68 Serial peripheral interface (SPI)

68.1 Introduction

The serial peripheral interface (SPI) can be used to communicate with external devices while using the specific synchronous protocol. The SPI protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as master or slave and is capable of operating in multi slave or multi master configurations. The device configured as master provides communication clock (SCK) to the slave device. The Slave select (SS) and ready (RDY) signals can be applied optionally just to setup communication with concrete slave and to assure it handles the data flow properly. The Motorola data format is used by default, but some other specific modes are supported as well.

68.2 SPI main features

- Full-duplex synchronous transfers on three lines
- Half-duplex synchronous transfer on two lines (with bidirectional data line)
- Simplex synchronous transfers on two lines (with unidirectional data line)
- From 4- up to 32-bit data size selection or fixed to 8-bit multiples
- Multi master or multi slave mode capability
- Dual clock domain, the peripheral kernel clock is independent from the APB bus clock
- Baud rate prescaler up to kernel frequency/2 or bypass from RCC in Master mode
- Protection of configuration and setting
- Hardware or software management of SS for both master and slave
- Adjustable minimum delays between data and between SS and data flow
- Configurable SS signal polarity and timing, MISO x MOSI swap capability
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Programmable number of data within a transaction to control SS and CRC
- Dedicated transmission and reception flags with interrupt capability
- SPI Motorola and TI formats support
- Hardware CRC feature can verify integrity of the communication at the end of transaction by:
  - Adding CRC value in Tx mode
  - Automatic CRC error checking for Rx mode
- Error detection with interrupt capability in case of data overrun, CRC error, data underrun, the mode fault and frame error, depending upon the operating mode
- Two multiples of 8-bit embedded Rx and Tx FIFOs (FIFO size depends on instance)
- Configurable FIFO thresholds (data packing)
- Capability to handle data streams by system DMA controller
- Configurable behavior for slave underrun condition (support of cascaded circular buffers)

- Autonomous functionality in Stop modes (handling of the transaction flow and required clock distribution) with wake-up from Stop capability
- Optional status pin RDY signalizing the slave device ready to handle the data flow

68.3 SPI implementation

Table 690 describes the SPI implementation. The instances are applied either with a full set or a limited set of features.

Table 690. SPI features

SPI feature	SPI1, SPI2 (full feature)	SPI3 (limited feature)
Data size	Configurable from 4 to 32 bits	8/16 bits
CRC computation	CRC polynomial length configurable from 5 to 33 bits	CRC polynomial length 9/17 bits
Size of FIFOs	16 x 8 bits	8 x 8 bits
Number of data control	Up to 65536	Up to 1024, no remaining data counter (CTSIZE)
Autonomous in Stop mode with wake-up capability	Yes	Yes
Autonomous in Standby mode with wake-up capability	No	No

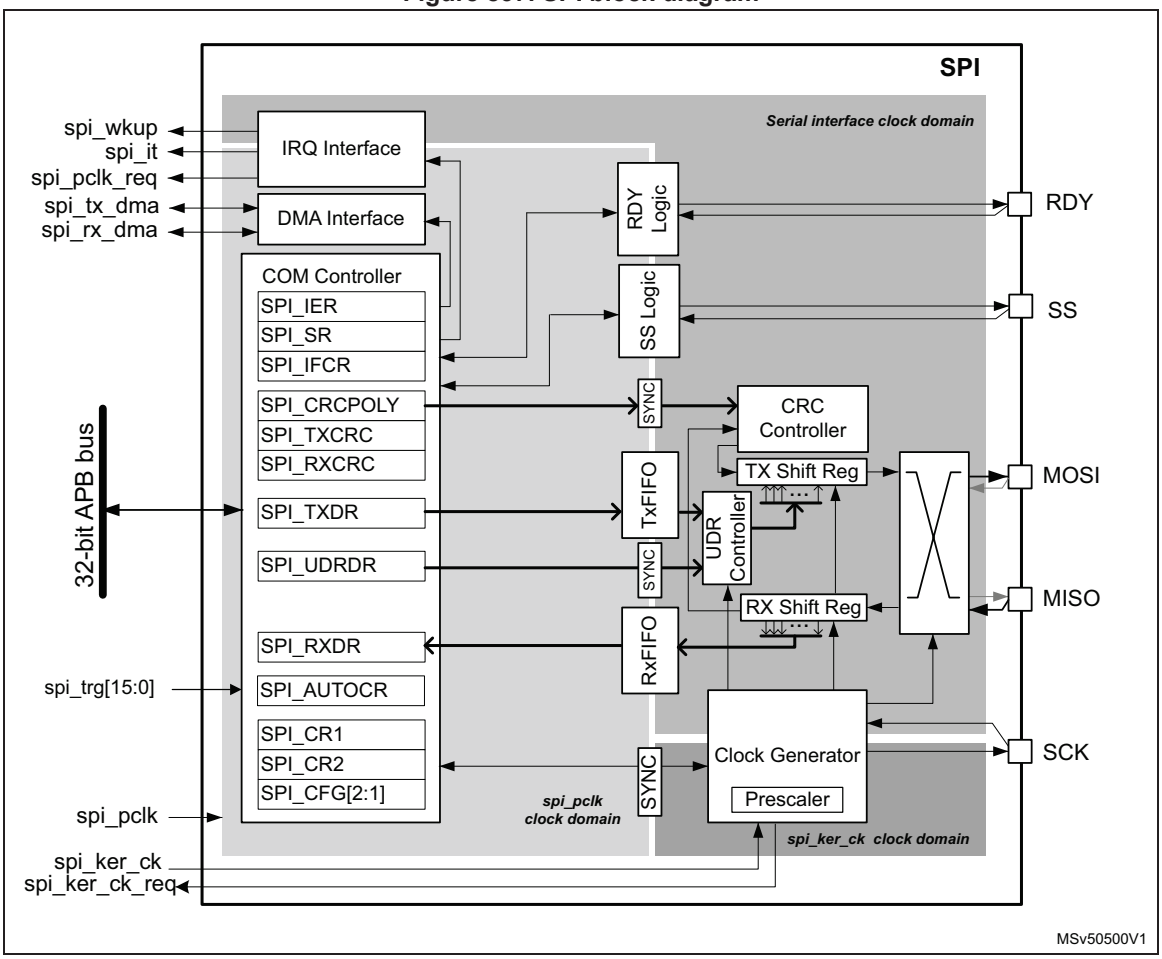
Note: For detailed information about instances capabilities to exit from Stop and Standby modes refer to Table 696: SPI wake-up and interrupt requests and Table 100: Functionalities depending on the working mode.

68.4 SPI functional description

68.4.1 SPI block diagram

The SPI enables a synchronous, serial communication between the MCU and external devices. The application software can manage the communication by polling the status flag or using a dedicated SPI interrupt. The main SPI elements and their interactions are shown in *Figure 837*.

Figure 837. SPI block diagram



The simplified scheme of *Figure 837* shows three fully independent clock domains:

- the **spi_pclk** clock domain
- the **spi_ker_ck** kernel clock domain
- the serial interface clock domain

All the control and status signals between these domains are strictly synchronized. There is no specific constraint concerning the frequency ratio between these clock signals. The user has to consider a ratio compatible with the data flow speed to avoid data underrun or overrun events.

The **spi_pclk** clock signal feeds the peripheral bus interface. It must be active when accesses to the SPI registers are required.

The SPI working in Slave mode handles a data flow using the serial interface clock derived from the external SCK signal provided by the external master SPI device. This is why the SPI slave is able to receive and send data even when the **spi_pclk** and **spi_ker_ck** clock signals are inactive. As a consequence, a specific slave logic working within the serial interface clock domain needs some additional traffic to be setup correctly (for example when underrun or overrun is evaluated, see *Section 68.5.2* for details). This cannot be done when the bus becomes idle. In some cases the slave even requires the clock generator working (see *Section 68.5.1*).

