 B in CPU-controlled mode, 0 ~ 32 KB in DMA-controlled single transfer mode and unlimited data length in DMA-controlled configurable segmented mode; master output, slave input.
5. DIN: 0 ~ 64 B in CPU-controlled mode and 0 ~ 32 KB in DMA-controlled single transfer mode and unlimited data length in DMA-controlled configurable segmented mode; master input, slave output.

Configuration (Take GP-SPI2 as an example)

The register configuration is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK) and module clock (clk_spi_mst) for the GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication in master mode.
4. Configure GP-SPI2 registers listed in Table 30.5-8.
5. Configure SPI CS setup time and hold time according to Section 30.6.
6. Set the property of FSPICLK according to Section 30.7.
7. Prepare data according to the selected transfer mode:
    - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
    - In DMA-controlled mode,
        - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA;
        - configure GDMA TX/RX link;
        - start GDMA TX/RX engine, as described in Section 30.5.6 and Section 30.5.7.
8. Configure interrupts and wait for SPI slave to get ready for transfer.
9. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
10. Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

Application Example

The following example shows how GP-SPI2 accesses flash and external RAM in master half-duplex mode.

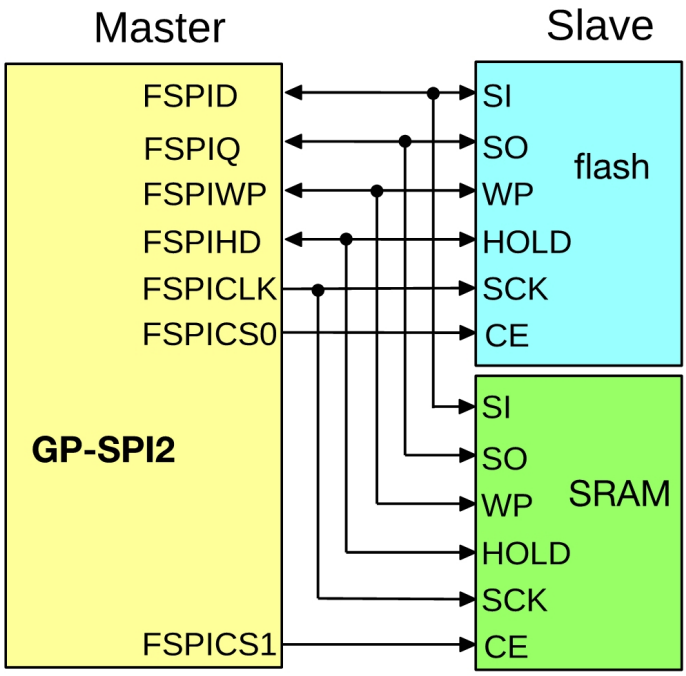


Figure 30.5-7. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

Figure 30.5-8 indicates GP-SPI2 Quad I/O Read sequence according to standard flash specification. Other GP-SPI2 command sequences are implemented in accordance with the requirements of SPI slaves.

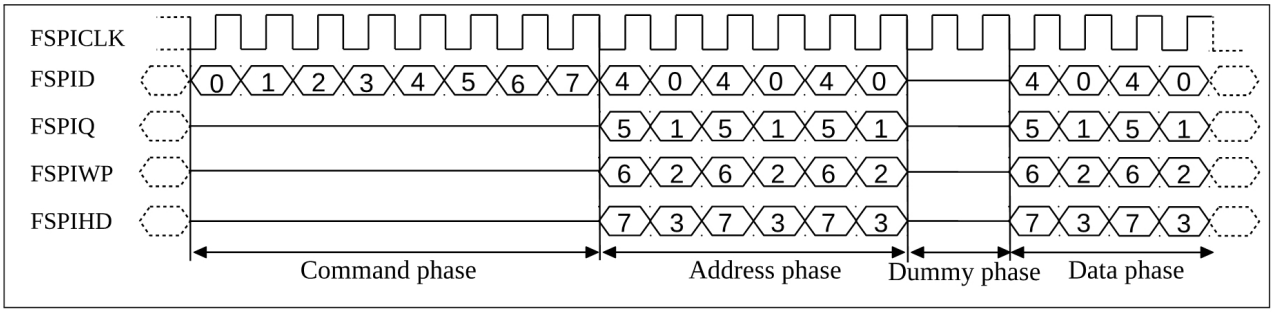


Figure 30.5-8. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

30.5.8.5 DMA-Controlled Configurable Segmented Transfer

Note:

- This feature is only supported by GP-SPI2.
- Note that there is no separate section on how to configure a single transfer in master mode, since the CONF state of a configurable segmented transfer can be skipped to implement a single transfer.

Introduction

When GP-SPI2 works as a master, it provides a feature named: configurable segmented transfer controlled by DMA.

A DMA-controlled transfer in master mode can be

- a single transfer, consisting of only one transaction;
- or a configurable segmented transfer, consisting of several transactions (segments).

In a configurable segmented transfer, the registers of each single transaction (segment) are configurable. This feature enables GP-SPI2 to do as many transactions (segments) as configured after such transfer is triggered once by the CPU. Figure 30.5-9 shows how this feature works.

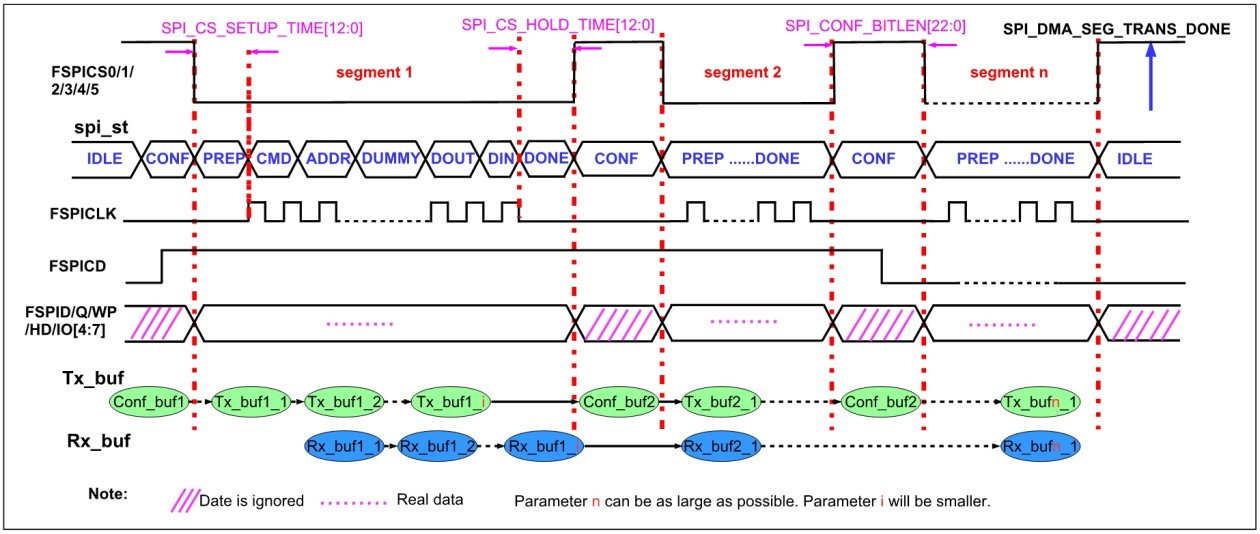


Figure 30.5-9. Configurable Segmented Transfer in DMA-Controlled Master Mode

As shown in Figure 30.5-9, the registers for one transaction (segment *n*) can be reconfigured by GP-SPI2 hardware according to the content in its Conf_buf*n* during a CONF state, before this segment starts.

It's recommended to provide separate GDMA CONF links and CONF buffers (Conf_buf*i* in Figure 30.5-9) for each CONF state. A GDMA TX link is used to connect all the CONF buffers and TX data buffers (Tx_buf*i* in Figure 30.5-9) into a chain. Hence, the behavior of the FSPI bus in each segment can be controlled independently.

For example, in a configurable segmentent transfer, its segment*i*, segment*j*, and segment*k* can be configured to full-duplex, half-duplex MISO, and half-duplex MOSI, respectively. *i*, *j*, and *k* are integer variables, which can be any segment number.

Meanwhile, the state of GP-SPI2, the data length and cycle length of the FSPI bus, and the behavior of the GDMA, can be configured independently for each segment. When this whole DMA-controlled transfer (consisting of several segments) has finished, a GP-SPI2 interrupt, SPI_DMA_SEG_TRANS_DONE_INT, is triggered.

Configuration

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK) and module clock (clk_spi_mst) for GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication in master mode.
4. Configure GP-SPI2 registers listed in Table 30.5-8.
5. Configure SPI CS setup time and hold time according to Section 30.6.
6. Set the property of FSPICLK according to Section 30.7.
7. Prepare descriptors for GDMA CONF buffer and TX data (optional) for each segment. Chain the descriptors of CONF buffer and TX buffers of several segments into one linked list.
8. Similarly, prepare descriptors for RX buffers for each segment and chain them into one linked list.

9. Configure all the needed CONF buffers, TX buffers and RX buffers, respectively for each segment before this DMA-controlled transfer begins.
10. Point GDMA_OUTLINK_ADDR_CH*n* to the head address of the CONF and TX buffer descriptor linked list, and then set GDMA_OUTLINK_START_CH*n* to start the TX GDMA.
11. Clear the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Point GDMA_INLINK_ADDR_CH*n* to the head address of the RX buffer descriptor linked list, and then set GDMA_INLINK_START_CH*n* to start the RX GDMA.
12. Set SPI_USR_CONF to enable CONF state.
13. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA to enable the SPI_DMA_SEG_TRANS_DONE_INT interrupt. Configure other interrupts if needed according to Section 30.10.
14. Wait for all the slaves to get ready for transfer.
15. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST and SPI_RX_AFIFO_RST, to reset these buffers.
16. Set SPI_USR to start this DMA-controlled transfer.
17. Wait for SPI_DMA_SEG_TRANS_DONE_INT interrupt, which means this transfer has finished and the data has been stored into corresponding memory.

Configuration of CONF Buffer and Magic Value

In a configurable segmented transfer, only registers which will change from the last transaction (segment) need to be re-configured to new values in CONF state. The configuration of other registers can be skipped (i.e., kept the same) to save time and chip resources.

The first word in GDMA CONF buffer*i*, called SPI_BIT_MAP_WORD, defines whether given GP-SPI2 register is to be updated or not in segment*i*. The relation of SPI_BIT_MAP_WORD and GP-SPI2 registers to update can be seen in Table 30.5-11 Bitmap (BM) Table. If a bit in the BM table is set to 1, its corresponding register value will be updated in this segment. Registers with corresponding bit set to 0 remains unchanged.

Table 30.5-11. BM Table for CONF State

BM Bit	Register Name	BM Bit	Register Name
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	SPI_DIN_MODE_REG
2	SPI_CLOCK_REG	9	SPI_DIN_NUM_REG
3	SPI_USER_REG	10	SPI_DOUT_MODE_REG
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

Then new values of all the registers to be modified should be placed right after SPI_BIT_MAP_WORD, in consecutive words in the CONF buffer.

To ensure the correctness of the content in each CONF buffer, the value in SPI_BIT_MAP_WORD[31:28] is used as "magic value", and will be compared with SPI_DMA_SEG_MAGIC_VALUE in register SPI_SLAVE_REG. The value of SPI_DMA_SEG_MAGIC_VALUE should be configured before this DMA-controlled transfer starts, and can not be changed during these segments.

- If SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE, this DMA-controlled transfer continues normally; the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered at the end of this DMA-controlled transfer.
- If SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE, GP-SPI2 state (spi_st) goes back to IDLE and the transfer is ended immediately. The interrupt SPI_DMA_SEG_TRANS_DONE_INT is still triggered, with SPI_SEG_MAGIC_ERR_INT_RAW bit set to 1.

CONF Buffer Configuration Example

Table 30.5-12 and Table 30.5-13 provide an example to show how to configure a CONF buffer for a transaction (segment *i*) in which SPI_ADDR_REG, SPI_CTRL_REG, SPI_CLOCK_REG, SPI_USER_REG, SPI_USER1_REG need to be updated.

Table 30.5-12. An Example of CONF buffer*i* in Segment*i*

CONF buffer*i*	Note
SPI_BIT_MAP_WORD	The first word in this buffer. Its value is 0xA000001F in this example when the SPI_DMA_SEG_MAGIC_VALUE is set to 0xA. As shown in Table 30.5-13, bits 0, 1, 2, 3, and 4 are set, indicating the following registers will be updated.
SPI_ADDR_REG	The second word, stores the new value to SPI_ADDR_REG.
SPI_CTRL_REG	The third word, stores the new value to SPI_CTRL_REG.
SPI_CLOCK_REG	The fourth word, stores the new value to SPI_CLOCK_REG.
SPI_USER_REG	The fifth word, stores the new value to SPI_USER_REG.
SPI_USER1_REG	The sixth word, stores the new value to SPI_USER1_REG.

Table 30.5-13. BM Bit Value v.s. Register to Be Updated in This Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	SPI_DIN_MODE_REG
2	1	SPI_CLOCK_REG	9	0	SPI_DIN_NUM_REG
3	1	SPI_USER_REG	10	0	SPI_DOUT_MODE_REG
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

Notes:

In a DMA-controlled configurable segmented transfer, please pay special attention to the following bits:

- SPI_USR_CONF: set SPI_USR_CONF before SPI_USR is set, to enable this transfer.
- SPI_USR_CONF_NXT: if segment*i* is not the final transaction of this whole DMA-controlled transfer, its SPI_USR_CONF_NXT should be set to 1.
- SPI_CONF_BITLEN: GP-SPI2 CS setup time and hold time are programmable independently in each segment, see Section 30.6 for detailed configuration. The CS high time in each segment is about:

$$(SPI\_CONF\_BITLEN + 5) \times T_{APB\_CLK}$$

The CS high time in CONF state can be set from 62.5 ns to 3.2768 ms when f_{APB_CLK} is 80 MHz. (SPI_CONF_
BITLEN + 5) will overflow from (0x40000 - SPI_CONF_BITLEN - 5) if SPI_CONF_BITLEN is larger than 0x3FFFA.

30.5.9 GP-SPI Works as a Slave

GP-SPI can be used as a slave to communicate with an SPI master. As a slave, GP-SPI supports 1-bit SPI, 2-bit dual SPI, 4-bit quad SPI, and QPI modes, with specific communication formats. To enable this mode, set SPI_SLAVE_MODE in register SPI_SLAVE_REG.

The CS signal must be held low during the transmission, and its falling/rising edges indicate the start/end of a single or segmented transfer.

Note:
The length of transferred data must be in unit of bytes, otherwise the extra bits will be lost. The extra bits here means the result of total bits % 8.

30.5.9.1 Communication Formats

In GP-SPI slave mode, SPI full-duplex and half-duplex communications are available. To select from the two communications, configure SPI_DOUTDIN in register SPI_USER_REG.

Full-duplex communication means that input data and output data are transmitted simultaneously throughout the entire transaction. All bits are treated as input or output data, which means no command, address or dummy states are expected. The interrupt SPI_TRANS_DONE_INT is triggered once the transaction ends.

In half-duplex communication, the format is CMD+ADDR+DUMMY+DATA (DIN or DOUT).

- "DIN" means that an SPI master reads data from GP-SPI.
- "DOUT" means that an SPI master writes data to GP-SPI.

The detailed properties of each state are as follows:

1. CMD:
    - Indicate the function of SPI slave;
    - One byte from master to slave;
    - Only the values in Table 30.5-14 and Table 30.5-15 are valid;
    - Can be sent in 1-bit SPI mode or 4-bit QPI mode.
2. ADDR:
    - The address for Wr_BUF and Rd_BUF commands in CPU-controlled transfer, or placeholder bits in other transfers and can be defined by application;
    - One byte from master to slave;
    - Can be sent in 1-bit, 2-bit or 4-bit modes (according to the command).
3. DUMMY:

- Its value is meaningless. SPI slave prepares data in this state;
- Bit mode of FSPI/SPI3 bus is also meaningless here;
- Last for eight SPI_CLK cycles.

4. DIN or DOUT:
   - Data length can be 0 ~ 64 B in CPU-controlled mode and unlimited in DMA-controlled mode;
   - Can be sent in 1-bit, 2-bit or 4-bit modes according to the CMD value.

> **Note:**
> The states of ADDR and DUMMY can never be skipped in any half-duplex communications.

When a half-duplex transaction is complete, the transferred CMD and ADDR values are latched into SPI_SLV_LAST_COMMAND and SPI_SLV_LAST_ADDR respectively. The SPI_SLV_CMD_ERR_INT_RAW will be set if the transferred CMD value is not supported by GP-SPI slave mode. The SPI_SLV_CMD_ERR_INT_RAW can only be cleared by software.

30.5.9.2 Supported CMD Values in Half-Duplex Communication

In half-duplex communication, the defined values of CMD determine the transfer types. Unsupported CMD values are disregarded, meanwhile the related transfer is ignored and SPI_SLV_CMD_ERR_INT_RAW is set. The transfer format is CMD (8 bits) + ADDR (8 bits) + DUMMY (8 SPI_CLK cycles) + DATA (unit in bytes). The detailed description of CMD[3:0] is as follows:

- 0x1 (Wr_BUF): CPU-controlled write mode. Master sends data and GP-SPI receives data. The data is stored in the related address of SPI_W0_REG ~ SPI_W15_REG.
- 0x2 (Rd_BUF): CPU-controlled read mode. Master receives the data sent by GP-SPI. The data comes from the related address of SPI_W0_REG ~ SPI_W15_REG.
- 0x3 (Wr_DMA): DMA-controlled write mode. Master sends data and GP-SPI receives data. The data is stored in GP-SPI GDMA RX buffer.
- 0x4 (Rd_DMA): DMA-controlled read mode. Master receives the data sent by GP-SPI. The data comes from GP-SPI GDMA TX buffer.
- 0x7 (CMD7): used to generate an SPI_SLV_CMD7_INT interrupt. It can also generate a GDMA_IN_SUC_EOF_CH*n*_INT interrupt in a slave segmented transfer when GDMA RX link is used. But it will not end GP-SPI's slave segmented transfer.
- 0x8 (CMD8): only used to generate an SPI_SLV_CMD8_INT interrupt, which will not end GP-SPI's slave segmented transfer.
- 0x9 (CMD9): only used to generate an SPI_SLV_CMD9_INT interrupt, which will not end GP-SPI's slave segmented transfer.
- 0xA (CMDA): only used to generate an SPI_SLV_CMDA_INT interrupt, which will not end GP-SPI's slave segmented transfer.

The detailed function of CMD7, CMD8, CMD9, and CMDA commands is reserved for user definition. These commands can be used as handshake signals, the passwords of some specific functions, the triggers of some user defined actions, and so on.

1/2/4-bit modes in states of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles. The definition of CMD[7:4] is as follows:

- 0x0: CMD, ADDR, and DATA states all are in 1-bit mode.
- 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
- 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
- 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
- 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode. Or in QPI mode.

In addition, if the value of CMD[7:0] is 0x05, 0xA5, 0x06, or 0xDD, DUMMY and DATA states are skipped. The definition of CMD[7:0] is as follows:

- 0x05 (End_SEG_TRANS): master sends 0x05 command to end slave segmented transfer in SPI mode.
- 0xA5 (End_SEG_TRANS): master sends 0xA5 command to end slave segmented transfer in QPI mode.
- 0x06 (En_QPI): GP-SPI enters QPI mode when receiving the 0x06 command and the bit SPI_QPI_MODE in register SPI_USER_REG is set.
- 0xDD (Ex_QPI): GP-SPI exits QPI mode when receiving the 0xDD command and the bit SPI_QPI_MODE is cleared.

All the CMD values supported by GP-SPI are listed in Table 30.5-14 and Table 30.5-15. Note that DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles.

Table 30.5-14. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0x01	1-bit mode	1-bit mode	1-bit mode
	0x11	1-bit mode	1-bit mode	2-bit mode
	0x21	1-bit mode	1-bit mode	4-bit mode
	0x51	1-bit mode	2-bit mode	2-bit mode
	0xA1	1-bit mode	4-bit mode	4-bit mode
Rd_BUF	0x02	1-bit mode	1-bit mode	1-bit mode
	0x12	1-bit mode	1-bit mode	2-bit mode
	0x22	1-bit mode	1-bit mode	4-bit mode
	0x52	1-bit mode	2-bit mode	2-bit mode
	0xA2	1-bit mode	4-bit mode	4-bit mode
Wr_DMA	0x03	1-bit mode	1-bit mode	1-bit mode
	0x13	1-bit mode	1-bit mode	2-bit mode
	0x23	1-bit mode	1-bit mode	4-bit mode
	0x53	1-bit mode	2-bit mode	2-bit mode
	0xA3	1-bit mode	4-bit mode	4-bit mode
Rd_DMA	0x04	1-bit mode	1-bit mode	1-bit mode
	0x14	1-bit mode	1-bit mode	2-bit mode

Table 30.5-14. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
	0x24	1-bit mode	1-bit mode	4-bit mode
	0x54	1-bit mode	2-bit mode	2-bit mode
	0xA4	1-bit mode	4-bit mode	4-bit mode
CMD7	0x07	1-bit mode	1-bit mode	-
	0x17	1-bit mode	1-bit mode	-
	0x27	1-bit mode	1-bit mode	-
	0x57	1-bit mode	2-bit mode	-
	0xA7	1-bit mode	4-bit mode	-
CMD8	0x08	1-bit mode	1-bit mode	-
	0x18	1-bit mode	1-bit mode	-
	0x28	1-bit mode	1-bit mode	-
	0x58	1-bit mode	2-bit mode	-
	0xA8	1-bit mode	4-bit mode	-
CMD9	0x09	1-bit mode	1-bit mode	-
	0x19	1-bit mode	1-bit mode	-
	0x29	1-bit mode	1-bit mode	-
	0x59	1-bit mode	2-bit mode	-
	0xA9	1-bit mode	4-bit mode	-
CMDA	0x0A	1-bit mode	1-bit mode	-
	0x1A	1-bit mode	1-bit mode	-
	0x2A	1-bit mode	1-bit mode	-
	0x5A	1-bit mode	2-bit mode	-
	0xAA	1-bit mode	4-bit mode	-
End_SEG_TRANS	0x05	1-bit mode	-	-
En_QPI	0x06	1-bit mode	-	-

Table 30.5-15. Supported CMD Values in QPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0xA1	4-bit mode	4-bit mode	4-bit mode
Rd_BUF	0xA2	4-bit mode	4-bit mode	4-bit mode
Wr_DMA	0xA3	4-bit mode	4-bit mode	4-bit mode
Rd_DMA	0xA4	4-bit mode	4-bit mode	4-bit mode
CMD7	0xA7	4-bit mode	4-bit mode	-
CMD8	0xA8	4-bit mode	4-bit mode	-
CMD9	0xA9	4-bit mode	4-bit mode	-
CMDA	0xAA	4-bit mode	4-bit mode	-
End_SEG_TRANS	0xA5	4-bit mode	4-bit mode	-
Ex_QPI	0xDD	4-bit mode	4-bit mode	-

Master sends 0x06 CMD (En_QPI) to set GP-SPI slave to QPI mode and all the states of supported transfer will be in 4-bit mode afterwards. If 0xDD CMD (Ex_QPI) is received, GP-SPI slave will be back to SPI mode.

Other transfer types than described in Table 30.5-14 and Table 30.5-15 are ignored. If the transferred data is not in unit of byte, GP-SPI can send or receive these extra bits (total bits mod 8), however, the correctness of the data is not guaranteed. But if the CS low time is longer than two APB clock (APB_CLK) cycles, SPI_TRANS_DONE_INT will be triggered. For more information on interrupts triggered at the end of transmissions, please refer to Section 30.10.

30.5.9.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI works as a slave, it supports full-duplex and half-duplex communications controlled by DMA and by CPU. DMA-controlled transfer can be a single transfer, or a slave segmented transfer consisting of several transactions (segments). The CPU-controlled transfer can only be one single transfer, since each CPU-controlled transaction needs to be triggered by CPU.

In a slave segmented transfer, all transfer types listed in Table 30.5-14 and Table 30.5-15 are supported in a single transaction (segment). It means that CPU-controlled transaction and DMA-controlled transaction can be mixed in one slave segmented transfer.

It is recommended that in a slave segmented transfer:

- CPU-controlled transaction is used for handshake communication and short data transfers.
- DMA-controlled transaction is used for large data transfers.

30.5.9.4 Configuration of Slave Single Transfer

In slave mode, GP-SPI supports CPU/DMA-controlled full-duplex/half-duplex single transfers. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set the bit SPI_SLAVE_MODE, to enable slave mode.
4. Configure SPI_DOUTDIN:
    - 1: enable full-duplex communication.
    - 0: enable half-duplex communication.
5. Prepare data:
    - if CPU-controlled transfer mode is selected and GP-SPI is used to send data, then prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
    - if DMA-controlled transfer mode is selected,
        - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA and SPI_RX_EOF_EN.
        - configure GDMA TX/RX link.
        - start GDMA TX/RX engine, as described in Section 30.5.6 and Section 30.5.7.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Clear SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave single transfer mode.

8. Set SPI_TRANS_DONE_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_TRANS_DONE_INT. In DMA-controlled mode, it is recommended to wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT when GDMA RX buffer is used, which means that data has been stored in the related memory. Other interrupts described in Section 30.10 are optional.

30.5.9.5 Configuration of Slave Segmented Transfer in Half-Duplex

GDMA must be used in this mode. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE to enable slave mode.
4. Clear SPI_DOUTDIN to enable half-duplex communication.
5. Prepare data in registers SPI_W0_REG ~ SPI_W15_REG, if needed.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST and SPI_RX_AFIFO_RST to reset these buffers.
7. Set bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA. Clear the bit SPI_RX_EOF_EN. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 30.5.6 and Section 30.5.7.
8. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer.
9. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_DMA_SEG_TRANS_DONE_INT, which means that the segmented transfer has finished and data has been put into the related memory. Other interrupts described in Section 30.10 are optional.

When End_SEG_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI, this slave segmented transfer is ended and the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered.

30.5.9.6 Configuration of Slave Segmented Transfer in Full-Duplex

GDMA must be used in this mode. In such transfer, the data is transferred from and to the GDMA buffer. The interrupt GDMA_IN_SUC_EOF_CH*n*_INT is triggered when the transfer ends. The configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE and SPI_DOUTDIN, to enable full-duplex communication in slave mode.
4. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST, to reset these buffers.
5. Set SPI_DMA_TX_ENA/SPI_DMA_RX_ENA. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 30.5.6 and Section 30.5.7.
6. Set the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Configure SPI_MS_DATA_BITLEN[17:0] in register SPI_MS_DLEN_REG to the byte length of the received DMA data.
7. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer mode.
8. Set GDMA_IN_SUC_EOF_CH*n*_INT_ENA and wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT.

30.6 CS Setup Time and Hold Time Control

SPI bus CS (SPI_CS) setup time and hold time are very important to meet the timing requirements of various SPI devices (e.g., flash or PSRAM).

CS setup time is the time between the CS falling edge and the first latch edge of SPI bus CLK (SPI_CLK). The first latch edge for mode 0 and mode 3 is rising edge, and falling edge for mode 1 and mode 2.

CS hold time is the time between the last latch edge of SPI_CLK and the CS rising edge.

In slave mode, the CS setup time and hold time should be longer than 0.5 x T_SPI_CLK, otherwise the SPI transfer may be incorrect. T_SPI_CLK: one cycle of SPI_CLK.

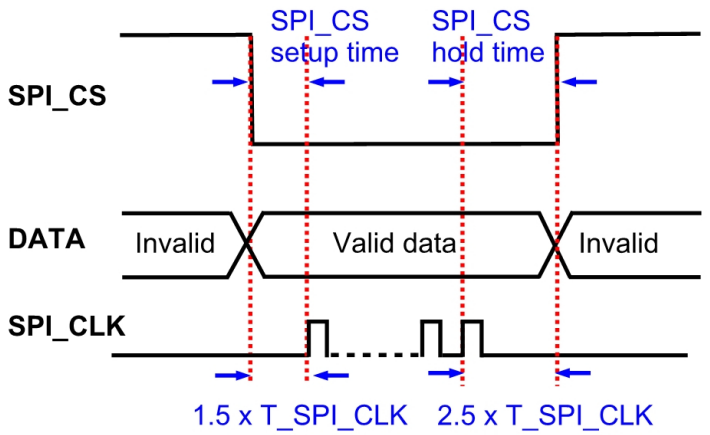
In master mode, set the CS setup time by specifying SPI_CS_SETUP in SPI_USER_REG and SPI_CS_SETUP_TIME in SPI_USER1_REG:

- If SPI_CS_SETUP is cleared, the SPI CS setup time is 0.5 x T_SPI_CLK.
- If SPI_CS_SETUP is set, the SPI CS setup time is (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK.

Set the CS hold time by specifying SPI_CS_HOLD in SPI_USER_REG and SPI_CS_HOLD_TIME in SPI_USER1_REG:

- If SPI_CS_HOLD is cleared, the SPI CS hold time is 0.5 x T_SPI_CLK.
- If SPI_CS_HOLD is set, the SPI CS hold time is (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK.

Figure 30.6-1 and Figure 30.6-2 show the recommended CS timing and register configuration to access external RAM and flash.



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

Figure 30.6-1. Recommended CS Timing and Settings When Accessing External RAM

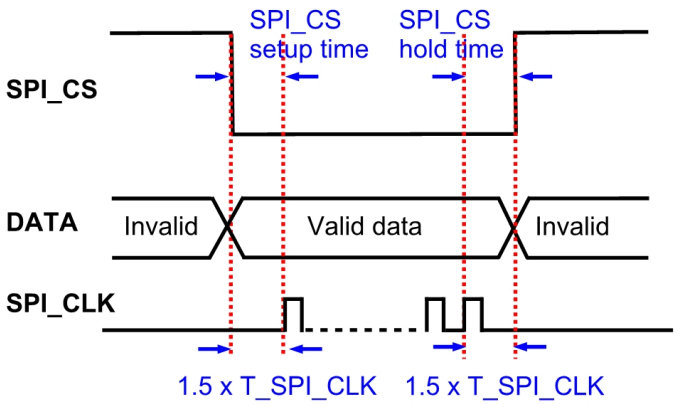


Figure 30.6-2. Recommended CS Timing and Settings When Accessing Flash

30.7 GP-SPI Clock Control

GP-SPI has the following clocks:

- **clk_spi_mst**: module clock of GP-SPI, derived from PLL_CLK or XTAL_CLK. It is controlled by bits SPI_MST
  _CLK_ACTIVE and SPI_MST_CLK_SEL. Used in GP-SPI master mode, to generate SPI_CLK signal for data transfer and for slaves.
- **clk_hclk**: module timing compensation clock of GP-SPI. When PLL_CLK is available and the bit SPI_TIMING_
  HCLK_ACTIVE is set, the frequency is 160 MHz; otherwise, it is powered off.
- **SPI_CLK**: output clock in master mode.
- **APB_CLK**: clock for register configuration.

In master mode, the maximum output clock frequency of GP-SPI is $f_{clk\_spi\_mst}$. To have slower frequencies, the output clock frequency can be divided as follows:

$$f_{SPI\_CLK} = \frac{f_{clk\_spi\_mst}}{(SPI\_CLKCNT\_N + 1)(SPI\_CLKDIV\_PRE + 1)}$$

The divider is configured by SPI_CLKCNT_N and SPI_CLKDIV_PRE in register SPI_CLOCK_REG. When the bit SPI_CLK_EQU_SYSCLK in register SPI_CLOCK_REG is set to 1, the output clock frequency of GP-SPI will be $f_{clk\_spi\_mst}$. And for other integral clock divisions, SPI_CLK_EQU_SYSCLK should be set to 0.

In slave mode, the supported input clock frequency (f_{SPI_CLK}) of GP-SPI is:

- If f_{APB_CLK} >= 60 MHz, f_{SPI_CLK} <= 60 MHz;
- If f_{APB_CLK} < 60 MHz, f_{SPI_CLK} <= f_{APB_CLK}.

30.7.1 Clock Phase and Polarity

There are four clock modes in SPI protocol, modes 0 ~ 3, see Figure 30.7-1 and Figure 30.7-2 (excerpted from SPI protocol):