When the SPI works as master, it needs **spi_ker_ck** kernel clock coming from RCC active during communication to feed the serial interface clock via the clock generator where it can be divided by prescaler or bypassed optionally. The signal is then provided to slaves via the SCK pin and internally to the serial interface domain of the master.

68.4.2 SPI pins and internal signals

Up to five I/O pins are dedicated to SPI communication with external devices.

- **MISO:** master in / slave out data. In the general case, this pin is used to transmit data in Slave mode and receive data in Master mode.
- **MOSI:** master out / slave in data. In the general case, this pin is used to transmit data in Master mode and receive data in Slave mode.
- **SCK:** serial clock output pin for SPI masters and input pin for SPI slaves.
- **SS:** slave select pin. Depending on the SPI and SS settings, this pin can be used to either:
  - select an individual slave device for communication
  - synchronize the data frame, or
  - detect a conflict between multiple masters

  See *Section 68.4.7* for details.
- **RDY:** optional status pin signaling slave FIFO occupancies and so the slave availability to continue the transaction without any risk of data flow corruption. It can be checked by master to control temporal suspension of the ongoing communication.

The SPI bus enables the communication between one master device and one or more slave devices. The bus consists of at least two wires: one for the clock signal and the other for synchronous data transfer. Other signals are optional and can be added depending on the data exchange between SPI nodes and their communication control management.

Refer to *Table 691* and *Table 692* for the list of SPI input / output pins and internal signals.

Table 691. SPI/ input/output pins

Pin name	I/O type	Description
MISO(1)	Input/output	Master data input / slave data output
MOSI(1)	Input/output	Master data output / slave data input
SCK	Input/output	Master clock output / slave clock input
SS	Input/output	Master output / slave selection input
RDY	Input/output	SPI master input / slave FIFOs status occupancy output

1. Functionality of MOSI and MISO pins can be swapped. Their directions may vary in SPI bidirectional half duplex mode.

Description of SPI input/output signals

Table 692. SPI internal input/output signals

Signal name	Signal type	Description
spi_pclk	Input	SPI clock signal feeds the peripheral bus interface
spi_ker_ck	Input	SPI kernel clock
spi_ker_ck_req	Output	SPI kernel clock request
spi_pclk_req	Output	SPI clock request
spi_wkup	Output	SPI provides a wake-up interrupt
spi_it	Output	SPI global interrupt
spi_tx_dma	Input/output	SPI transmit DMA request
spi_rx_dma	Input/output	SPI receive DMA request
spi_trg[15:0]	Input	SPI triggers

Description of SPI interconnections

Table 693. SPI interconnection (SPI1 and SPI2)

Signal name	Trigger source
spi_trg0	gpdma1_ch0_tc
spi_trg1	gpdma1_ch1_tc
spi_trg2	gpdma1_ch2_tc
spi_trg3	gpdma1_ch3_tc
spi_trg4	exti4
spi_trg5	exti9
spi_trg6	lptim1_ch1
spi_trg7	lptim2_ch1
spi_trg8	comp1_out
spi_trg9	comp2_out(1)
spi_trg10	rtc_alra_trg
spi_trg11	rtc_wut_trg
spi_trg12	-
spi_trg13	-
spi_trg14	-
spi_trg15	-

1. This connection is not present in STM32U535/545 since COMP2 is not available.

Table 694. SPI interconnection (SPI3)

Signal name	Trigger source
spi_trg0	lpdma1_ch0_tc
spi_trg1	lpdma1_ch1_tc
spi_trg2	lpdma1_ch2_tc
spi_trg3	lpdma1_ch3_tc
spi_trg4	exti4
spi_trg5	exti8
spi_trg6	lptim1_ch1
spi_trg7	lptim3_ch1
spi_trg8	comp1_out
spi_trg9	comp2_out(1)
spi_trg10	rtc_alra_trg
spi_trg11	rtc_wut_trg
spi_trg12	-
spi_trg13	-
spi_trg14	-
spi_trg15	-

1. This connection is not present in STM32U535/545 since COMP2 is not available.

68.4.3 SPI communication general aspects

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software SS management) or 3 or 4 wires (with hardware SS management). The communication is always initiated and controlled by the master. The master provides a clock signal on the SCK line and selects or synchronizes slave(s) for communication by SS line when it is managed by hardware. The data between the master and the slave flow on the MOSI and/or MISO lines.

68.4.4 Communications between one master and one slave

The communication flow may use one of 3 possible modes: the full-duplex (3 wires) mode, half-duplex (2 wires) mode or the simplex (2 wires) mode. The SS signal is optional in single master-slave configuration and is often not connected between the two communication nodes. Nevertheless, the SS signal can be helpful in this configuration to synchronize the data flow and it is used by default for some specific SPI modes (for example the TI mode).

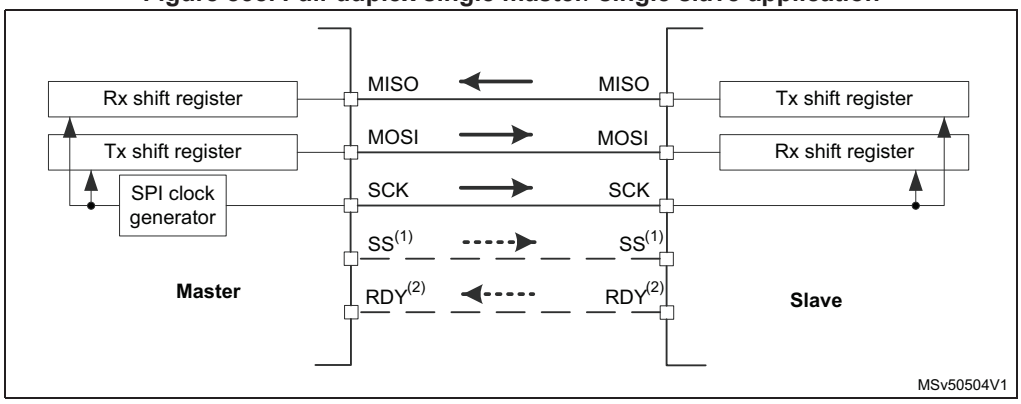
The next optional RDY signal can help to assure correct management of all the transacted data at slave side.

Full-duplex communication

By default, the SPI is configured for full-duplex communication (bits COMM[1:0] = 00 in the SPI_CFG2 register). In this configuration, the shift registers of the master and slave are

linked using two unidirectional lines between the MOSI and the MISO pins. During the SPI communication, the data are shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line simultaneously. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.

Figure 838. Full-duplex single master/ single slave application

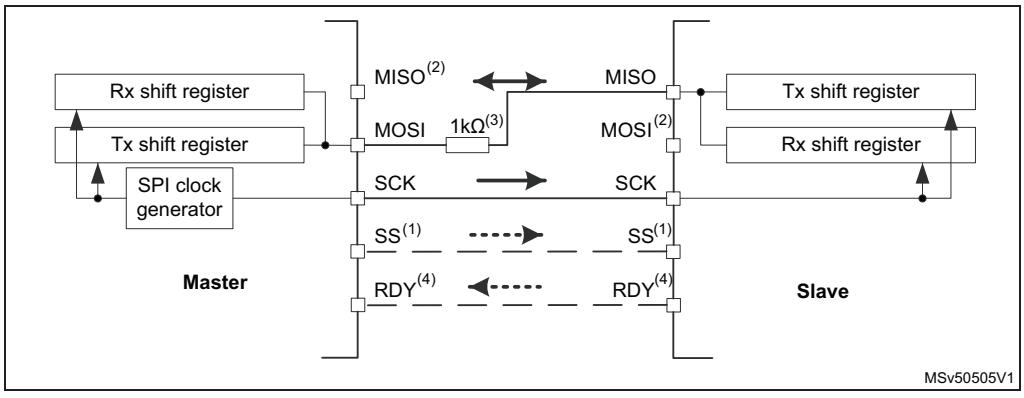


1. To apply SS pins interconnection is not mandatory to make the SPI interface working (see *Section 68.4.7* for details).
2. RDY signal provided by slave can be read by master optionally.