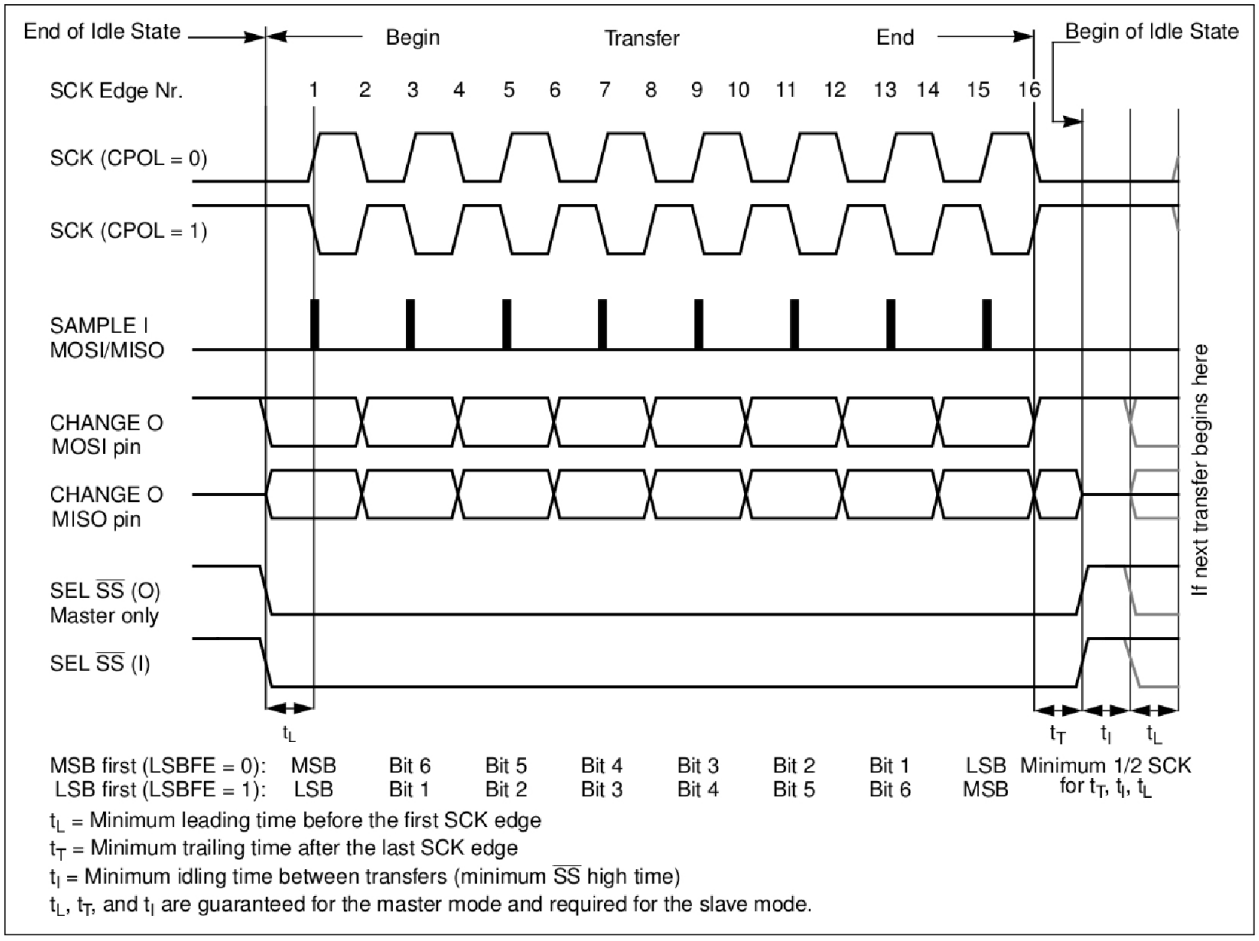


Figure 30.7-1. SPI Clock Mode 0 or 2

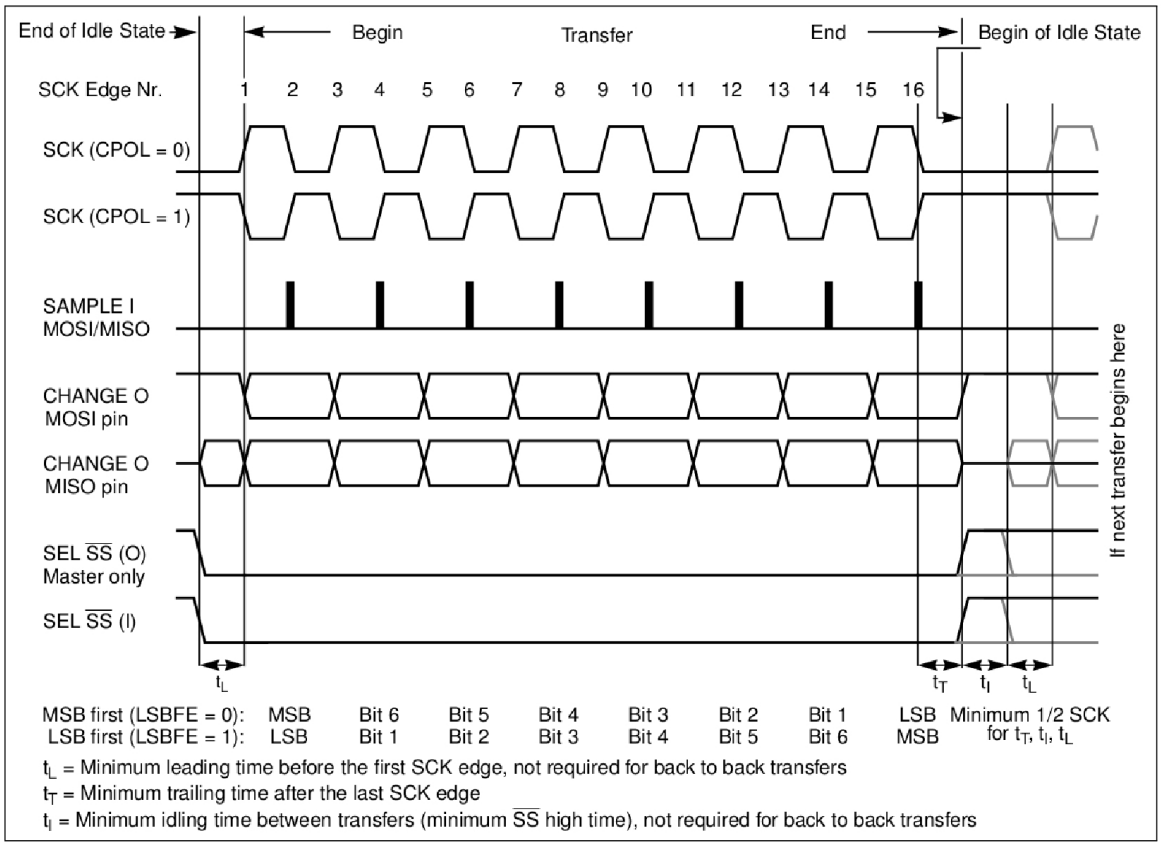


Figure 30.7-2. SPI Clock Mode 1 or 3

- Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge. The first data is shifted out before the first negative edge of SCK.
- Mode 1: CPOL = 0, CPHA = 1; SCK is 0 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge.
- Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge. The first data is shifted out before the first positive edge of SCK.
- Mode 3: CPOL = 1, CPHA = 1; SCK is 1 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge.

30.7.2 Clock Control in Master Mode

The four clock modes 0 ~ 3 are supported in GP-SPI master mode. The polarity and phase of GP-SPI clock are controlled by the bit SPI_CK_IDLE_EDGE in register SPI_MISC_REG and the bit SPI_CK_OUT_EDGE in register SPI_USER_REG. The register configuration for SPI clock modes 0 ~ 3 is provided in Table 30.7-1, and can be changed according to the path delay in the application.

Table 30.7-1. Clock Phase and Polarity Configuration in Master Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

SPI_CLK_MODE is used to select the number of rising edges of SPI_CLK, when SPI_CS raises high, to be 0, 1, 2 or SPI_CLK always on.

Note:
When SPI_CLK_MODE is configured to 1 or 2, the bit SPI_CS_HOLD must be set and the value of SPI_CS_HOLD_TIME should be larger than 1.

30.7.3 Clock Control in Slave Mode

GP-SPI slave mode also supports clock modes 0 ~ 3. The polarity and phase are configured by the bits SPI_TSCK
_I_EDGE and SPI_RSCK_I_EDGE in register SPI_USER_REG. The output edge of data is controlled by SPI_CLK_
MODE_13 in register SPI_SLAVE_REG. The detailed register configuration is shown in Table 30.7-2:

Table 30.7-2. Clock Phase and Polarity Configuration in Slave Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

30.8 GP-SPI Timing Compensation

Introduction (Take GP-SPI2 as an example)

The I/O lines are mapped via GPIO matrix or IO MUX for GP-SPI2. But there is no timing adjustment in IO MUX. The input data and output data can be delayed for 1 or 2 APB_CLK cycles at the rising or falling edge in GPIO matrix. For detailed register configuration, see Chapter *6 IO MUX and GPIO Matrix (GPIO, IO MUX)*.

Figure 30.8-1 shows the timing compensation control for GP-SPI2 master mode, including the following paths:

- "CLK": the output path of GP-SPI2 bus clock. The clock is sent out by SPI_CLK out control module, passes through GPIO matrix or IO MUX and then goes to an external SPI device.
- "IN": data input path of GP-SPI2 (see line 3 path in color purple in Figure 30.8-1). The input data from an external SPI device passes through GPIO matrix or IO MUX, then is adjusted by the Timing Module (see Figure 30.5-2) and finally is stored into spi_rx_afifo.
- "OUT": data output path of GP-SPI2 (see line 2 path in color rose-red in Figure 30.8-1). The output data is sent out to the Timing Module, passes through GPIO matrix or IO MUX and is then captured by an external SPI device.

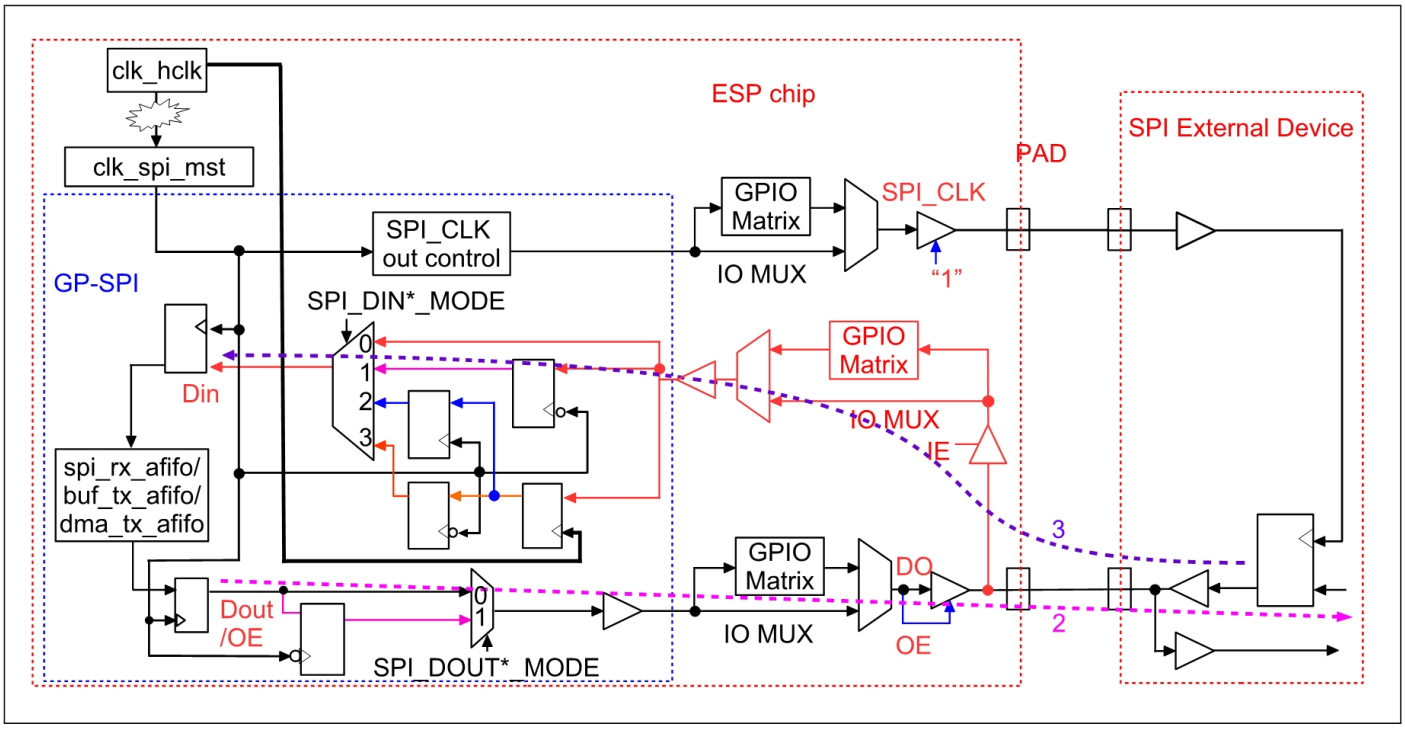


Figure 30.8-1. Timing Compensation Control Diagram in GP-SPI2 Master Mode

Every input and output data is passing through the Timing Module and the module can be used to apply delay in units of $T_{\text{clk\_spi\_mst}}$ (one cycle of clk_spi_mst) on rising or falling edge.

Key Registers

- SPI_DIN_MODE_REG: select the latch edge of input data
- SPI_DIN_NUM_REG: select the delay cycles of input data
- SPI_DOUT_MODE_REG: select the latch edge of output data

Timing Compensation Example

Figure 30.8-2 shows a timing compensation example in GP-SPI2 master mode. Note that DUMMY cycle length is configurable to compensate the delay in I/O lines, so as to enhance the performance of GP-SPI2.

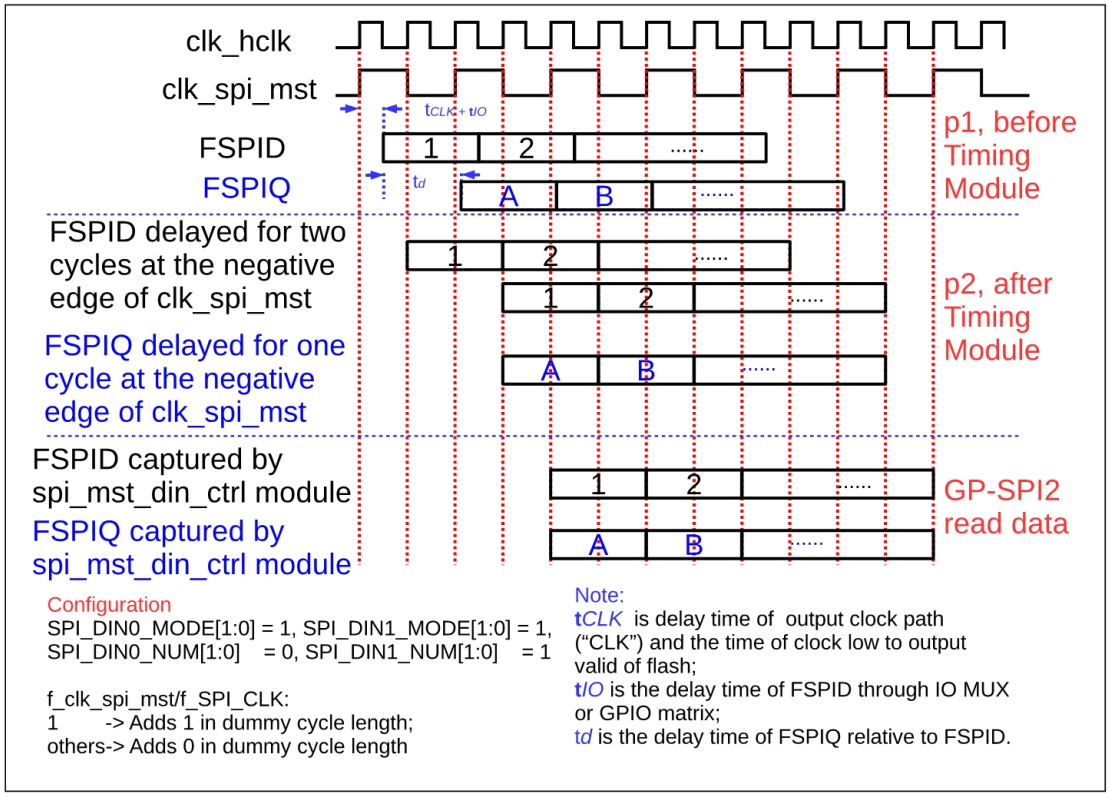


Figure 30.8-2. Timing Compensation Example in GP-SPI2 Master Mode

In Figure 30.8-2, "p1" is the point of input data of Timing Module, "p2" is the point of output data of Timing Module. Since the input data FSPIQ is unaligned to FSPID, the read data of GP-SPI2 will be wrong without the timing compensation.

To get correct read data, follow the the settings below, assuing $f_{clk\_spi\_mst}$ equals to f_{SPI_CLK}:

- Delay FSPID for two cycles at the falling edge of clk_spi_mst.
- Delay FSPIQ for one cycle at the falling edge of clk_spi_mst.
- Add one extra dummy cycle.

In GP-SPI2 slave mode, if the bit SPI_RSCK_DATA_OUT in register SPI_SLAVE_REG is set to 1, the output data is sent at latch edge, which is half an SPI clock cycle earlier. This can be used for slave mode timing compensation.

30.9 Differences Between GP-SPI2 and GP-SPI3

The feature differences between GP-SPI2 and GP-SPI3 are as follows:

- The communication mode for each GP-SPI2 state (CMD, ADDR, DOUT or DIN) can be configured independently. Data can either be in 1/2/4/8-bit master mode or 1/2/4-bit slave mode. Whereas GP-SPI3 supports 1/2/4-bit master mode or 1/2/4-bit slave mode.
- DMA-controlled configurable segmented transfer is only supported in GP-SPI2. Therefore, CONF state is not used in GP-SPI3.
- The I/O lines of GP-SPI2 can be mapped to physical GPIO pins either via GPIO matrix or IO MUX. However, GP-SPI3 lines can be configured only via GPIO matrix.
- GP-SPI2 has six CS signals in master mode. GP-SPI3 only has three CS signals in master mode.

Apart from that, the functions of GP-SPI2 and GP-SPI3 are the same. GP-SPI2 can use all the GP-SPI registers, while GP-SPI3 can only use some of the GP-SPI registers, see Table 30.9-1 for details.

Table 30.9-1. Invalid Registers and Fields for GP-SPI3

Invalid Register	Invalid Field
SPI_USER_REG	SPI_OPI_MODE SPI_FWRITE_OCT
SPI_CTRL_REG	SPI_FADDR_OCT SPI_FCMD_OCT SPI_FREAD_OCT
SPI_MISC_REG	SPI_CS3_DIS SPI_CS4_DIS SPI_CS5_DIS SPI_MASTER_CS_POL[5:3]
SPI_DIN_MODE_REG	SPI_DIN4_MODE SPI_DIN5_MODE SPI_DIN6_MODE SPI_DIN7_MODE
SPI_DIN_NUM_REG	SPI_DIN4_NUM SPI_DIN5_NUM SPI_DIN6_NUM SPI_DIN7_NUM
SPI_DOUT_MODE_REG	SPI_DOUT4_MODE SPI_DOUT5_MODE SPI_DOUT6_MODE SPI_DOUT7_MODE

GP-SPI3 has the same 1/2/4-bit mode functions and register configuration rules as to GP-SPI2. GP-SPI3 interface can be seen as a 1/2/4-bit mode GP-SPI2 interface, without DMA-controlled configurable segmented transfer.

30.10 Interrupts

Interrupt Summary

GP-SPI provides SPI_INTR_2/3 interrupt interfaces. When an SPI transfer ends, an interrupt is generated in GP-SPI:

- SPI_DMA_INFIFO_FULL_ERR_INT: triggered when GDMA RX FIFO length is shorter than the real transferred data length.
- SPI_DMA_OUTFIFO_EMPTY_ERR_INT: triggered when GDMA TX FIFO length is shorter than the real transferred data length.
- SPI_SLV_EX_QPI_INT: triggered when Ex_QPI is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_EN_QPI_INT: triggered when En_QPI is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_CMD7_INT: triggered when CMD7 is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_CMD8_INT: triggered when CMD8 is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_CMD9_INT: triggered when CMD9 is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_CMDA_INT: triggered when CMDA is received correctly in GP-SPI slave mode and the SPI transfer ends.
- SPI_SLV_RD_DMA_DONE_INT: triggered at the end of Rd_DMA transfer in slave mode.
- SPI_SLV_WR_DMA_DONE_INT: triggered at the end of Wr_DMA transfer in slave mode.
- SPI_SLV_RD_BUF_DONE_INT: triggered at the end of Rd_BUF transfer in slave mode.
- SPI_SLV_WR_BUF_DONE_INT: triggered at the end of Wr_BUF transfer in slave mode.
- SPI_TRANS_DONE_INT: triggered at the end of SPI bus transfer in both master and slave modes.
- SPI_DMA_SEG_TRANS_DONE_INT: triggered at the end of End_SEG_TRANS transfer in GP-SPI slave segmented transfer mode or at the end of configurable segmented transfer in master mode.
- SPI_SEG_MAGIC_ERR_INT: triggered when a Magic error occurs in CONF buffer during configurable segmented transfer in master mode. (Only valid in GP-SPI2)
- SPI_MST_RX_AFIFO_WFULL_ERR_INT: triggered by RX AFIFO write-full error in GP-SPI master mode.
- SPI_MST_TX_AFIFO_REMPTY_ERR_INT: triggered by TX AFIFO read-empty error in GP-SPI master mode.
- SPI_SLV_CMD_ERR_INT: triggered when a received command value is not supported in GP-SPI slave mode.
- SPI_APP2_INT: Set SPI_APP2_INT_SET to trigger this interrupt. It is only used for user defined function.
- SPI_APP1_INT: Set SPI_APP1_INT_SET to trigger this interrupt. It is only used for user defined function.