Half-duplex communication

The SPI can communicate in half-duplex mode by setting COMM[1:0] = 11 in the SPI_CFG2 register. In this configuration, one single cross connection line is used to link the shift registers of the master and slave together. During this communication, the data are synchronously shifted between the shift registers on the SCK clock edge in the transfer direction selected reciprocally by both master and slave with the HDDIR bit in their SPI_CR1 registers. Note that the SPI must be disabled when changing direction of the communication. In this configuration, the MISO pin at master and the MOSI pin at slave are free for other application uses and act as GPIOs.

Figure 839. Half-duplex single master/ single slave application



1. To apply SS pins interconnection is not mandatory to make the SPI interface working (see *Section 68.4.7*

for details).

2. In this configuration, the MISO pin at master and MOSI pin at slave can be used as GPIOs
3. A critical situation can happen when the communication direction is not changed synchronously between two nodes working in bidirectional mode. The new transmitter accesses the common data line while the former transmitter still keeps an opposite value on the line (the value depends on the SPI configuration and communicated data). The nodes can conflict temporary with opposite output levels on the line until the former transmitter changes its data direction setting. It is suggested to insert a serial resistance between MISO and MOSI pins in this mode to protect the conflicting outputs and limit the current flow between them.
4. RDY signal provided by slave can be read by master optionally.

Simplex communications

The SPI can communicate in simplex mode by setting the SPI in transmit-only or in receive-only using the COMM[1:0] field in the SPI_CFG2 register. In this configuration, only one line is used for the transfer between the shift registers of the master and slave. The remaining MISO or MOSI pins pair is not used for communication and can be used as standard GPIOs.

- **Transmit-only mode: COMM[1:0] = 01**

  The master in transmit-only mode generates the clock as long as there are data available in the TxFIFO and the master transfer is ongoing.

  The slave in transmit-only mode sends data as long as it receives a clock on the SCK pin and the SS pin (or software managed internal signal) is active (see *Section 68.4.7*).

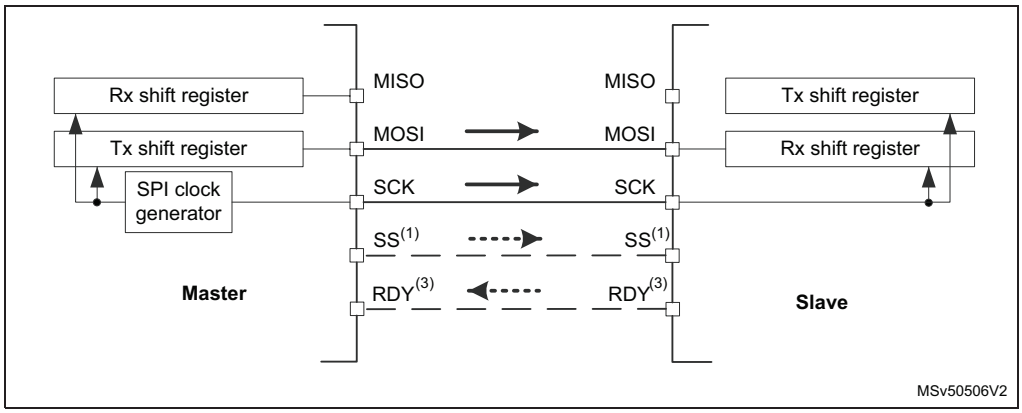
- **Receive-only mode: COMM[1:0] = 10**

  In Master mode, the MOSI output is disabled and may be used as GPIO. The clock signal is generated continuously as long as the SPI is enabled and the CSTART bit in the SPI_CR1 register is set. The clock is stopped either by software explicitly requesting this by setting the CSUSP bit in the SPI_CR1 register or automatically when the RxFIFO is full, when the MASRX bit in the SPI_CR1 is set.

  In slave configuration, the MISO output is disabled and the pin can be used as a GPIO. The slave continues to receive data from the MOSI pin while its slave select signal is active (see *Section 68.4.7*).

Note: In whatever Master and Slave modes, the data pin dedicated for transmission can be replaced by the data pin dedicated for reception and vice versa by changing the IOSWP bit value in the SPI_CFG2 register (This bit may only be modified when the SPI is disabled). Any simplex communication can be replaced by a variant of the half duplex communication with a constant setting of the transaction direction (bidirectional mode is enabled, while the HDDIR bit is never changed) or by full duplex control when unused data line and corresponding data flow is ignored.

Figure 840. Simplex single master / single slave application (master in transmit-only / slave in receive-only mode)

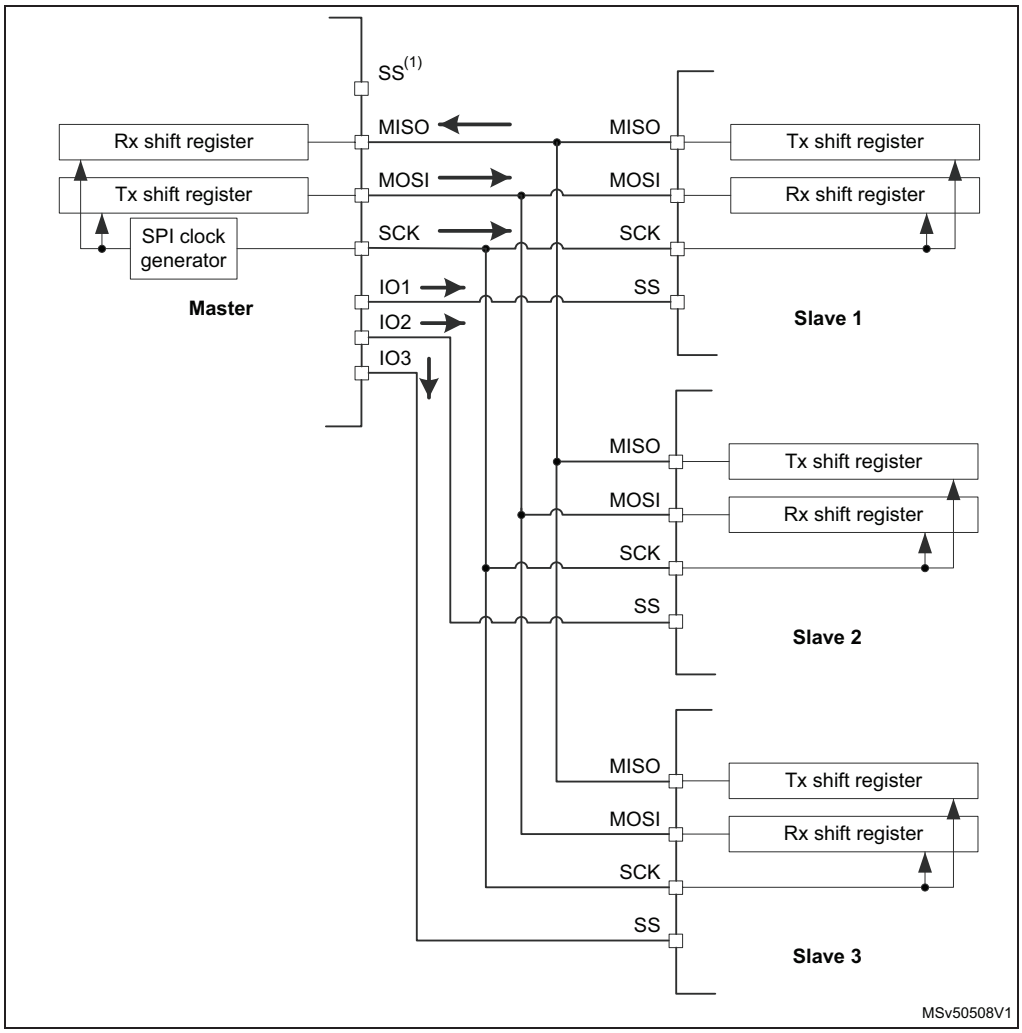


1. SS pins interconnection is not mandatory to make the SPI interface working (see *Section 68.4.7*).
2. In this configuration, both the MISO pins can be used as GPIOs.
3. RDY signal provided by slave can be read by master optionally.