Interrupts Used in Master and Slave Modes

Table 30.10-1 and Table 30.10-2 show the interrupts used in GP-SPI master and slave modes. Set the interrupt enable bit SPI_*_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the SPI_INT interrupt. When the transfer ends, the related interrupt is triggered and should be cleared by software before the next transfer.

Table 30.10-1. GP-SPI Master Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	Half-duplex MISO Mode	DMA	GDMA_IN_SUC_EOF_CH*n*_INT
		CPU	SPI_TRANS_DONE_INT
Configurable Segmented Transfer	Full-duplex	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	Not supported
	Half-duplex MOSI Mode	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported
	Half-duplex MISO	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported

[1] If GDMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data of GP-SPI has been stored in the RX buffer, and the TX data has been transferred to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG ~ SPI_W15_REG with slave in this mode.

[3] If SPI_DMA_SEG_TRANS_DONE_INT is triggered, it means that the whole configurable segmented transfer (consisting of several segments) has finished, i.e., the RX data has been stored in the RX buffer completely and all the TX data has been sent out.

Table 30.10-2. GP-SPI Slave Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	GDMA_IN_SUC_EOF_CH*n*_INT [3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT [4]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_TRANS_DONE_INT [5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT [6]
Slave Segmented Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [7]
		CPU	Not supported [8]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [9]
		CPU (Wr_BUF)	Not supported [10]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [11]
		CPU (Rd_BUF)	Not supported [12]

Table 30.10-2 – Continued from the previous page

Transfer Type	Communication Mode	Controlled by	Interrupt

[1] If GDMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data has been stored in the RX buffer, and the TX data has been sent to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG ~ SPI_W15_REG with slave in this mode.

[3] SPI_SLV_WR_DMA_DONE_INT just means that the transmission on the SPI bus is done, but can not ensure that all the push data has been stored in the RX buffer. For this reason, GDMA_IN_SUC_EOF_CH*n*_INT is recommended.

[4] Or wait for SPI_SLV_WR_BUF_DONE_INT.

[5] Or wait for SPI_SLV_RD_DMA_DONE_INT.

[6] Or wait for SPI_SLV_RD_BUF_DONE_INT.

[7] Slave should set the total read data byte length in SPI_MS_DATA_BITLEN before the transfer begins. Set SPI_RX_EOF_EN to 1 before the end of the interrupt program.

[8] Master and slave should define a method to end the segmented transfer, such as via GPIO interrupt.

[9] Master sends End_SEG_TRAN to end the segmented transfer or slave sets the total read data byte length in SPI_MS_DATA_BITLEN and waits for GDMA_IN_SUC_EOF_CH*n*_INT.

[10] Half-duplex Wr_BUF single transfer can be used in a slave segmented transfer.

[11] Master sends End_SEG_TRAN to end the segmented transfer.

[12] Half-duplex Rd_BUF single transfer can be used in a slave segmented transfer.

30.11 Register Summary

The addresses in this section are relative to SPI2/SPI3 base address provided in Table 4.3-3 in Chapter 4 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

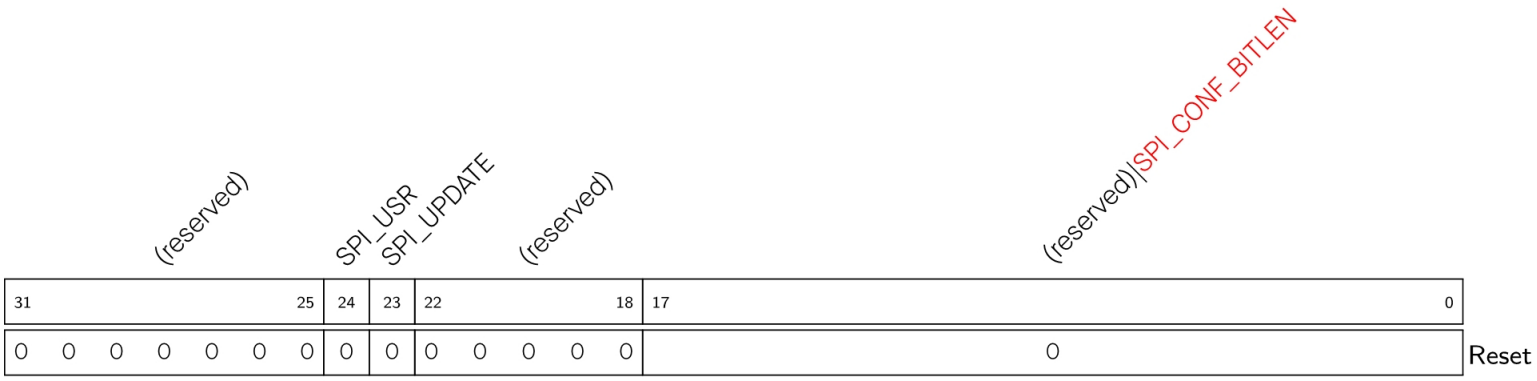
Name	Description	SPI2 Address	SPI3 Address	Access
User-defined control registers				
SPI_CMD_REG	Command control register	0x0000	0x0000	varies
SPI_ADDR_REG	Address value register	0x0004	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0010	0x0010	varies
SPI_USER1_REG	SPI USER control register 1	0x0014	0x0014	R/W
SPI_USER2_REG	SPI USER control register 2	0x0018	0x0018	R/W
Control and configuration registers				
SPI_CTRL_REG	SPI control register	0x0008	0x0008	R/W
SPI_MS_DLEN_REG	SPI data bit length control register	0x001C	0x001C	R/W
SPI_MISC_REG	SPI misc register	0x0020	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA control register	0x0030	0x0030	varies
SPI_SLAVE_REG	SPI slave control register	0x00E0	0x00E0	varies
SPI_SLAVE1_REG	SPI slave control register 1	0x00E4	0x00E4	R/W/SS
Clock control registers				
SPI_CLOCK_REG	SPI clock control register	0x000C	0x000C	R/W

Name	Description	SPI2 Address	SPI3 Address	Access
SPI_CLK_GATE_REG	SPI module clock and register clock control	0x00E8	0x00E8	R/W
Timing registers				
SPI_DIN_MODE_REG	SPI input delay mode configuration	0x0024	0x0024	R/W
SPI_DIN_NUM_REG	SPI input delay number configuration	0x0028	0x0028	R/W
SPI_DOUT_MODE_REG	SPI output delay mode configuration	0x002C	0x002C	R/W
Interrupt registers				
SPI_DMA_INT_ENA_REG	SPI interrupt enable register	0x0034	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI interrupt clear register	0x0038	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI interrupt raw register	0x003C	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI interrupt status register	0x0040	0x0040	RO
SPI_DMA_INT_SET_REG	SPI interrupt software set register	0x0044	0x0044	WT
CPU-controlled data buffer				
SPI_W0_REG	SPI CPU-controlled buffer0	0x0098	0x0098	R/W/SS
SPI_W1_REG	SPI CPU-controlled buffer1	0x009C	0x009C	R/W/SS
SPI_W2_REG	SPI CPU-controlled buffer2	0x00A0	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU-controlled buffer3	0x00A4	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU-controlled buffer4	0x00A8	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU-controlled buffer5	0x00AC	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU-controlled buffer6	0x00B0	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU-controlled buffer7	0x00B4	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU-controlled buffer8	0x00B8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU-controlled buffer9	0x00BC	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU-controlled buffer10	0x00C0	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU-controlled buffer11	0x00C4	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU-controlled buffer12	0x00C8	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU-controlled buffer13	0x00CC	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU-controlled buffer14	0x00D0	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU-controlled buffer15	0x00D4	0x00D4	R/W/SS
Version register				
SPI_DATE_REG	Version control	0x00F0	0x00F0	R/W

30.12 Registers

The addresses in this section are relative to SPI2/SPI3 base address provided in Table 4.3-3 in Chapter 4 *System and Memory*.

Register 30.1. SPI_CMD_REG (0x0000)

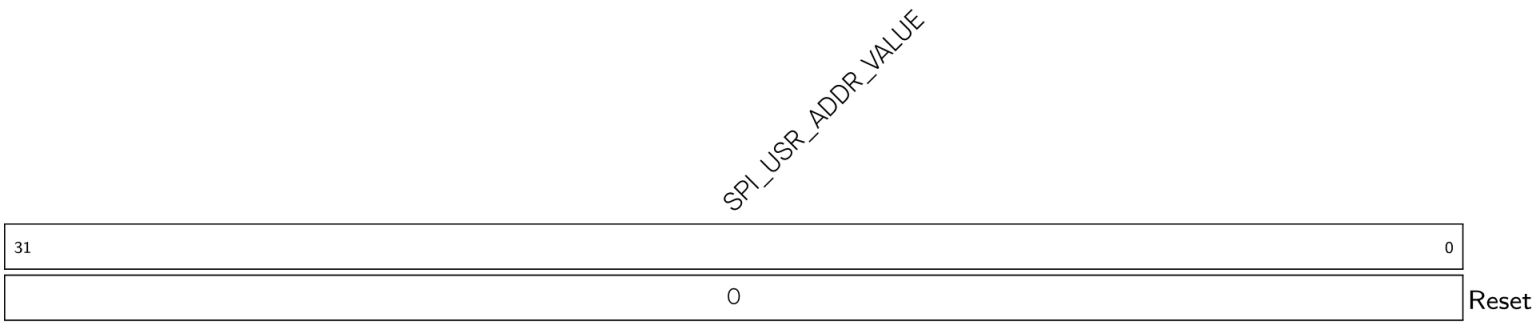


SPI_CONF_BITLEN (for SPI2 only) Define the cycles of APB_CLK in CONF state. Can be configured in CONF state. (R/W)

SPI_UPDATE Set this bit to synchronize SPI registers from APB clock domain into SPI module clock domain. This bit is only used in SPI master mode. (WT)

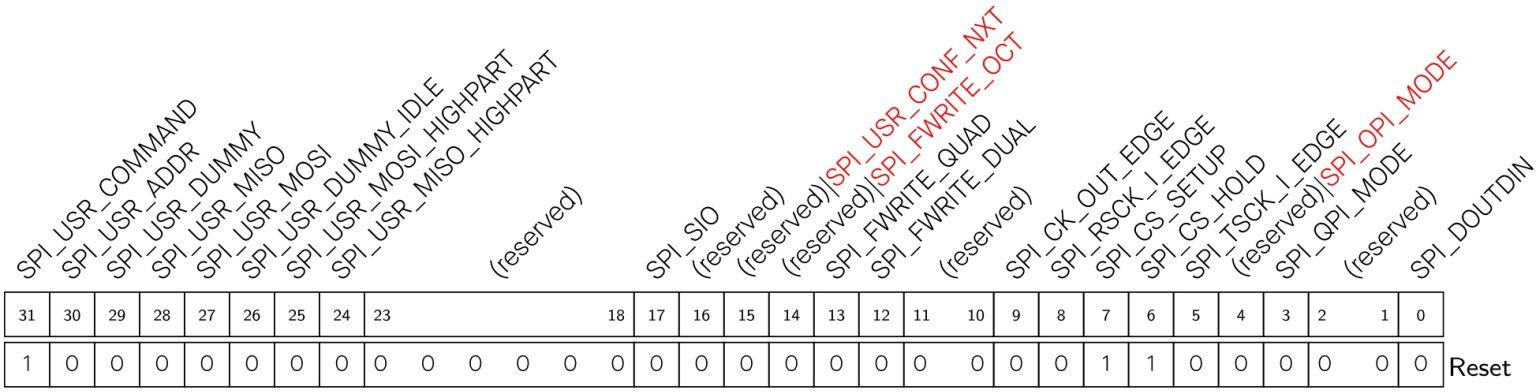
SPI_USR User-defined command enable. An SPI operation will be triggered when the bit is set. The bit will be cleared once the operation is done. 1: enable. 0: disable. Can not be changed by CONF_buf. (R/W/SC)

Register 30.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE Address to slave. Can be configured in CONF state. (R/W)

Register 30.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN Set the bit to enable full-duplex communication. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_QPI_MODE 1: Enable QPI mode. 0: Disable QPI mode. This configuration is applicable when the SPI controller works as master or slave. Can be configured in CONF state. (R/W/SS/SC)

SPI_OPI_MODE (for SPI2 only) 1: Enable OPI mode (all in 8-bit mode). 0: Disable OPI mode. This configuration is applicable when the SPI controller works as master. Can be configured in CONF state. (R/W)

SPI_TSCK_I_EDGE In slave mode, this bit can be used to support four SPI clock modes, which is described in Subsection 30.7.3. (R/W)

SPI_CS_HOLD Keep SPI CS low when SPI is in DONE state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_CS_SETUP Enable SPI CS when SPI is in prepare (PREP) state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_RSCK_I_EDGE In slave mode, this bit can be used to support four SPI clock modes, which is described in Subsection 30.7.3. (R/W)

SPI_CK_OUT_EDGE This bit together with SPI_MOSI_DELAY_MODE is used to set MOSI signal delay mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_DUAL In write operations, read-data phase is in 2-bit mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_QUAD In write operations, read-data phase is in 4-bit mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_OCT (for SPI2 only) In write operations, read-data phase is in 8-bit mode. Can be configured in CONF state. (R/W)

Continued from the previous page...

SPI_USR_CONF_NXT (for SPI2 only) Enable the CONF state for the next transaction (segment) in a configurable segmented transfer. Can be configured in CONF state. (R/W)

- If this bit is set, it means this configurable segmented transfer will continue its next transaction (segment).
- If this bit is cleared, it means this transfer will end after the current transaction (segment) is finished. Or this is not a configurable segmented transfer.

SPI_SIO Set the bit to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_MISO_HIGHPART In read-data phase, only access to high-part of the buffers SPI_W8_REG ~ SPI_W15_REG. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_MOSI_HIGHPART In write-data phase, only access to high-part of the buffers SPI_W8_REG ~ SPI_W15_REG. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY_IDLE If this bit is set, SPI clock is disable in DUMMY state. Can be configured in CONF state. (R/W)

SPI_USR_MOSI Set this bit to enable the write-data (DOUT) state of an operation. Can be configured in CONF state. (R/W)

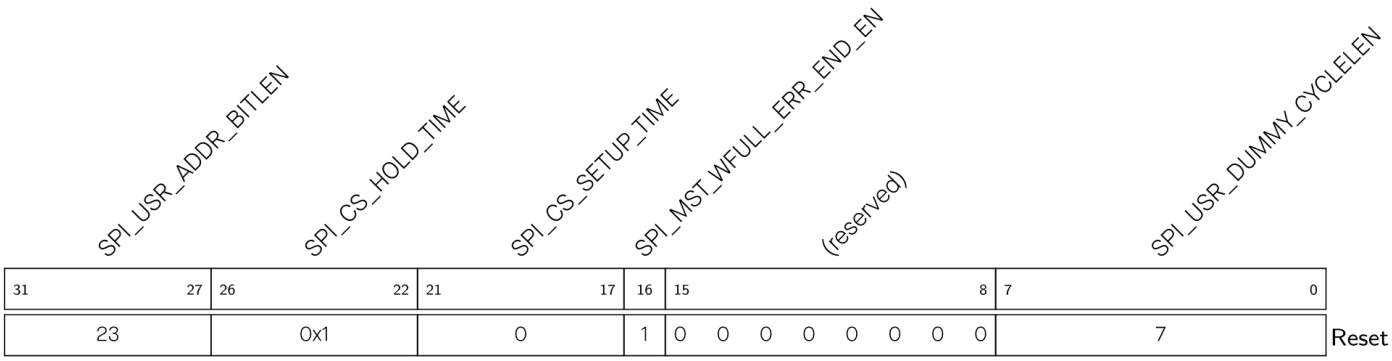
SPI_USR_MISO Set this bit to enable the read-data (DIN) state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY Set this bit to enable the DUMMY state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_ADDR Set this bit to enable the address (ADDR) state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_COMMAND Set this bit to enable the command (CMD) state of an operation. Can be configured in CONF state. (R/W)

Register 30.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN The length of DUMMY state, in unit of SPI_CLK cycles. The value is (the expected cycle number - 1). Can be configured in CONF state. (R/W)

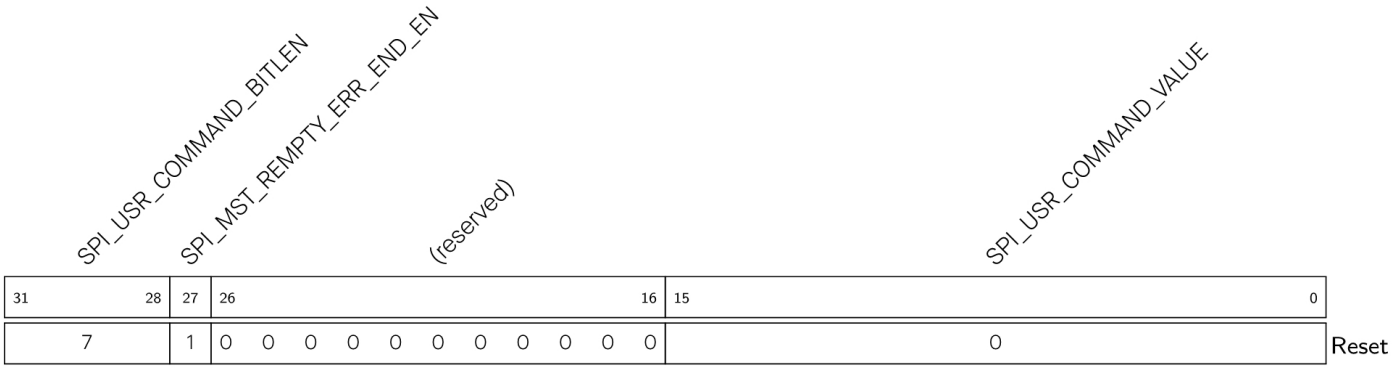
SPI_MST_WFULL_ERR_END_EN 1: SPI transfer is ended when SPI RX AFIFO wfull error occurs in GP-SPI master full-/half-duplex modes. 0: SPI transfer is not ended when SPI RX AFIFO wfull error occurs in GP-SPI master full-/half-duplex modes. (R/W)

SPI_CS_SETUP_TIME The length of prepare (PREP) state, in unit of SPI_CLK cycles. This value is equal to the expected cycles -1. This field is used together with SPI_CS_SETUP. Can be configured in CONF state. (R/W)

SPI_CS_HOLD_TIME Delay cycles of CS pin, in units of SPI_CLK cycles. This field is used together with SPI_CS_HOLD. Can be configured in CONF state. (R/W)

SPI_USR_ADDR_BITLEN The bit length in address state. This value is (expected bit number - 1). Can be configured in CONF state. (R/W)

Register 30.5. SPI_USER2_REG (0x0018)

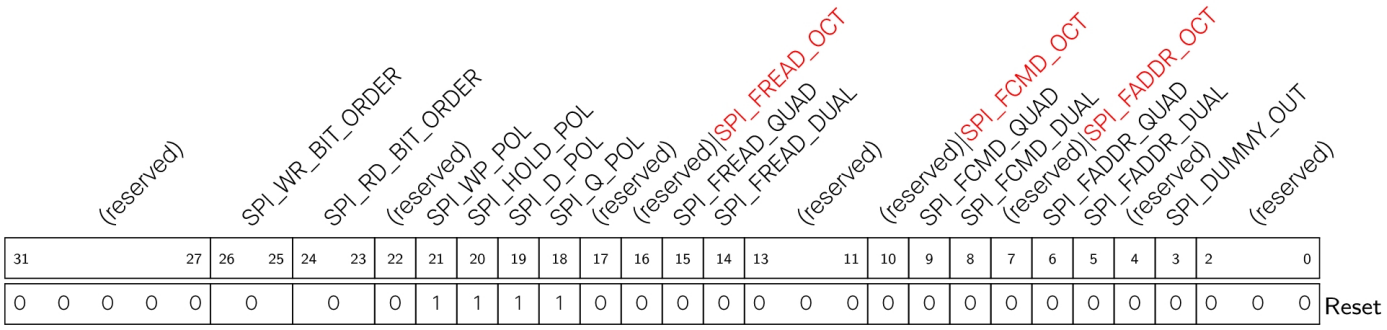


SPI_USR_COMMAND_VALUE The value of command. Can be configured in CONF state. (R/W)

SPI_MST_REMPTY_ERR_END_EN 1: SPI transfer is ended when SPI TX AFIFO read empty error occurs in GP-SPI master full-/half-duplex modes. 0: SPI transfer is not ended when SPI TX AFIFO read empty error occurs in GP-SPI master full-/half-duplex modes. (R/W)

SPI_USR_COMMAND_BITLEN The bit length of command state. This value is (expected bit number - 1). Can be configured in CONF state. (R/W)

Register 30.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT Can be configured in CONF state. (R/W)

- For SPI2:
  - 0: In DUMMY state, the FSPI bus signals are not output.
  - 1: In DUMMY state, the FSPI bus signals are output.
- For SPI3: In DUMMY state, the signal level of SPI is output by the SPI controller.

SPI_FADDR_DUAL Apply 2-bit mode during address (ADDR) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FADDR_QUAD Apply 4-bit mode during address (ADDR) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FADDR_OCT (for SPI2 only) Apply 8-bit mode during address (ADDR) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_DUAL Apply 2-bit mode during command (CMD) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_QUAD Apply 4-bit mode during command (CMD) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_OCT (for SP2 only) Apply 8-bit mode during command (CMD) state. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_DUAL In read operations, read-data (DIN) state is in 2-bit mode. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_QUAD In read operations, read-data (DIN) state is in 4-bit mode. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_OCT (for SP2 only) In read operations, read-data (DIN) state is in 8-bit mode. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_Q_POL The bit is used to set MISO line polarity. 1: high. 0: low. Can be configured in CONF state. (R/W)

SPI_D_POL The bit is used to set MOSI line polarity, 1: high. 0, low. Can be configured in CONF state. (R/W)

Register 30.6. SPI_CTRL_REG (0x0008)

Continued from the previous page...

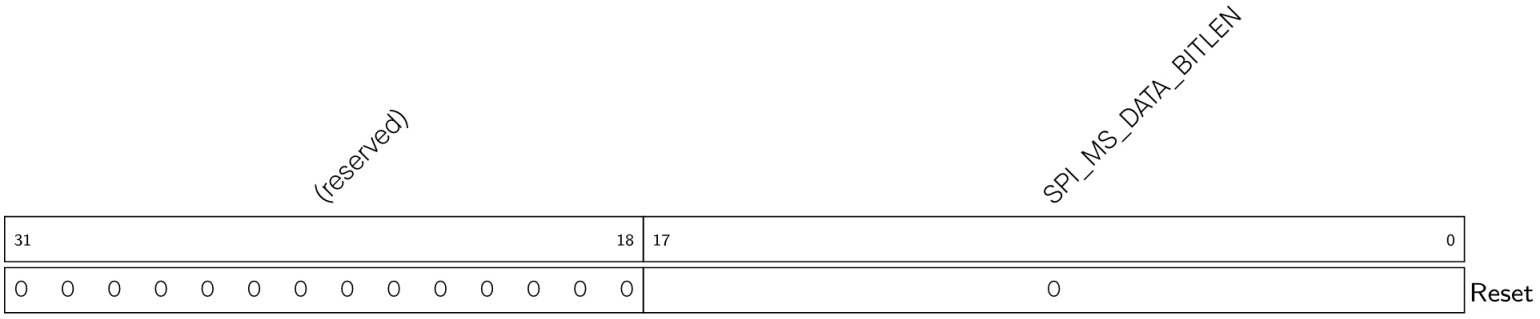
SPI_HOLD_POL This bit is used to set SPI_HOLD output value when SPI is in idle. 1: output high. 0: output low. Can be configured in CONF state. (R/W)

SPI_WP_POL This bit is used to set the output value of write-protect signal when SPI is in idle. 1: output high. 0: output low. Can be configured in CONF state. (R/W)

SPI_RD_BIT_ORDER In read-data (MISO) state, 1: LSB first. 0: MSB first. Can be configured in CONF state. (R/W)

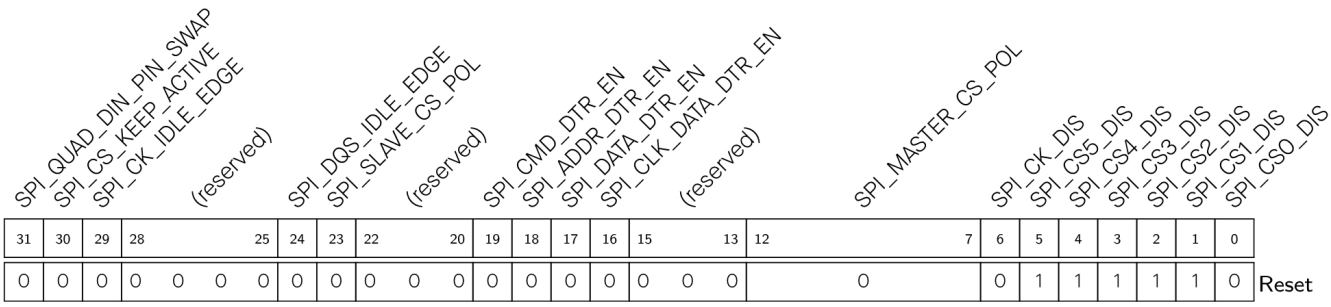
SPI_WR_BIT_ORDER In command (CMD), address (ADDR), and write-data (MOSI) states, 1: LSB first. 0: MSB first. Can be configured in CONF state. (R/W)

Register 30.7. SPI_MS_DLEN_REG (0x001C)



SPI_MS_DATA_BITLEN The value of this field is the configured SPI transmission data bit length in master mode DMA-controlled transfer or CPU-controlled transfer. The value is also the configured bit length in slave mode DMA RX controlled transfer. The register value shall be (expected bit number - 1). Can be configured in CONF state. (R/W)

Register 30.8. SPI_MISC_REG (For SPI2 Only) (0x0020)



SPI_CS0_DIS SPI CS0 pin enable bit. 1: disable CS0. 0: SPI_CS0 signal is from/to CS0 pin. Can be configured in CONF state. (R/W)

SPI_CS1_DIS SPI CS1 pin enable bit. 1: disable CS1. 0: SPI_CS1 signal is from/to CS1 pin. Can be configured in CONF state. (R/W)

SPI_CS2_DIS SPI CS2 pin enable bit. 1: disable CS2. 0: SPI_CS2 signal is from/to CS2 pin. Can be configured in CONF state. (R/W)

SPI_CS3_DIS SPI CS3 pin enable bit. 1: disable CS3. 0: SPI_CS3 signal is from/to CS3 pin. Can be configured in CONF state. (R/W)

SPI_CS4_DIS SPI CS4 pin enable bit. 1: disable CS4. 0: SPI_CS4 signal is from/to CS4 pin. Can be configured in CONF state. (R/W)

SPI_CS5_DIS SPI CS5 pin enable bit. 1: disable CS5 0: SPI_CS5 signal is from/to CS5 pin. Can be configured in CONF state. (R/W)

SPI_CK_DIS 1: Disable SPI_CLK output. 0: Enable SPI_CLK output. Can be configured in CONF state. (R/W)

SPI_MASTER_CS_POL SPI_MASTER_CS_POL[*i*] configures the polarity of SPI CS*i* (*i* is from 0 ~ 2) line in master mode. 0: CS*i* is low active. 1: CS*i* is high active. Can be configured in CONF state. (R/W)

SPI_CLK_DATA_DTR_EN 1: SPI master DDR mode is applied to SPI clock, data, and SPI_DQS. 0: SPI master DDR mode is only applied to SPI_DQS. This bit should be used with bit 17/18/19. (R/W)

SPI_DATA_DTR_EN 1: SPI clock and data of DOUT and DIN states are in DDR mode, including master 1/2/4/8-bit mode. 0: SPI clock and data of DOUT and DIN states are in SDR mode. Can be configured in CONF state. (R/W)

SPI_ADDR_DTR_EN 1: SPI clock and data of SPI_SEND_ADDR state are in DDR mode, including master 1/2/4/8-bit mode. 0: SPI clock and data of SPI_SEND_ADDR state are in SDR mode. Can be configured in CONF state. (R/W)

SPI_CMD_DTR_EN 1: SPI clock and data of SPI_SEND_CMD state are in DDR mode, including master 1/2/4/8-bit mode. 0: SPI clock and data of SPI_SEND_CMD state are in SDR mode. Can be configured in CONF state. (R/W)

SPI_SLAVE_CS_POL Configure SPI slave input CS polarity. 1: invert. 0: not change. Can be configured in CONF state. (R/W)

Register 30.8. SPI_MISC_REG (For SPI2 Only) (0x0020)

Continued from the previous page...

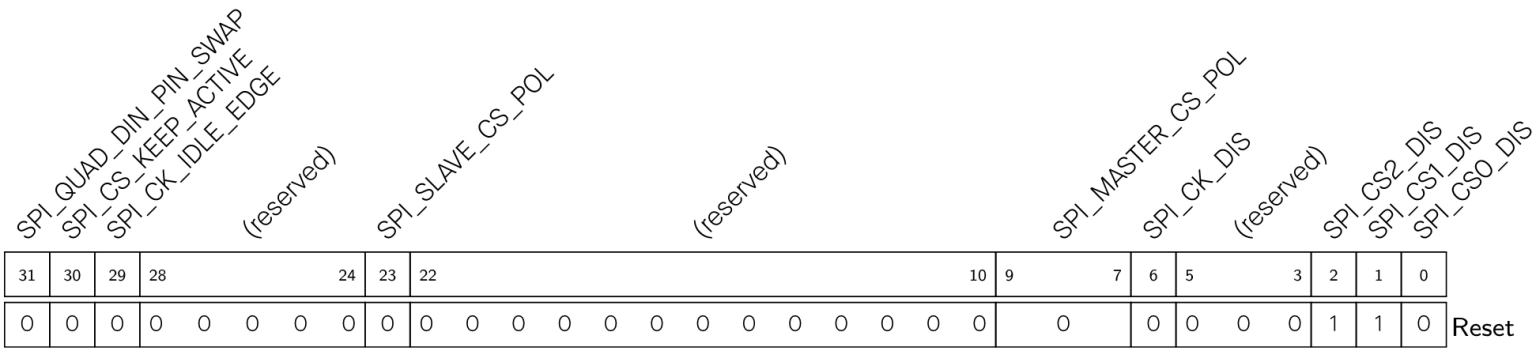
SPI_DQS_IDLE_EDGE The default value of SPI_DQS. 0: low voltage level; 1: high voltage level. Can be configured in CONF state. (R/W)

SPI_CK_IDLE_EDGE 1: SPI_CLK line is high when GP-SPI2 is in idle. 0: SPI_CLK line is low when GP-SPI2 is in idle. Can be configured in CONF state. (R/W)

SPI_CS_KEEP_ACTIVE SPI CS line keeps low when the bit is set. Can be configured in CONF state. (R/W)

SPI_QUAD_DIN_PIN_SWAP SPI quad input swap enable. 1: swap FSPID with FSPIQ, swap FSPIWP with FSPIHD. 0: SPI quad input swap disable. Can be configured in CONF state. (R/W)

Register 30.9. SPI_MISC_REG (For SPI3 Only) (0x0020)



SPI_CS0_DIS SPI CS0 pin enable bit. 1: disable CS0. 0: SPI_CS0 signal is from/to CS0 pin. Can be configured in CONF state. (R/W)

SPI_CS1_DIS SPI CS1 pin enable bit. 1: disable CS1. 0: SPI_CS1 signal is from/to CS1 pin. Can be configured in CONF state. (R/W)

SPI_CS2_DIS SPI CS2 pin enable bit. 1: disable CS2. 0: SPI_CS2 signal is from/to CS2 pin. Can be configured in CONF state. (R/W)

SPI_CK_DIS 1: Disable SPI_CLK output. 0: Enable SPI_CLK output. Can be configured in CONF state. (R/W)

SPI_MASTER_CS_POL SPI_MASTER_CS_POL[*i*] configures the polarity of SPI CS*i* (*i* is from 0 ~ 5) line in master mode. 0: CS*i* is low active. 1: CS*i* is high active. Can be configured in CONF state. (R/W)

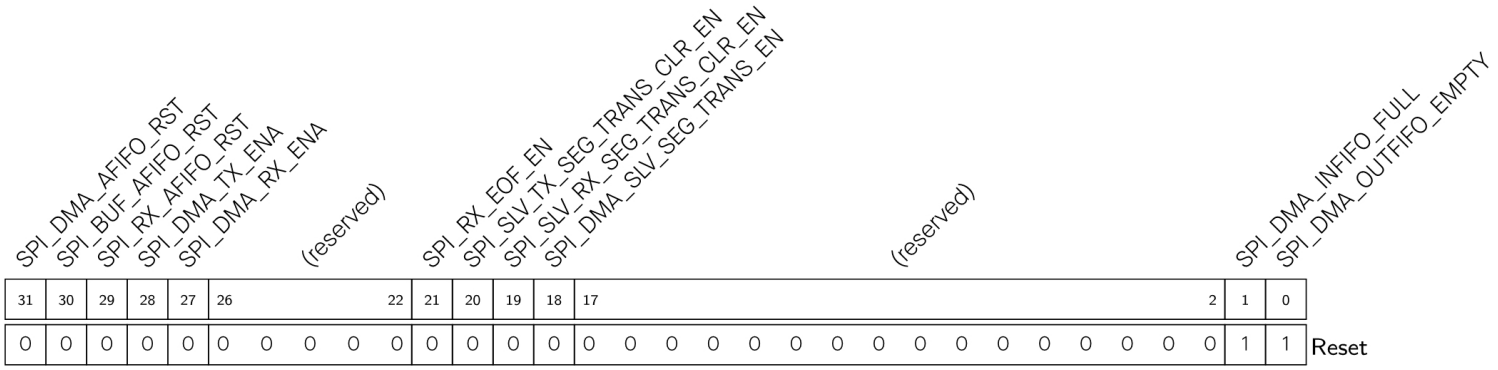
SPI_SLAVE_CS_POL Configure SPI slave input CS polarity. 1: invert. 0: not change. Can be configured in CONF state. (R/W)

SPI_CK_IDLE_EDGE 1: SPI_CLK line is high when GP-SPI3 is in idle. 0: SPI_CLK line is low when GP-SPI3 is in idle. Can be configured in CONF state. (R/W)

SPI_CS_KEEP_ACTIVE SPI CS line keeps low when the bit is set. Can be configured in CONF state. (R/W)

SPI_QUAD_DIN_PIN_SWAP 1: SPI quad input swap enable. 0: SPI quad input swap disable. Can be configured in CONF state. (R/W)

Register 30.10. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY Records the status of DMA TX FIFO. 1: DMA TX FIFO is not ready for sending data. 0: DMA TX FIFO is ready for sending data. (RO)

SPI_DMA_INFIFO_FULL Records the status of DMA RX FIFO. 1: DMA RX FIFO is not ready for receiving data. 0: DMA RX FIFO is ready for receiving data. (RO)

SPI_DMA_SLV_SEG_TRANS_EN 1: Enable DMA-controlled segmented transfer in slave half-duplex mode. 0: disable. (R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN In DMA-controlled half-duplex slave mode, if the size of DMA RX buffer is smaller than the size of the received data, 1: the data in following transfers will not be received. 0: the data in this transfer will not be received, but in the following transfers, if the size of DMA RX buffer is not 0, the data in following transfers will be received, otherwise not. (R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN In DMA-controlled half-duplex slave mode, if the size of DMA TX buffer is smaller than the size of the transmitted data, 1: the data in the following transfers will not be updated, i.e., the old data is transmitted repeatedly. 0: the data in this transfer will not be updated. But in the following transfers, if new data is filled in DMA TX FIFO, new data will be transmitted, otherwise not. (R/W)