68.4.5 Standard multislave communication

In a configuration with two or more independent slaves, the master uses a star topology with dedicated GPIO pins to manage the chip select lines for each slave separately (see *Figure 841.*). The master must select one of the slaves individually by pulling low the GPIO connected to the slave SS input (only one slave can control data on common MISO line at a given time). When this is done, a communication between the master and the selected slave is established. Except the simplicity, the advantage of this topology is that a specific SPI configuration can be applied for each slave as all the communication sessions are performed separately just within single master-slave pair. Optionally, when there is no need to read any information from slaves, the master can transmit the same information to the multiple slaves.

Figure 841. Master and three independent slaves connected in star topology



1. Master single SS pin hardware output functionality cannot support this topology (to be replaced by set of GPIOs under SW control) and user should avoid SPI AF setting at the pin (see *Section 68.4.7* for details).
2. If the application cannot assure that no more than a single SS active signal is provided by the master at a given time, it is better to configure MISO pins into open-drain configuration with an external pull up on the MISO line to prevent conflicts between interconnected outputs of the slaves. Else, a push-pull configuration can be applied without extra resistor (see I/O alternate function input/output (GPIO) section).
3. RDY signals can be read by master from the slaves optionally.

68.4.6 Multimaster communication

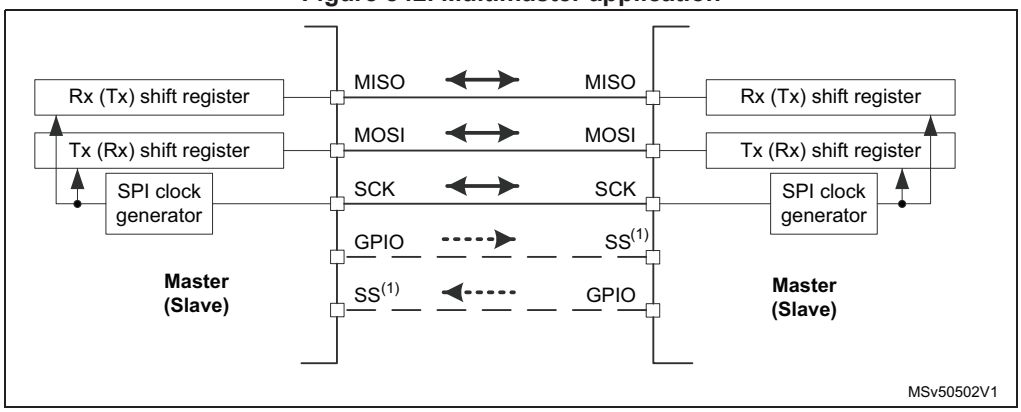
Unless the SPI bus is not designed primarily for a multimaster capability, it is possible to use a built-in feature that detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, the SS pin is used configured in hardware input mode. The connection of more than two SPI nodes working in this mode is impossible, as only one node can apply its output on a common data line at a given time.

When the nodes are not active, both stay in Slave mode by default. When a node wants to take control on the bus, it switches itself into Master mode and applies active level on the slave select input of the other node via the dedicated GPIO pin. After the session is

completed, the active slave select signal is released and the node mastering the bus temporary returns back to passive Slave mode waiting for a next session to start.

If both nodes raise their mastering request at the same time, a bus conflict event appears (see mode fault MODF event). The user can apply some simple arbitration process (for example postpone next attempt by predefined different time-outs applied in both nodes).

Figure 842. Multimaster application



1. The SS pin is configured at hardware input mode at both nodes. Its active level enables the MISO line output control as passive node is configured as a slave.
2. The RDY signal is not used in this communication.

68.4.7 Slave select (SS) pin management

In Slave mode, the SS works as a standard ‘chip select’ input and lets the slave communicate with the master. In Master mode, the SS can be used either as an output or an input. As an input it can prevent a multi master bus collision, and as an output it can drive a slave select signal of a single slave. The SS signal can be managed internally (software management of the SS input) or externally when both the SS input and output are associated with the SS pin (hardware SS management). The user can configure which level of this input/output external signal (present on the SS pin) is considered as active one by the SSIOP bit setting. SS level is considered as active if it is equal to SSIOP.

The hardware or software slave select management can be set using the SSM bit in the SPI_CFG2 register:

- **Software SS management (SSM = 1)**: in this configuration, slave select information is driven internally by the SSI bit value in the register SPI_CR1. The external SS pin is free for other application uses (as GPIO or other alternate function).
- **Hardware SS management (SSM = 0)**: in this case, there are two possible configurations. The configuration used depends on the SS output configuration (SSOE bit in register SPI_CFG2).
  - **SS output enable (SSOE = 1)**: this configuration is only used when the MCU is set as master. The SS pin is managed by the hardware. The functionality is tied to CSTART and EOT control. As a consequence, the master must apply proper TSIZE>0 setting to control the SS output correctly. Even if SPI AF is not applied at the SS pin (it can be used as a standard GPIO then), keep anyway SSOE = 1 to

assure default SS input level and prevent any mode fault evaluation at input of the master SS internal logic applicable at a multimaster topology exclusively.

a) When SSOM = 0 and SP = 000, the SS signal is driven to the active level as soon as the master transfer starts (CSTART = 1) and it is kept active until its EOT flag is set or the transmission is suspended.

b) When SP = 001, a pulse is generated as defined by the TI mode.

c) When SSOM = 1, SP = 000 and MIDI > 1 the SS is pulsed inactive between data frames, and kept inactive for a number of SPI clock periods defined by the MIDI value decremented by one (1 to 14).

d) SS input is forced to non active state internally at master to prevent its any mode fault.

– **SS output disable (SSM = 0, SSOE = 0)**:

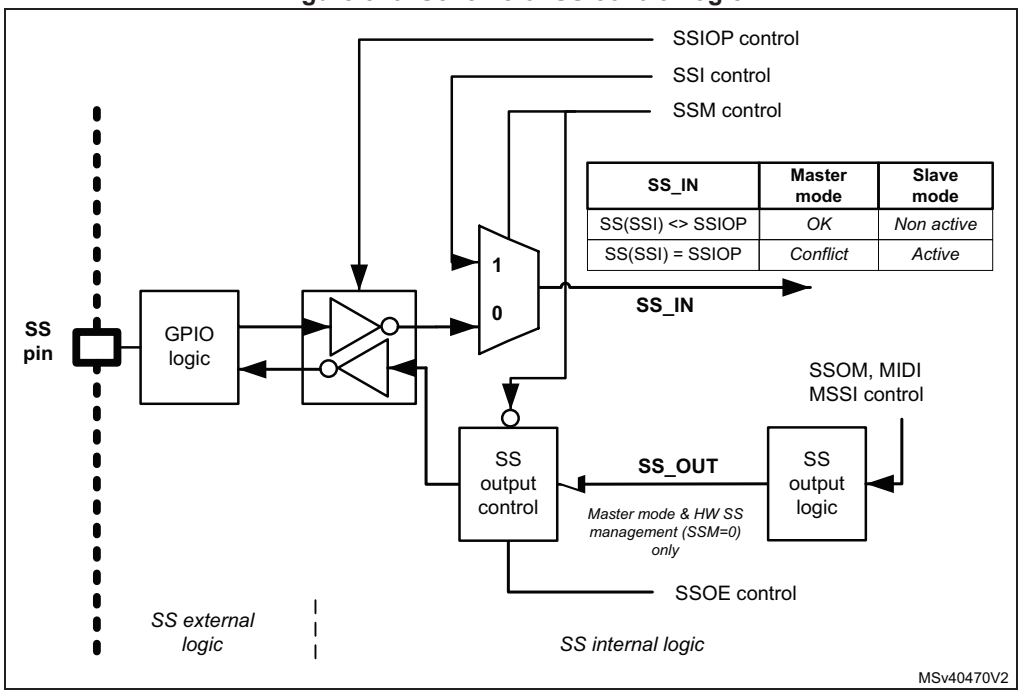
a) if the micro-controller is acting as the master on the bus, this configuration allows multi master capability. If the SS pin is pulled into an active level in this mode, the SPI enters Master mode fault state and the SPI device is automatically reconfigured in Slave mode (MASTER = 0).

b) In Slave mode, the SS pin works as a standard ‘chip select’ input and the slave is selected while the SS line is at its active level.