SPI_RX_EOF_EN 1: In a DAM-controlled transfer, if the bit number of transferred data is equal to (SPI_MS_DATA_BITLEN + 1), then GDMA_IN_SUC_EOF_CH*n*_INT_RAW will be set by hardware. 0: GDMA_IN_SUC_EOF_CH*n*_INT_RAW is set by SPI_TRANS_DONE_INT event in a non-segmented transfer, or by a SPI_DMA_SEG_TRANS_DONE_INT event in a segmented transfer. (R/W)

SPI_DMA_RX_ENA Set this bit to enable DMA-controlled receive data mode. (R/W)

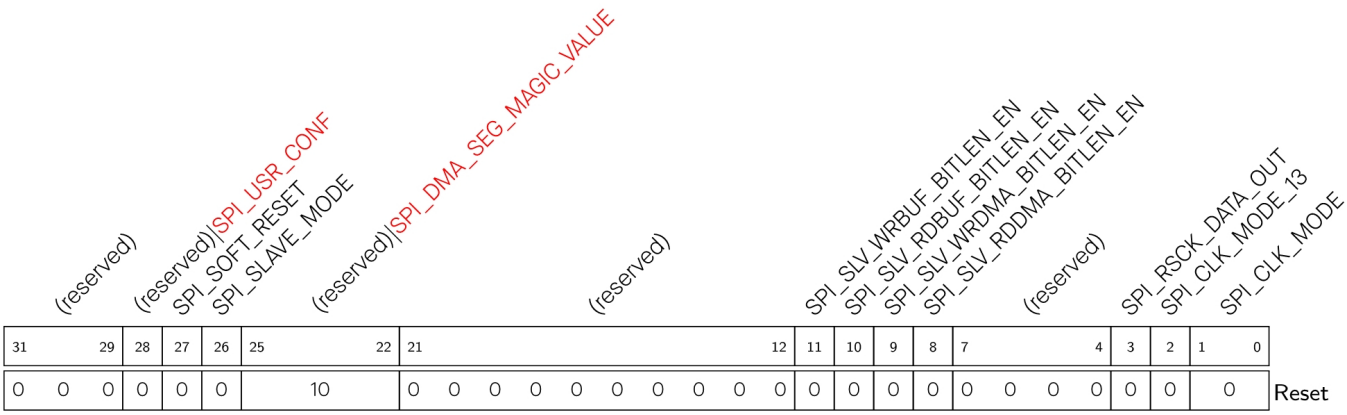
SPI_DMA_TX_ENA Set this bit to enable DMA-controlled send data mode. (R/W)

SPI_RX_AFIFO_RST Set this bit to reset spi_rx_afifo as shown in Figure 30.5-3 and in Figure 30.5-4. spi_rx_afifo is used to receive data in SPI master and slave transfer. (WT)

SPI_BUF_AFIFO_RST Set this bit to reset buf_tx_afifo as shown in Figure 30.5-3 and in Figure 30.5-4. buf_tx_afifo is used to send data out in CPU-controlled master and slave transfer. (WT)

SPI_DMA_AFIFO_RST Set this bit to reset dma_tx_afifo as shown in Figure 30.5-3 and in Figure 30.5-4. dma_tx_afifo is used to send data out in DMA-controlled slave transfer. (WT)

Register 30.11. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE SPI clock mode control bits. Can be configured in CONF state. (R/W)

- 0: SPI clock is off when CS becomes inactive.
- 1: SPI clock is delayed one cycle after CS becomes inactive.
- 2: SPI clock is delayed two cycles after CS becomes inactive.
- 3: SPI clock is always on.

SPI_CLK_MODE_13 Configure clock mode. (R/W)

- 1: support SPI clock mode 1 and 3. Output data B[0]/B[7] at the first edge.
- 0: support SPI clock mode 0 and 2. Output data B[1]/B[6] at the first edge.

SPI_RSCK_DATA_OUT Save half a cycle when TSCK is the same as RSCK. 1: output data at RSCK rising edge. 0: output data at TSCK rising edge. (R/W)

SPI_SLV_RDDMA_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store the data bit length of Rd_DMA transfer. (R/W)

SPI_SLV_WRDMA_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store the data bit length of Wr_DMA transfer. (R/W)

SPI_SLV_RDBUF_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store data bit length of Rd_BUF transfer. (R/W)

SPI_SLV_WRBUF_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store data bit length of Wr_BUF transfer. (R/W)

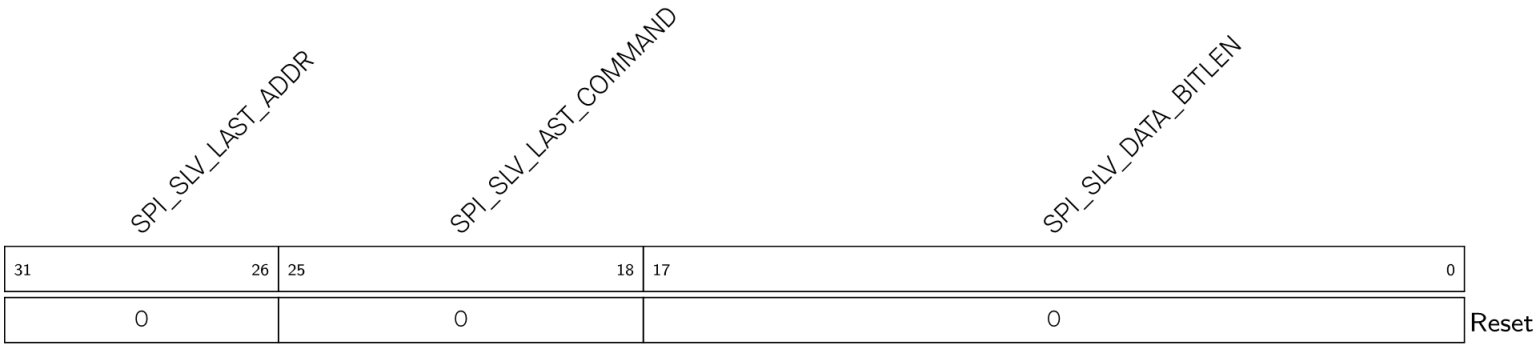
SPI_DMA_SEG_MAGIC_VALUE (for SPI2 only) Configure the magic value of BM table in DMA-controlled configurable segmented transfer. (R/W)

SPI_SLAVE_MODE Set SPI work mode. 1: slave mode. 0: master mode. (R/W)

SPI_SOFT_RESET Software reset enable bit. If this bit is set, the SPI clock line, CS line, and data line are reset. Can be configured in CONF state. (WT)

SPI_USR_CONF (for SPI2 only) 1: Enable the CONF state of current DMA-controlled configurable segmented transfer, which means the configurable segmented transfer is started. 0: This is not a configurable segmented transfer. (R/W)

Register 30.12. SPI_SLAVE1_REG (0x00E4)

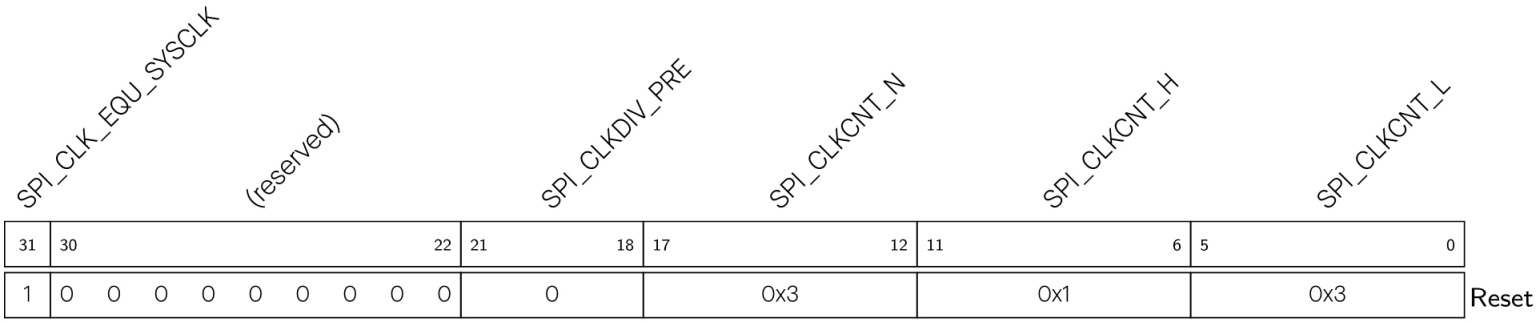


SPI_SLV_DATA_BITLEN Configure the transferred data bit length in SPI slave full-/half-duplex modes. (R/W/SS)

SPI_SLV_LAST_COMMAND In slave mode, it is the value of command. (R/W/SS)

SPI_SLV_LAST_ADDR In slave mode, it is the value of address. (R/W/SS)

Register 30.13. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L In master mode, this field must be equal to SPI_CLKCNT_N. In slave mode, it must be 0. Can be configured in CONF state. (R/W)

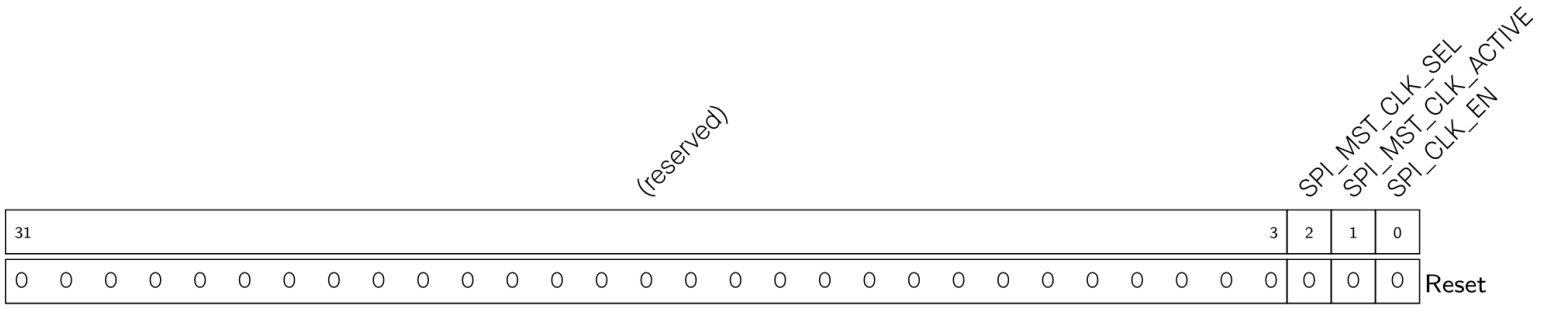
SPI_CLKCNT_H In master mode, this field must be floor((SPI_CLKCNT_N + 1)/2 - 1). floor() here is to down round a number, floor(2.2) = 2. In slave mode, it must be 0. Can be configured in CONF state. (R/W)

SPI_CLKCNT_N In master mode, this is the divider of SPI_CLK. So SPI_CLK frequency is f_{apb_clk}/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1). Can be configured in CONF state. (R/W)

SPI_CLKDIV_PRE In master mode, this is pre-divider of SPI_CLK. Can be configured in CONF state. (R/W)

SPI_CLK_EQU_SYSCLK In master mode, 1: SPI_CLK is equal to APB_CLK. 0: SPI_CLK is divided from APB_CLK. Can be configured in CONF state. (R/W)

Register 30.14. SPI_CLK_GATE_REG (0x00E8)



SPI_CLK_EN Set this bit to enable clock gate. (R/W)

SPI_MST_CLK_ACTIVE Set this bit to power on the SPI module clock. (R/W)

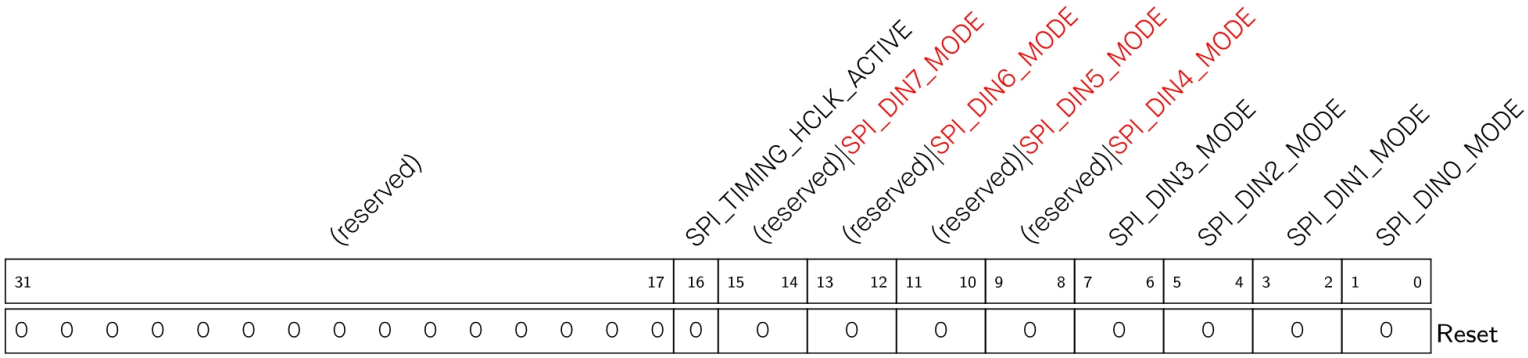
SPI_MST_CLK_SEL This bit is used to select SPI module clock source in master mode.

1: PLL_F80M_CLK

0: XTAL_CLK

(R/W)

Register 30.15. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE Configure the input mode for input data bit0 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN0_NUM + 1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN0_NUM + 1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN0_NUM + 1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_DIN1_MODE Configure the input mode for input data bit1 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN1_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN1_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN1_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_DIN2_MODE Configure the input mode for input data bit2 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN2_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN2_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN2_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

Continued from the previous page...

SPI_DIN3_MODE Configure the input mode for input data bit3 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN3_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN3_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN3_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_DIN4_MODE (for SPI2 only) Configure the input mode for input data bit4 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN4_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN4_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN4_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_DIN5_MODE (for SPI2 only) Configure the input mode for input data bit5 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN5_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN5_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN5_NUM + 1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_DIN6_MODE (for SPI2 only) Configure the input mode for input data bit6 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN6_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN6_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN6_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

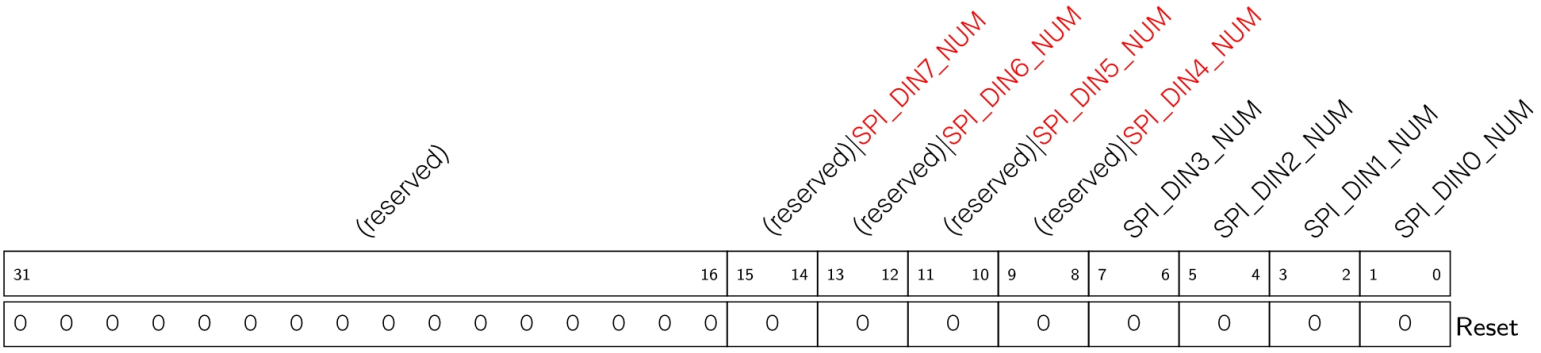
Continued from the previous page...

SPI_DIN7_MODE (for SPI2 only) Configure the input mode for input data bit7 signal. Can be configured in CONF state. (R/W)

- 0: input without delay
- 1: input data is delayed by the falling edge of SPI_CLK for (SPI_DIN7_NUM +1) cycles
- 2: input data is delayed by the rising edge of clk_hclk for (SPI_DIN7_NUM +1) cycles, and then delayed by the rising edge of SPI_CLK for one cycle
- 3: input data is delayed by the rising edge of clk_hclk for (SPI_DIN7_NUM +1) cycles, and then delayed by the falling edge of SPI_CLK for one cycle

SPI_TIMING_HCLK_ACTIVE 1: Enable clk_hclk (high-frequency clock) in SPI input timing module. 0: disable clk_hclk. Can be configured in CONF state. (R/W)

Register 30.16. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM Configure the delays to input data bit0 signal based on the setting of SPI_DIN0_MODE. Can be configured in CONF state. (R/W)

SPI_DIN1_NUM Configure the delays to input data bit1 signal based on the setting of SPI_DIN1_MODE. Can be configured in CONF state. (R/W)

SPI_DIN2_NUM Configure the delays to input data bit2 signal based on the setting of SPI_DIN2_MODE. Can be configured in CONF state. (R/W)

SPI_DIN3_NUM Configure the delays to input data bit3 signal based on the setting of SPI_DIN3_MODE. Can be configured in CONF state. (R/W)

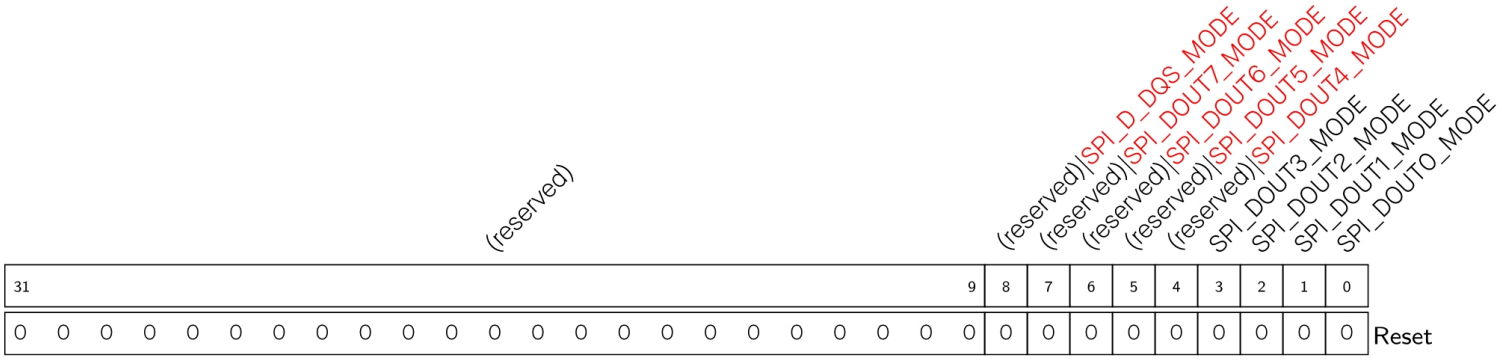
SPI_DIN4_NUM (for SPI2 only)) Configure the delays to input data bit4 signal based on the setting of SPI_DIN4_MODE. Can be configured in CONF state. (R/W)

SPI_DIN5_NUM (for SPI2 only) Configure the delays to input data bit5 signal based on the setting of SPI_DIN5_MODE. Can be configured in CONF state. (R/W)

SPI_DIN6_NUM (for SPI2 only) Configure the delays to input data bit6 signal based on the setting of SPI_DIN6_MODE. Can be configured in CONF state. (R/W)

SPI_DIN7_NUM (for SPI2 only) Configure the delays to input data bit7 signal based on the setting of SPI_DIN7_MODE. Can be configured in CONF state. (R/W)

Register 30.17. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE Configure the output mode for output data bit0 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_DOUT1_MODE Configure the output mode for output data bit1 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_DOUT2_MODE Configure the output mode for output data bit2 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_DOUT3_MODE Configure the output mode for output data bit3 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_DOUT4_MODE (for SPI2 only) Configure the output mode for output data bit4 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_DOUT5_MODE (for SPI2 only) Configure the output mode for output data bit5 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

Continued from the previous page...

SPI_DOUT6_MODE (for SPI2 only) Configure the output mode for output data bit6 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

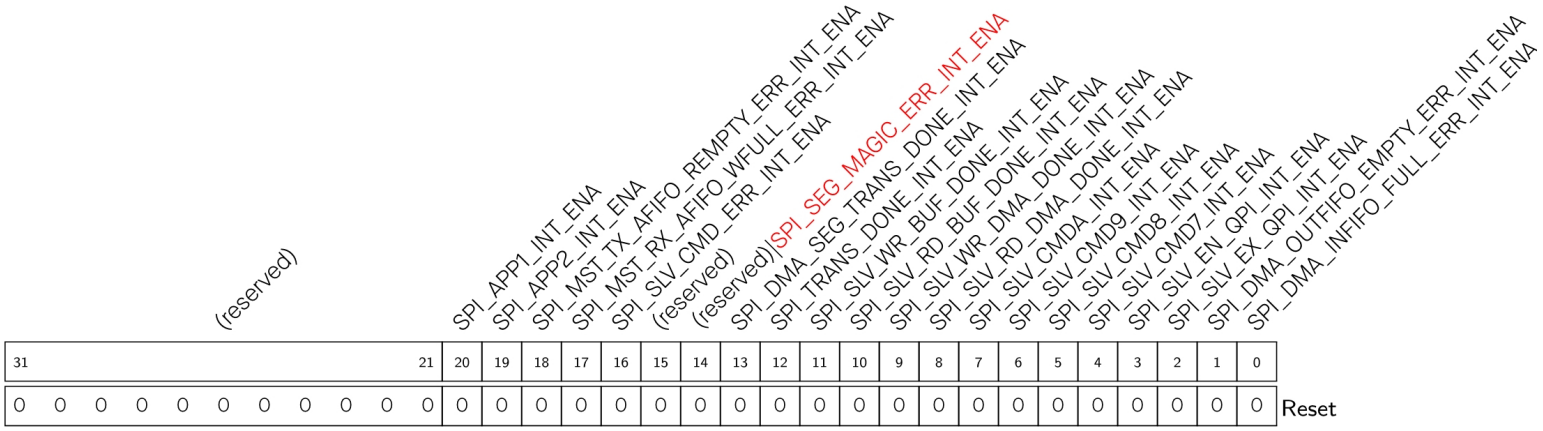
SPI_DOUT7_MODE (for SPI2 only) Configure the output mode for output data bit7 signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

SPI_D_DQS_MODE (for SPI2 only) Configure the output mode for output SPI_DQS signal. Can be configured in CONF state. (R/W)

- 0: output without delay
- 1: output data is delayed by the falling edge of SPI_CLK for one cycle

Register 30.18. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA The enable bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA The enable bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (R/W)

SPI_SLV_EX_QPI_INT_ENA The enable bit for SPI_SLV_EX_QPI_INT interrupt. (R/W)

SPI_SLV_EN_QPI_INT_ENA The enable bit for SPI_SLV_EN_QPI_INT interrupt. (R/W)

SPI_SLV_CMD7_INT_ENA The enable bit for SPI_SLV_CMD7_INT interrupt. (R/W)

SPI_SLV_CMD8_INT_ENA The enable bit for SPI_SLV_CMD8_INT interrupt. (R/W)

SPI_SLV_CMD9_INT_ENA The enable bit for SPI_SLV_CMD9_INT interrupt. (R/W)

SPI_SLV_CMDA_INT_ENA The enable bit for SPI_SLV_CMDA_INT interrupt. (R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA The enable bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA The enable bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA The enable bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA The enable bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (R/W)

SPI_TRANS_DONE_INT_ENA The enable bit for SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA The enable bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W)

SPI_SEG_MAGIC_ERR_INT_ENA (for SPI2 only) The enable bit for SPI_SEG_MAGIC_ERR_INT interrupt. (R/W)

SPI_SLV_CMD_ERR_INT_ENA The enable bit for SPI_SLV_CMD_ERR_INT interrupt. (R/W)

Continued from the previous page...

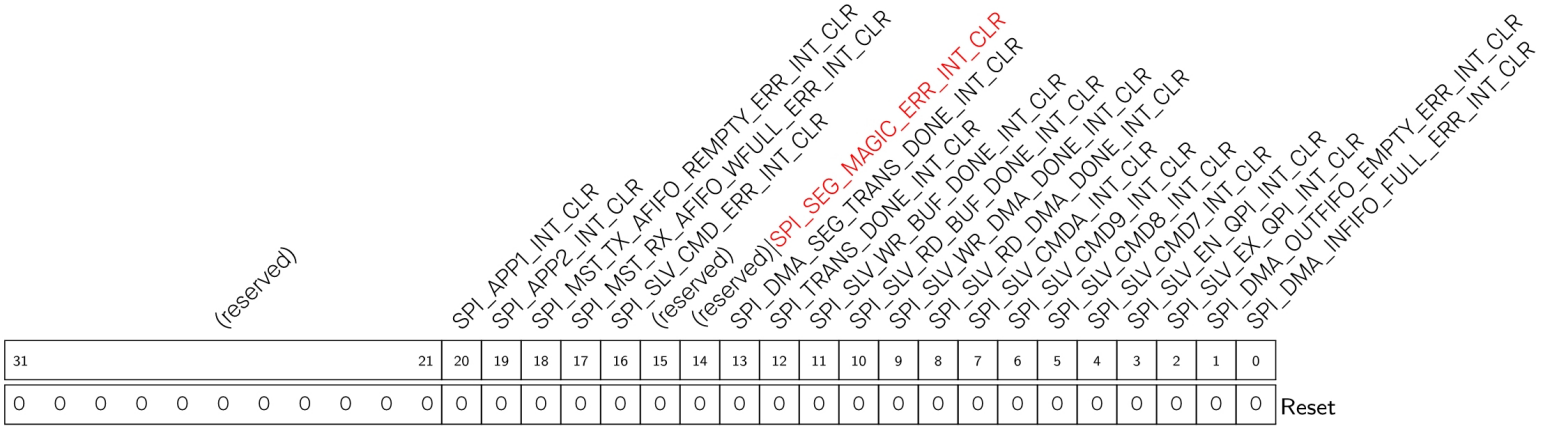
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA The enable bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA The enable bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

SPI_APP2_INT_ENA The enable bit for SPI_APP2_INT interrupt. (R/W)

SPI_APP1_INT_ENA The enable bit for SPI_APP1_INT interrupt. (R/W)

Register 30.19. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR The clear bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR The clear bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INTinterrupt. (WT)

SPI_SLV_EX_QPI_INT_CLR The clear bit for SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_CLR The clear bit for SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_CLR The clear bit for SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_CLR The clear bit for SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_CLR The clear bit for SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_CLR The clear bit for SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR The clear bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR The clear bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR The clear bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR The clear bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_CLR The clear bit for SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR The clear bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_CLR (for SPI2 only) The clear bit for SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_CLR The clear bit for SPI_SLV_CMD_ERR_INT interrupt. (WT)

Continued from the previous page...

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR The clear bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR The clear bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_CLR The clear bit for SPI_APP2_INT interrupt. (WT)

SPI_APP1_INT_CLR The clear bit for SPI_APP1_INT interrupt. (WT)

Register 30.20. SPI_DMA_INT_RAW_REG (0x003C)

SPI_DMA_INFIFO_FULL_ERR_INT_RAW The raw bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW The raw bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INTinterrupt. (R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW The raw bit for SPI_SLV_EX_QPI_INT interrupt. (R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW The raw bit for SPI_SLV_EN_QPI_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD7_INT_RAW The raw bit for SPI_SLV_CMD7_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD8_INT_RAW The raw bit for SPI_SLV_CMD8_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD9_INT_RAW The raw bit for SPI_SLV_CMD9_INT interrupt. (R/WTC/SS)

SPI_SLV_CMDA_INT_RAW The raw bit for SPI_SLV_CMDA_INT interrupt. (R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW The raw bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW The raw bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW The raw bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (R/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW The raw bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (R/WTC/SS)

SPI_TRANS_DONE_INT_RAW The raw bit for SPI_TRANS_DONE_INT interrupt. (R/WTC/SS)

Register 30.20. SPI_DMA_INT_RAW_REG (0x003C)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_RAW The raw bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW (for SPI2 only) The raw bit for SPI_SEG_MAGIC_ERR_INT interrupt. (R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW The raw bit for SPI_SLV_CMD_ERR_INT interrupt. (R/WTC/SS)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/WTC/SS)

SPI_APP2_INT_RAW The raw bit for SPI_APP2_INT interrupt. The value is only controlled by software. (R/WTC/SS)

SPI_APP1_INT_RAW The raw bit for SPI_APP1_INT interrupt. The value is only controlled by software. (R/WTC/SS)

Register 30.21. SPI_DMA_INT_ST_REG (0x0040)

SPI_DMA_INFIFO_FULL_ERR_INT_ST The status bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST The status bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (RO)

SPI_SLV_EX_QPI_INT_ST The status bit for SPI_SLV_EX_QPI_INT interrupt. (RO)

SPI_SLV_EN_QPI_INT_ST The status bit for SPI_SLV_EN_QPI_INT interrupt. (RO)

SPI_SLV_CMD7_INT_ST The status bit for SPI_SLV_CMD7_INT interrupt. (RO)

SPI_SLV_CMD8_INT_ST The status bit for SPI_SLV_CMD8_INT interrupt. (RO)

SPI_SLV_CMD9_INT_ST The status bit for SPI_SLV_CMD9_INT interrupt. (RO)

SPI_SLV_CMDA_INT_ST The status bit for SPI_SLV_CMDA_INT interrupt. (RO)

SPI_SLV_RD_DMA_DONE_INT_ST The status bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (RO)

SPI_SLV_WR_DMA_DONE_INT_ST The status bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (RO)

SPI_SLV_RD_BUF_DONE_INT_ST The status bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (RO)

SPI_SLV_WR_BUF_DONE_INT_ST The status bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (RO)

SPI_TRANS_DONE_INT_ST The status bit for SPI_TRANS_DONE_INT interrupt. (RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST The status bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (RO)

SPI_SEG_MAGIC_ERR_INT_ST (for SPI2 only) The status bit for SPI_SEG_MAGIC_ERR_INT interrupt. (RO)

SPI_SLV_CMD_ERR_INT_ST The status bit for SPI_SLV_CMD_ERR_INT interrupt. (RO)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The status bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (RO)

Register 30.21. SPI_DMA_INT_ST_REG (0x0040)

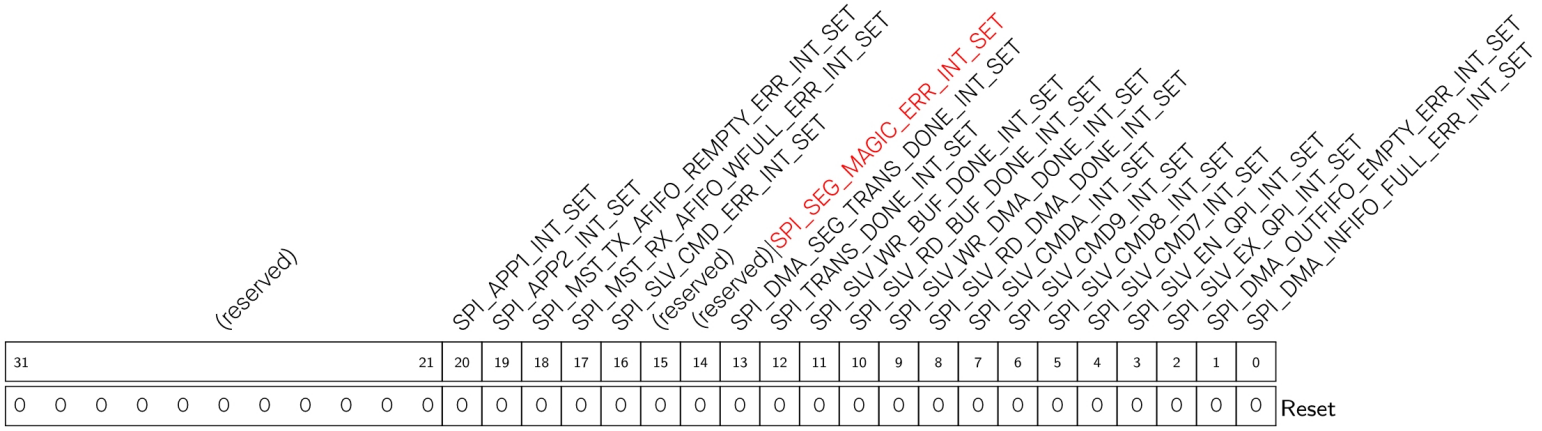
Continued from the previous page...

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The status bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (RO)

SPI_APP2_INT_ST The status bit for SPI_APP2_INT interrupt. (RO)

SPI_APP1_INT_ST The status bit for SPI_APP1_INT interrupt. (RO)

Register 30.22. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET The software set bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET The software set bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_SET The software set bit for SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_SET The software set bit for SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_SET The software set bit for SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_SET The software set bit for SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_SET The software set bit for SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_SET The software set bit for SPI_SLV_CMDA_INT interrupt. (WT)

Register 30.22. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

SPI_SLV_RD_DMA_DONE_INT_SET The software set bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_SET The software set bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_SET The software set bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_SET The software set bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_SET The software set bit for SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET The software set bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_SET (for SPI2 only) The software set bit for SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_SET The software set bit for SPI_SLV_CMD_ERR_INT interrupt. (WT)

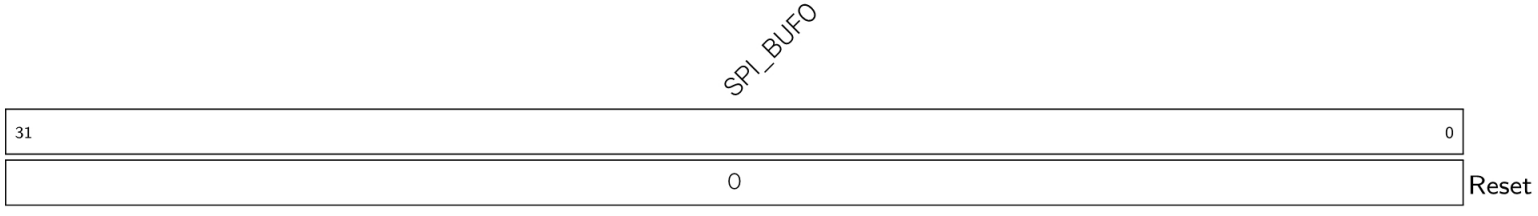
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET The software set bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET The software set bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_SET The software set bit for SPI_APP2_INT interrupt. (WT)

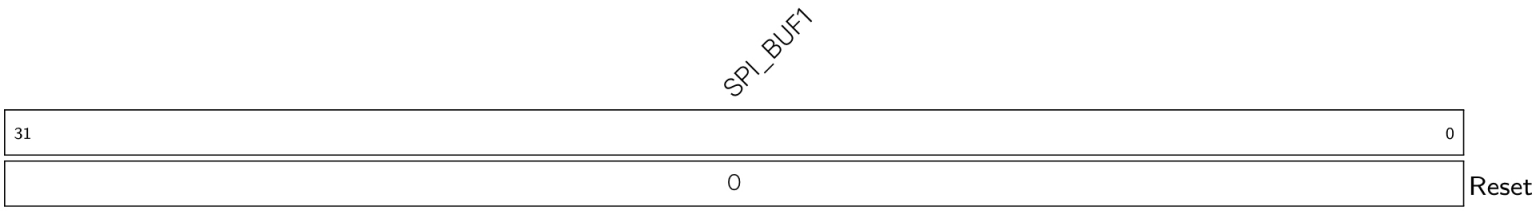
SPI_APP1_INT_SET The software set bit for SPI_APP1_INT interrupt. (WT)

Register 30.23. SPI_W0_REG (0x0098)