Note: The purpose of automatic switching into Slave mode at mode fault condition is to avoid the possible conflicts on data and clock line. As the SPE is automatically reset in this condition, both Rx and Tx FIFOs are flushed and current data is lost.

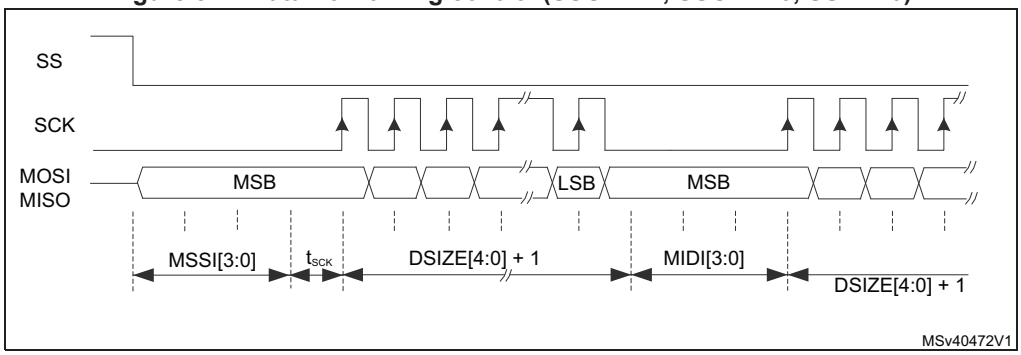
When the SPI slave is enabled in the hardware SS management mode, all the traffic is ignored even in case of the SS is found at active level. They are ignored until the slave detects a start of the SS signal (transition from non-active to active level) just synchronizing the slave with the master. This is because the hardware management mode cannot be used when the external SS pin is fixed. There is no such protection in the SS software management. Then the SSI bit must be changed when there is no traffic on the bus and the SCK signal is at idle state level between transfers exclusively in this case.

Figure 843. Scheme of SS control logic



When the hardware output SS control is applied (SSM = 0, SSOE = 1), by configuration of the MIDI[3:0] and MSSI[3:0] bitfields, the user can control the timing of the SS signal between data frames and can insert an extra delay at the beginning of every transaction (to separate the SS and clock starts). This can be useful when the slave needs to slow down the flow to obtain sufficient room for correct data handling (see *Figure 844*).

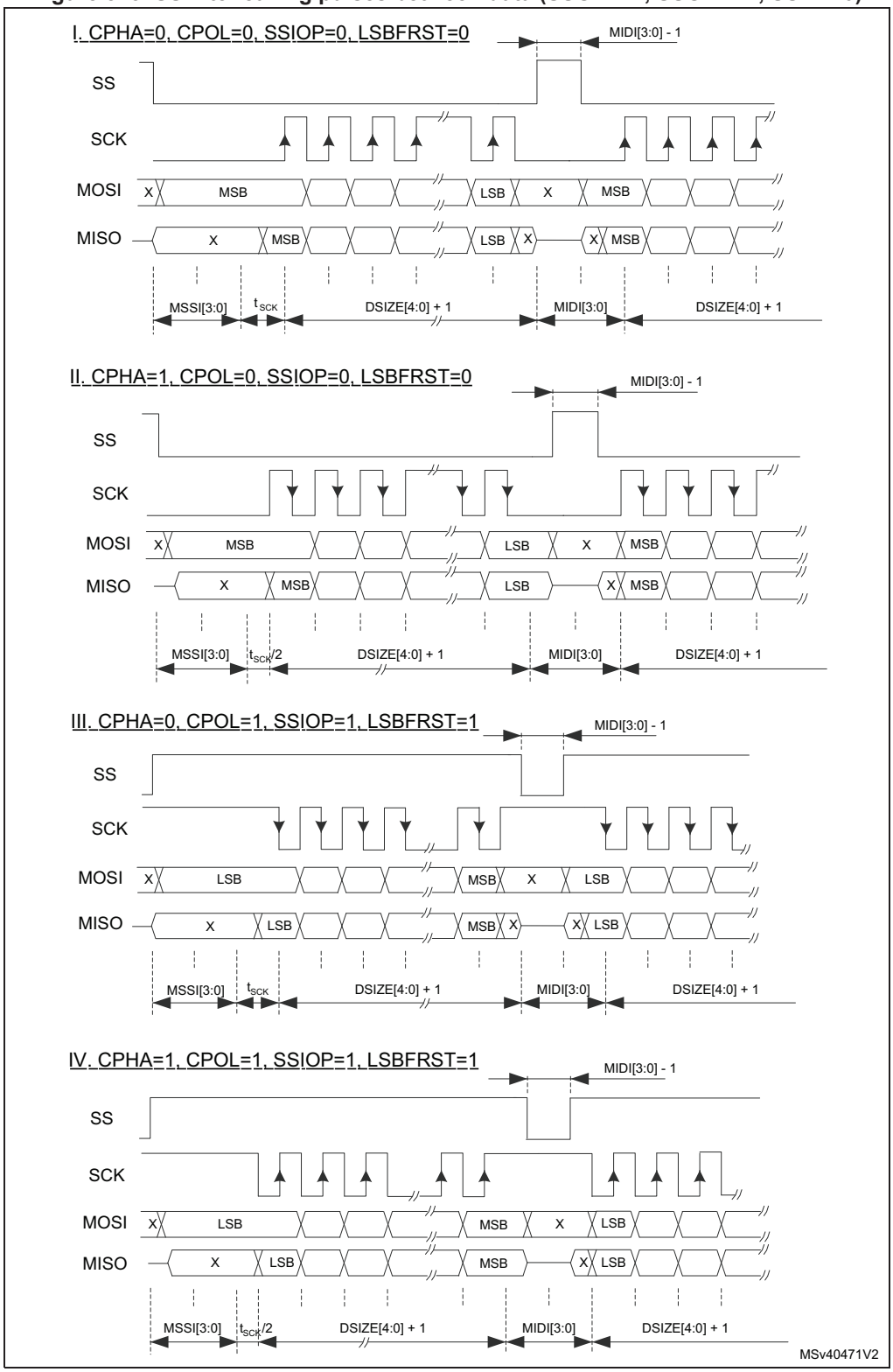
Figure 844. Data flow timing control (SSOE = 1, SSOM = 0, SSM = 0)



1. MSSI[3:0] = 0011, MIDI[3:0] = 0011 (SCK flow is continuous when MIDI[3:0] = 0).
2. CPHA = 0, CPOL = 0, SSIOP = 0, LSBFRST = 0.

Additionally, bit SSOM = 1 setting invokes specific mode which interleaves pulses between data frames if there is a sufficient space to provide them (MIDI[3:0] must be set greater then one SPI period). Some configuration examples are shown in *Figure 845*.

Figure 845. SS interleaving pulses between data (SSOE = 1, SSOM = 1, SSM = 0)



1. MSSI[3:0] = 0010, MIDI[3:0] = 0010.
2. SS interleaves between data when MIDI[3:0] > 1 wide of the interleaving pulse is always one SCK period less then gap provided between frames (defined by MIDI parameter). If MIDI is set the frames are separated by single SCK period but no interleaving pulse appears on SS.

68.4.8 Ready pin (RDY) management

The status of the slave capability to handle data, can be checked on the RDY pin. By default, a low level indicates that the slave is not ready for transaction. The reason can be that slave's TxFIFO is empty, RxFIFO full or the SPI is disabled. An active level of the signal can be selected by the RDIOP bit. If the master continues or starts to communicate with the slave when it indicates a not ready status, the transaction fails great probably.