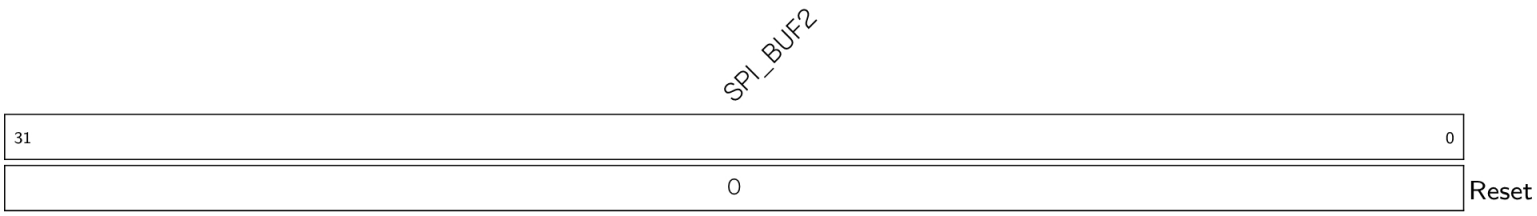
SPI_BUF0 32-bit data buffer 0. (R/W/SS)

Register 30.24. SPI_W1_REG (0x009C)



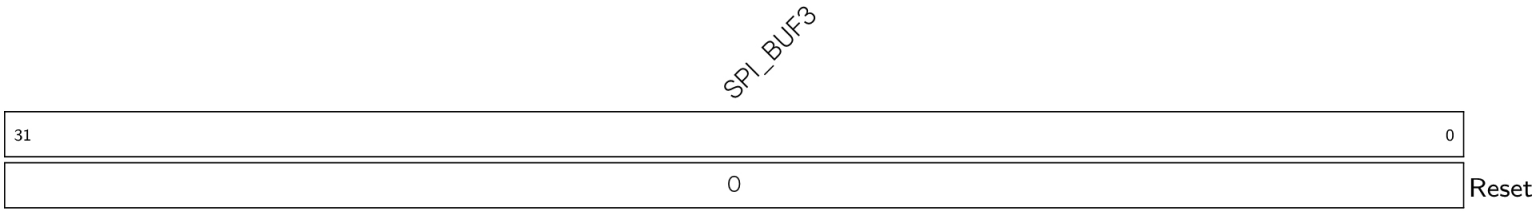
SPI_BUF1 32-bit data buffer 1. (R/W/SS)

Register 30.25. SPI_W2_REG (0x00A0)



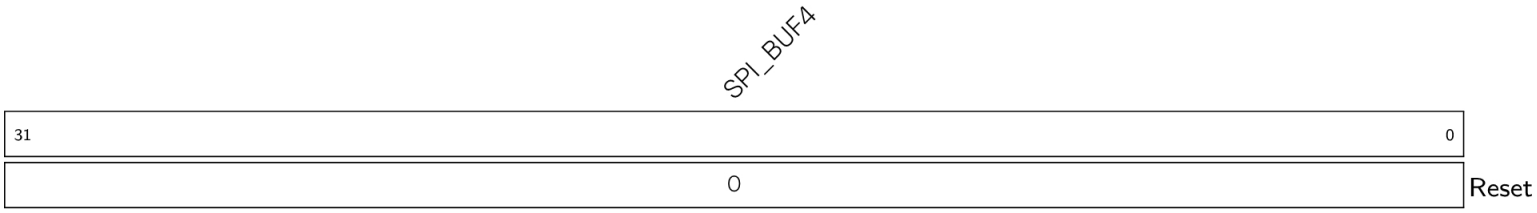
SPI_BUF2 32-bit data buffer 2. (R/W/SS)

Register 30.26. SPI_W3_REG (0x00A4)



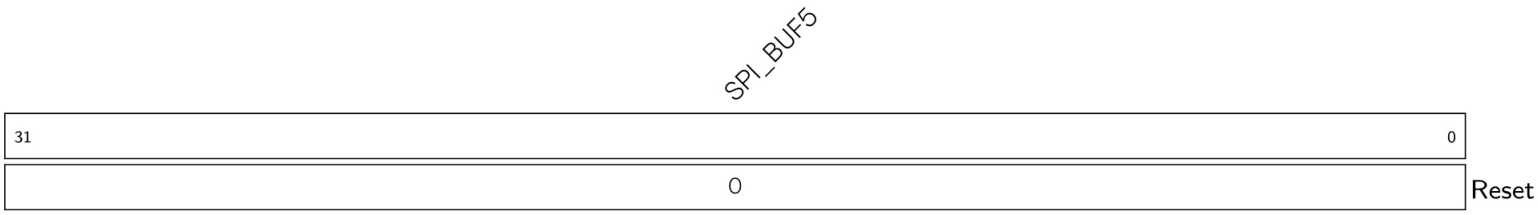
SPI_BUF3 32-bit data buffer 3. (R/W/SS)

Register 30.27. SPI_W4_REG (0x00A8)



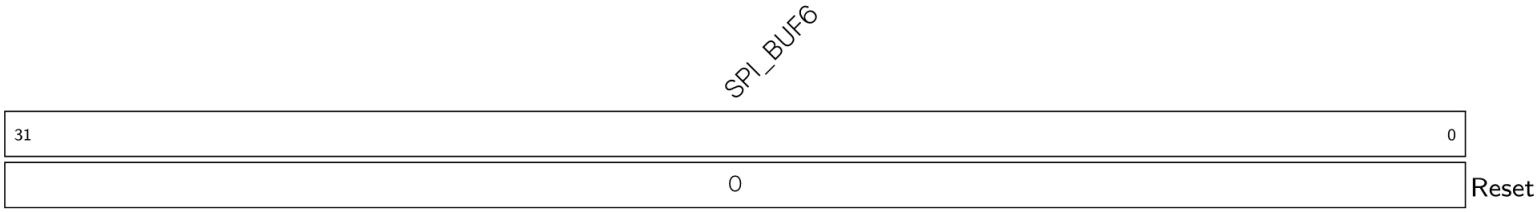
SPI_BUF4 32-bit data buffer 4. (R/W/SS)

Register 30.28. SPI_W5_REG (0x00AC)



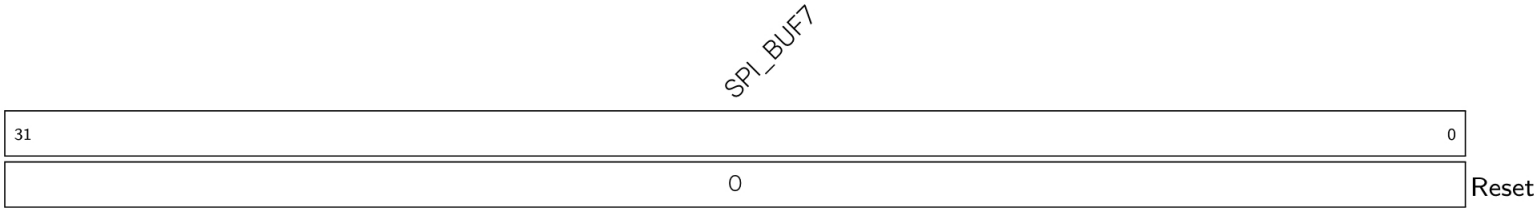
SPI_BUF5 32-bit data buffer 5. (R/W/SS)

Register 30.29. SPI_W6_REG (0x00B0)



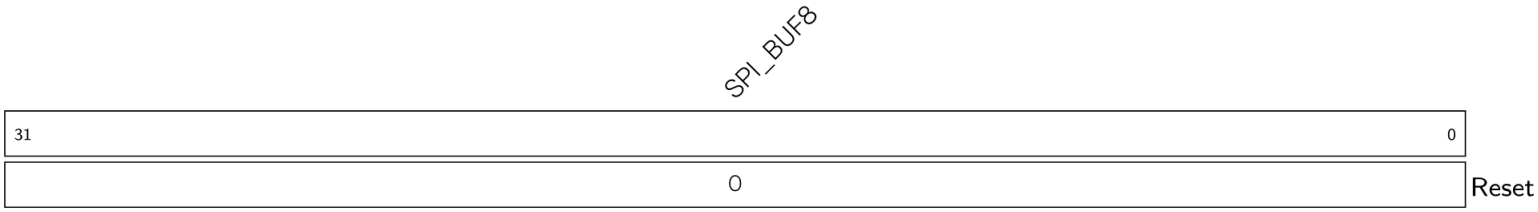
SPI_BUF6 32-bit data buffer 6. (R/W/SS)

Register 30.30. SPI_W7_REG (0x00B4)



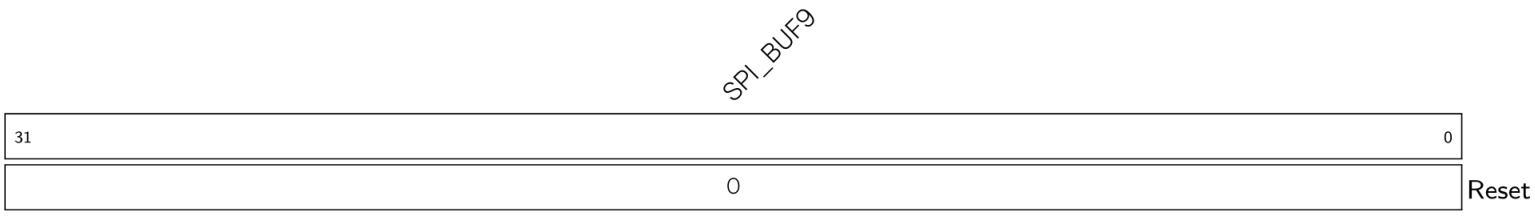
SPI_BUF7 32-bit data buffer 7. (R/W/SS)

Register 30.31. SPI_W8_REG (0x00B8)



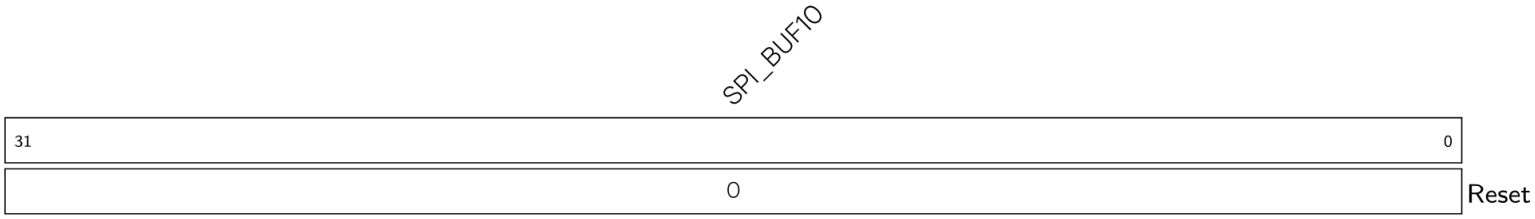
SPI_BUF8 32-bit data buffer 8. (R/W/SS)

Register 30.32. SPI_W9_REG (0x00BC)



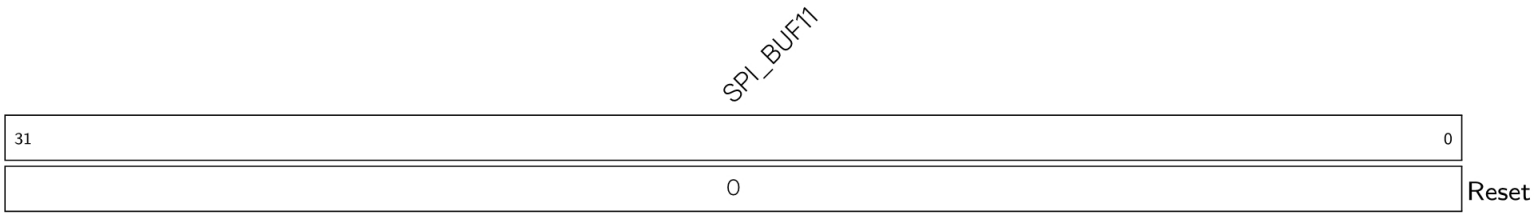
SPI_BUF9 32-bit data buffer 9. (R/W/SS)

Register 30.33. SPI_W10_REG (0x00C0)



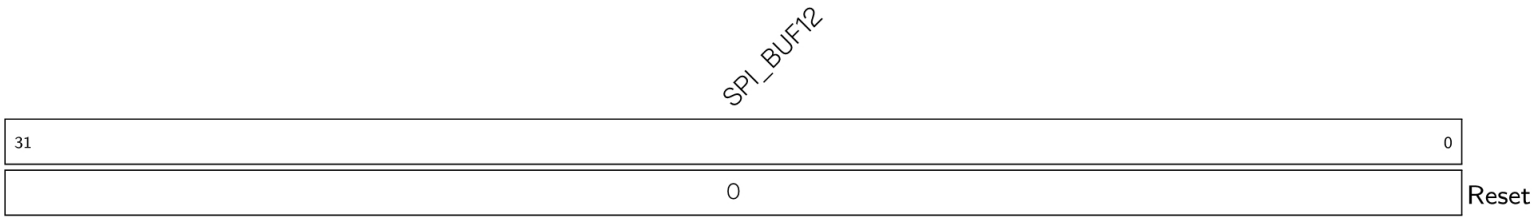
SPI_BUF10 32-bit data buffer 10. (R/W/SS)

Register 30.34. SPI_W11_REG (0x00C4)



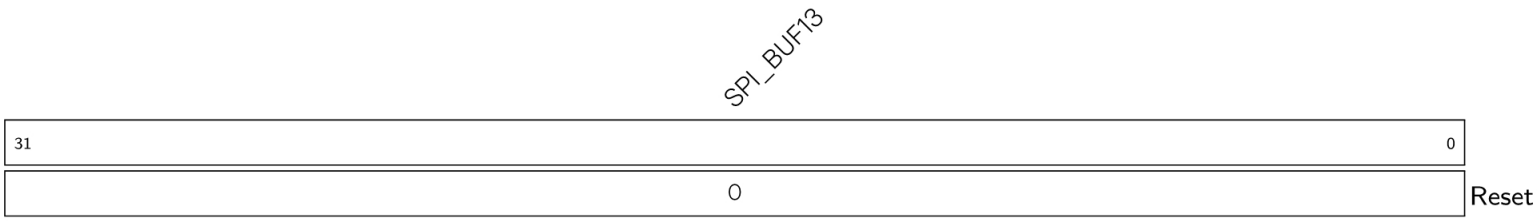
SPI_BUF11 32-bit data buffer 11. (R/W/SS)

Register 30.35. SPI_W12_REG (0x00C8)



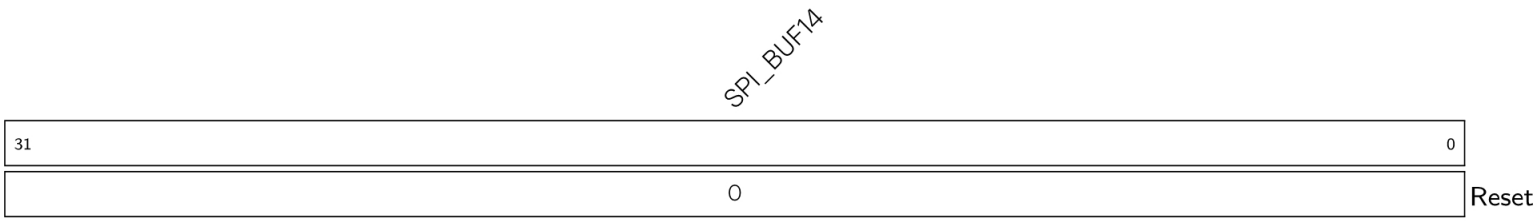
SPI_BUF12 32-bit data buffer 12. (R/W/SS)

Register 30.36. SPI_W13_REG (0x00CC)



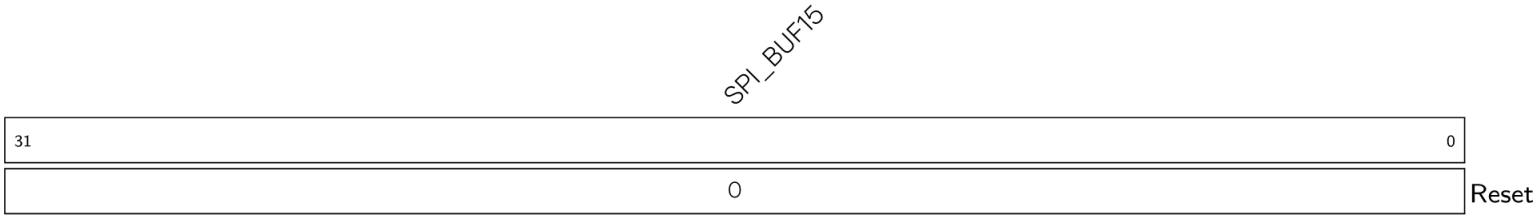
SPI_BUF13 32-bit data buffer 13. (R/W/SS)

Register 30.37. SPI_W14_REG (0x00D0)



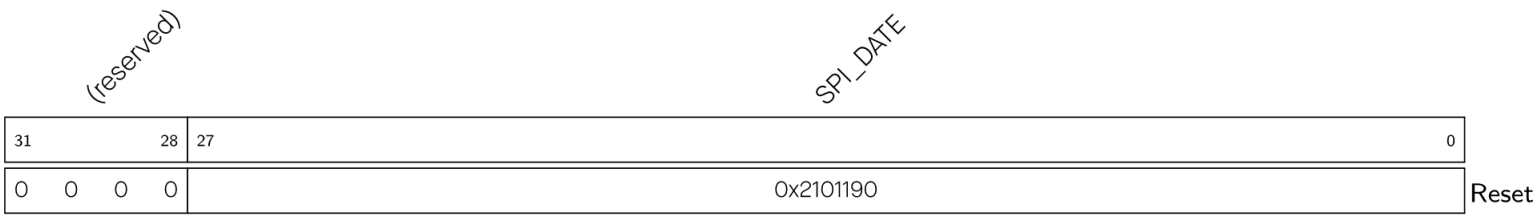
SPI_BUF14 32-bit data buffer 14. (R/W/SS)

Register 30.38. SPI_W15_REG (0x00D4)



SPI_BUF15 32-bit data buffer 15. (R/W/SS)

Register 30.39. SPI_DATE_REG (0x00F0)



SPI_DATE Version control register. (R/W)