The logic to control the RDY output is rather complex, tied closely with TSIZE and DSIZE settings. The RDY reaction is more pessimistic and sensitive to TxFIFO becoming nearly empty and/or RxFIFO nearly full during a frame transaction. This pessimistic logic is suppressed at the end of a transaction only when RDY stays active, despite TxFIFO becomes fully empty and/or RxFIFO becomes fully occupied. The target is to prevent any data corruption and inform master in time that it is necessary to suspend the transaction temporarily till the next transacted data can be processed safely again. When RDY signal input is enabled at master side, master suspends the communication once the slave indicates not ready status. This prevents the master to complete transaction of an ongoing frame which just empties slave's TxFIFO or full fills its RxFIFO till a next data is written and/or read there (despite the frame still can be completed without any constraint). It can make a problem if TSIZE = 0 configuration is applied at slave because slave then never evaluates end of transaction (which suppresses the not ready status just when the last data is sent). Then the user has to release the RxFIFO and/or write additional (even dummy) data to TxFIFO by software at slave side to release the not RDY signal, unblock ST master and so enable it to continue at the communication suspended at middle of a frame occasionally.

When RDY is not used by the master, it must be disabled (RDIOM = 0). Then an internal logic of the master simulates the slave status always ready. In this case, the RDIOP bit setting has no meaning.

Due to synchronization between clock domains and evaluation of the RDY logic on both master and slave sides, the RDY pin feature is not reliable and cannot be used when the size of data frames is configured shorter than 8-bit.

68.4.9 Communication formats

During SPI communication, receive and transmit operations are performed simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of the information on the data lines. The communication format depends on the clock phase, the clock polarity and the data frame format. To be able to communicate together, the master and slave devices must follow the same communication format and be synchronized correctly.

Clock phase and polarity controls

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPI_CFG2 register. The CPOL (clock polarity) bit controls the idle state value of the clock when no data are being transferred. This bit affects both Master and Slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

If the CPHA bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is set, rising edge if the CPOL bit is reset). Data are latched on each occurrence of this clock transition type.

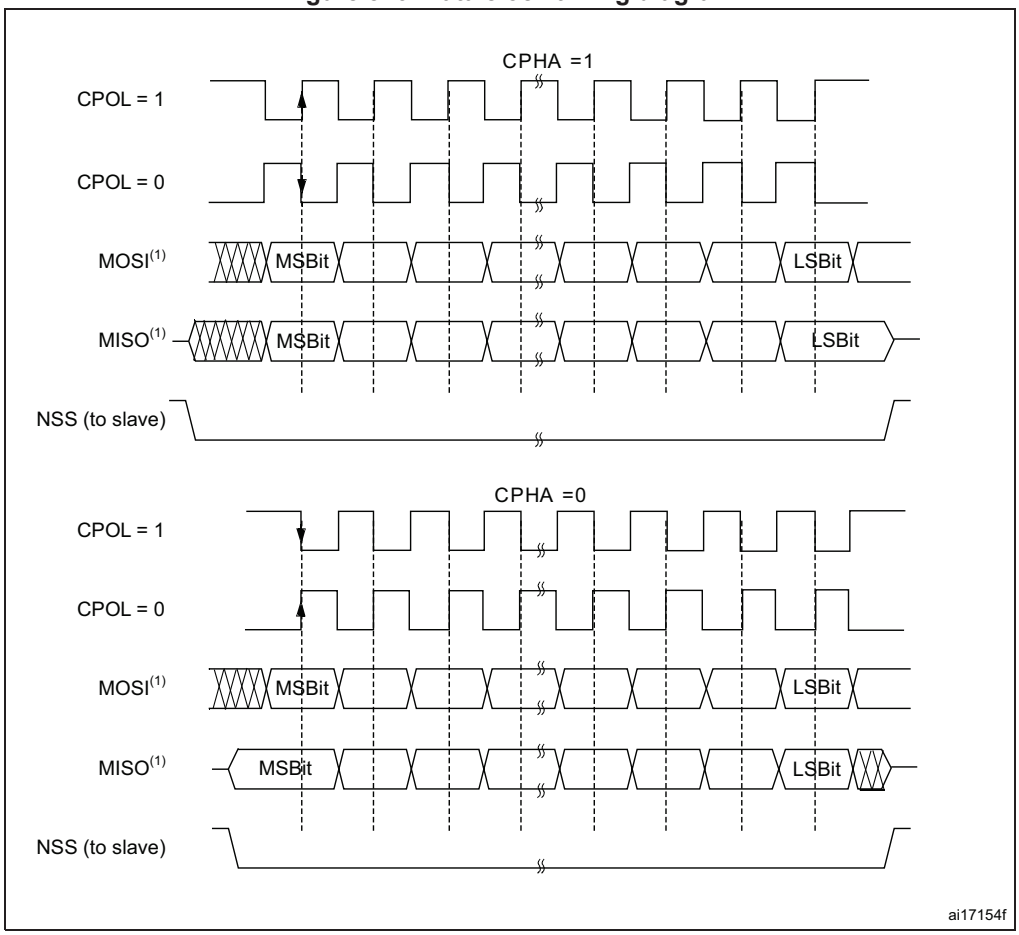
The combination of the CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edges (dotted lines in *Figure 846*).

Figure 846, shows an SPI full-duplex transfer with the four combinations of the CPHA and CPOL bits.

Note: Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.

The idle state of SCK must correspond to the polarity selected in the SPI_CFG2 register (by pulling the SCK pin up if CPOL = 1 or pulling it down if CPOL = 0).

Figure 846. Data clock timing diagram



1. The order of data bits depends on LSBFRST bit setting.

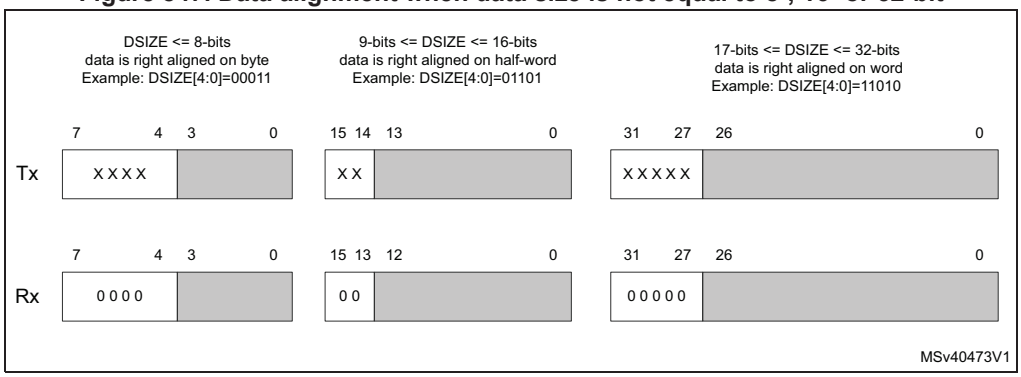
Data frame format

The SPI shift register can be set up to shift out MSB-first or LSB-first, depending on the value of the LSBFRST bit in SPI_CFG2 register.

At instance with full feature set, the data frame size is chosen by using the DSIZE[4:0] bits at SPI_CFG1 register. It can be set from 4-bit up to 32-bit length and the setting applies for both transmission and reception. When the SPI_TXDR/SPI_RXDR registers are accessed, data frames are always right-aligned into either a byte (if the data fit into a byte), a half-word or a word (see *Figure 847*).

If the access is a multiple of the configured data size, data packing is applied automatically. During communication, only bits within the data frame are clocked and transferred.

Figure 847. Data alignment when data size is not equal to 8-, 16- or 32-bit



Note: The minimum data length is 4 bits. If a data length of less than 4 bits is selected, it is forced to an 4-bit data frame size.

At the instance with limited set of features, data size is fixed to multiply 8-bit up to maximum data length (depends on instance) in according to DSIZE[4:3] bits value. If the SPI_TXDR or SPI_RXDR are accessed by wider access (a multiply of the configured the data size), data packing is applied automatically.

68.4.10 Configuring the SPI

The configuration procedure is almost the same for the master and the slave. For specific mode setups, follow the dedicated chapters. When a standard communication must be initialized, perform these steps:

1. Write the proper GPIO registers: configure GPIO alternate functions at MOSI, MISO, SCK, SS and RDY pins if applied.
2. Write into the SPI_CFG1 and SPI_CFG2 registers and set up proper values of all ‘not reserved’ bits and bitfields, prior SPI is enabled, with the following exceptions:
   a) The SSOM, MASRX, SSOE, RDIOM, MBR[2:0], BPASS, MIDI[3:0], MSSI[3:0] bits are taken into account in Master mode only, the MSSI[3:0] bits take effect when the SSOE bit is set, the RDIOP bit takes no effect when the RDIOM bit is not set in Master mode. When slave is configured at TI mode, MBR[2:0] setting is considered, too.
   b) UDRCFG is taken into account in Slave mode only.
   c) CRCSIZE[4:0] is required if CRCEN is set.

d) CPOL, CPHA, LSBFRST, SSOM, SSOE, SSIOP, SSM, RDIOP, RDIOM, MSSI and MIDI are not required in TI mode.

e) Once the AFCNTR bit is set in the SPI_CFG2 register, all the SPI outputs start to be propagated onto the associated GPIO pins regardless the peripheral enable. So any later configuration changes of the SPI_CFG1 and SPI_CFG2 registers can affect the level of signals on these pins.

3. Write to the SPI_CR2 register to select the length of the transfer, if it is not known TSIZE must be programmed to zero.
4. Write to SPI_CRCPOLY and into TCRCINI, RCRCINI and CRC33_17 bits at the SPI_CR1 register to configure the CRC polynomial and CRC calculation if needed.
5. Configure DMA streams dedicated for the SPI Tx and Rx in DMA registers if the DMA streams are used (see chapter Communication using DMA).
6. Configure SSI, HDDIR and MASRX at SPI_CR1 register if required.
7. Program the IOLOCK bit in the SPI_CFG1 register if the configuration protection is required (for safety).

68.4.11 Enabling the SPI

It is recommended to configure and enable the SPI slave before the master sends the clock. But there is no impact if the configuration and enabling procedure is done while a traffic is ongoing on the bus, assuming that the SS signal is managed by hardware at slave or kept inactive by slave's software when the software management of the SS signal is applied (see *Section 68.4.7*). The data register of the slave transmitter should contain data to be sent before the master starts its clocking.The SCK signal must be settled to the idle state level corresponding to the selected polarity, before the SPI slave is selected by SS, else the following transaction may be desynchronized.

When the SPI slave is enabled at the hardware SS management mode, all the traffics are ignored even in case of the SS is found at active level. They are ignored until the slave detects a start of the SS signal (its transition from non-active to active level) just synchronizing the slave with the master. This is why the hardware management mode cannot be used when external SS pin is fixed. There is no such protection at the SS software management. In this case, the SSI bit must be changed when there is no traffic on the bus and the SCK signal is at idle state level between transfers exclusively in this case.

The master in full duplex (or in any transmit-only mode) starts to communicate when the SPI is enabled, the CSTART bit is set and the TxFIFO is not empty, or with the next write to TxFIFO.

In any master receive-only mode, the master starts to communicate and the clock starts running after the SPI is enabled and the CSTART bit is set.

For handling DMA, see *Section 68.4.15*.

68.4.12 SPI data transmission and reception procedures

The setting of data communication format follows the basic principle that sure number of data with a flexible size must be transferred within a session (transaction) while, optionally, the data handling can be cumulated effectively into a single access of the SPI data registers (data packing) or even grouped into a sequence of such services if data is collected at consistent bigger data packets. The data handling services are based upon FIFO packet occupancy events. This is why the complete data packet must be serviced exclusively upon a dedicated packet flag.

To understand better the next detailed content of this section, the user should capture the configuration impact and meaning of the following items at first:

Data size (DSIZE) - defines data frame (sets the number of bits at single data frame).

FIFO threshold (FTHLV) - defines data packet, sets the number of data frames at single data packet and so the occurrence of the packet occupancy events to handle SPI data registers either by software or by DMA.

Data access – a way how to handle the SPI data register content when the transfer data between the application and the SPI FIFOs upon a packet event. It depends on the packet size configuration. Optionally, multiply data can be handled effectively by a single access of the register (by data packing) or by sequence of such accesses (when servicing a bigger data packet).

FIFO size – capacity or space to absorb available data. It depends on the data size and the internal hardware efficiency how the data is compressed and organized within this space. The FTHLV setting must respect the FIFO capacity to store two data packets at least.

Transaction size (TSIZE) – defines total number of data frames involved at a transaction session overall possibly covered by several data packet services. There is no need to align this number with the packet size (handling of a last not aligned data packet is supported if TSIZE is programmed properly).

Data handling via RxFIFO and TxFIFO

All SPI data transactions pass through the embedded FIFOs organized by bytes (N x 8-bit). The size of the FIFOs (N) is dependent on the product and the peripheral instance. This enables the SPI to work in a continuous flow, and prevents overruns when the data frame size is short or the interrupt/DMA latency is too long. Each direction has its own FIFO called TxFIFO and RxFIFO, respectively.

The handling of the FIFOs content is based on servicing data packet events exclusively raised by dedicated FIFO packet occupancy flags (TXP, RXP or DXP). The flags occurrence depends on the data exchange mode (duplex, simplex), the data frame size (number of bits in the frame) and how data are organized at data packets. The frequency of the packet events can be decreased significantly when data are organized into packets via defining the FIFOs threshold. Several data frames grouped at packet can be then handled effectively based on a single FIFO occupancy packet event either by a single SPI data register access or their sequence what consumes less system performance. The user can control the access type by casting the data register address to force a concrete CPU instruction applied for the register read or write. The access then can be 8-bit, 16-bit or 32-bit but single data frame must be always accessed at least. It is crucial to keep the setting of the packet size (FTHVL) and the data size (DSIZE) always balanced with the applied data registers access (no matter if a single access or their sequence is applied) just to apply and complete service of a single data packet upon its event. This principle, occurrence and clearing capabilities of the FIFO occupancy flags are common no matter if DMA, interrupt, or polling is applied.

A read access to the SPI_RXDR register returns the oldest value stored in the RxFIFO that has not been read yet. A write access to the SPI_TXDR stores the written data in the TxFIFO at the end of a send queue.

A read access to the SPI_RXDR register must be managed by the RXP event. This flag is set by hardware when at least one complete data packet (defined as receiver threshold by FTHLV[3:0] bits at the SPI_CFG1 register) is available at the reception FIFO while reception is active. The RXP is cleared as soon as less data than complete single packet is available in the RxFIFO, when reading SPI_RXDR by software or by DMA.

The RXP triggers an interrupt if the RXPIE bit is set and/or a DMA request if the RXDMAEN bit is set.

Upon setting of the RXP flag, the application performs the due number of SPI data register reads to download the content of one data packet. Once a complete data packet is downloaded, the application software or DMA checks the RXP value to see if other packets are pending into the receive FIFO and, if so, downloads them packet by packet until the RXP reads 0. RxFIFO can store up to N data frames (for frame size ≤ 8-bit), N/2 data frames (for 8-bit < frame ≤ 16-bit), N/3 data frames (for 16-bit < frame ≤ 24-bit) or N/4 data frames (if data frame > 24-bit) where N is the size of the FIFO in bytes.

At the end of a reception, it may happen that some data are still available in the RxFIFO, without reaching the FTHLV level, thus the RXP is not set. In this case, the number of remaining RX data frames in the FIFO is indicated by RXWNE and RXPLVL fields in the SPI_SR register. It happens when the number of the last data received in a transfer cannot fully accomplish the configured packet size; in case the transfer size and the packet size are not aligned. Nevertheless the application software can still perform the standard number of reads from the RxFIFO used for the previous complete data packets without drawbacks: only the consistent data (completed data frames) are popped from the RxFIFO while redundant reads (or any uncompleted data) are reading 0. Thanks to that, the application software can treat all the data in a transfer in the same way, and is off-loaded to foresee the reception of the last data in a transfer and to calculate the due number of reads to be popped from RxFIFO.

In a similar way, the write access of a data frame to be transmitted is managed by the TXP event. This flag is set by hardware when there is enough space for the application to push at least one complete data packet (defined at FTHLV[3:0] bits at SPI_CFG1 register) into the transmission FIFO while transmission is active. The TXP is cleared as soon as the TxFIFO is filled by software and/or by the DMA. The space currently available for any next complete data packet is lost. This can lead to oscillations of the TXP signal when data are released out from the TxFIFO while a new packet is stored frame by frame. Any write to the TxFIFO is ignored when there is no sufficient room to store at least a single data frame (TXP event is not respected), when TXTF is set or when the SPI is disabled.

The TXP triggers an interrupt if the TXPIE bit is set and/or with a DMA request if the TXDMAEN bit is set. The TXPIE mask is cleared by hardware when the TXTF flag is set.

Upon setting of the TXP flag, the application performs the due number of SPI data register writes to upload the content of one entire data packet. Once new complete data packet is uploaded, the application software or DMA checks the TXP value to see if other packets can be pushed into the TxFIFO and, if so, uploads them packet by packet until TXP reads 0.

The number of last data in a transfer can be shorter than the configured packet size in the case when the transfer size and the packet size are not aligned. Nevertheless the application software can still perform the standard number of data register writes used for the previous packets without drawbacks: only the consistent data are pushed into the TxFIFO while redundant writes are discarded. Thanks to that, the application software can treat all the data in a transfer in the same way and is off-loaded to foresee the transmission of the last data in a transfer and from calculating the due number of writes to push the last data into TxFIFO. Just for the last data case, the TXP event is asserted by SPI once there is enough space into TxFIFO to store remaining data to complete current transfer.

Both TXP and RXP events can be polled or handled by interrupts. The DXP bit can be monitored as a common TXP and RXP event at full-duplex mode.

Upon setting of the DXP flag the application performs the due number of writes to the SPI data register to upload the content of one entire data packet for transmission, followed by the same number of reads from the SPI data register to download the content of one data packet. Once one data packet is uploaded and one is downloaded, the application software or DMA checks the DXP value to see if other packets can be pushed and popped in sequence and, if so, uploads/downloads them packet by packet until DXP reads 0.

The DXP triggers an interrupt if the DXPIE bit is set. The DXPIE mask is cleared by hardware when the TXTF flag is set.

The DXP is useful in full-duplex communication in order to optimize performance in data uploading/downloading, and reducing the number of interrupts or DMA sequences required to support an SPI transfer thus minimizing the request for CPU bandwidth and system power especially when SPI is operated in Stop mode.

When relay on the DXP interrupt exclusively, the user must consider the drawback of such a simplification when TXP and RXP events are serviced by common procedures because the TXP services are delayed by purpose in this case. This is due to fact that the TXP events precedes the reception RXP ones normally to allow the TXP servicing prior transaction of the last frame fully emptying the TxFIFO else master cannot provide a continuous SCK clock flow and the slave can even face an underrun condition. The possible solution is to prefill the TxFIFO by few data packets ahead prior the session starts and to handle all the data received after the TXTF event by EOT exclusively at the end of the transaction (as TXTF suppresses the DXP interrupts at the end of the transaction). In case of CRC computation is enabled, the user must calculate with additional space to accommodate the CRC frame at RxFIFO when relying on EOT exclusively at the end of transaction.

Another way to manage the data exchange is to use DMA (see *Section 68.4.15*).

If the next data is received when the RxFIFO is full, an overrun event occurs (see description of OVR flag in *Section 68.5.2*). An overrun event can be polled or handled by an interrupt.

This may happen in Slave mode or in a master receive mode when MASRX = 0. If MASRX bit is set at a master receiver, the generated clock stops automatically when the RxFIFO is full, therefore overrun is prevented.

Both RxFIFO and TxFIFO content is kept flushed and cannot be accessed when SPI is disabled (SPE = 0).

Transaction handling

A few data frames can be passed at single sequence to complete a message. The user can handle a number of data within a message thanks to the value stored into TSIZE. In principle, the transaction of a message starts when the SPI is enabled by setting CSTART bit and finishes when the total number of required data is transacted. The end of transaction controls the CRC and the hardware SS management when applied. To restart the internal state machine properly, SPI is strongly suggested to be disabled and re-enabled before next transaction starts despite its setting is not changed.

If TSIZE is kept at zero while CSTART is set, an endless transaction is initialized (no control of transfer size is applied). During an endless transaction, the number of transacted data aligned with FIFOs threshold is supported exclusively. If the number of data (or its grouping into packets) is unpredictable, the user must keep the FIFO threshold setting (packet size) at single data (FTHLV = 0) to assure that each data frame raises its own packet event to be serviced by the application or DMA. The transaction can be suspended at any time thanks

to CSUSP which clears the CSTART bit. SPI must be always disabled after such software suspension and re-enabled before the next transaction starts.

When the transmission is enabled, a sequence begins and continues while any data is present in the TxFIFO of the master. The clock signal is provided permanently by the master until TxFIFO becomes empty, then it stops, waiting for additional data.

In receive-only modes, half-duplex (COMM[1:0] = 11, HDDIR = 0) or simplex (COMM[1:0] = 10) modes, the master starts the sequence when the SPI is enabled and the transaction is released by setting the CSTART bit. The clock signal is provided by the master and it does not stop until either SPI or receive-only mode is disabled/suspended by the master. The master receives data frames permanently up to this moment. The reception can be suspended either by software control, writing 1 to the CSUSP bit in the SPI_CR1 register, or automatically when MASRX = 1 and RxFIFO becomes full or upon the RDY status if this signal is applied (see *Section 68.4.8*). The reception is automatically stopped also when the number of frames programmed in TSIZE has been completed.

In order to disable the master receive-only mode, the SPI must first be suspended. When the SPI is suspended, the current frame is completed, before changing the configuration.

Caution: If the SPE bit is cleared in Master mode, while the reception is ongoing without any suspending, the clock is stopped without completing the current frame, and the RxFIFO is flushed.

While the master can provide all the transactions in continuous mode (SCK signal is continuous) it must respect the slave capability to handle data flow and its content at anytime. If the slave features the RDY signal option, the master can monitor the RDY signal issued by the slave, to control the communication flow. If the RDY pin is not used, the slave is considered always ready for communication with the master.

When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays by MIDI[3:0] bits setting or provide an initial delay by setting MSSI[1:0], which postpones any transaction start to give slave sufficient room for preparing data. Be aware data from the slave are always transacted and processed by the master even if the slave cannot prepare it correctly in time. It is preferable for the slave to use DMA, especially when data frames are short, FIFO is accessed by bytes and the SPI bus rate is high.

In order to add some software control on the SPI communication flow from a slave transmitter node, a specific value written in the SPI_UDRDR (SPI Underrun Data Register) may be used. On slave side, when TxFIFO becomes empty, this value is sent out automatically as next data and may be interpreted by software on the master receiver side (either simply dropped or interpreted as a XOFF like command, in order to suspend the master receiver by software).

In the multislave star topology, only a single slave only can be enabled for output data at a given time. The slave just selected for the communication with the master needs to detect a change of its SS input into active level before communication with the master starts. In a single slave system it is not necessary to control the slave with SS, but it is often better to provide the pulse here too, to synchronize the slave with the beginning of each data sequence. The SS can be managed by both software and hardware (*Section 68.4.7*).

68.4.13 Disabling the SPI

To disable the SPI, it is mandatory to follow the disable procedures described in this paragraph.

In the Master mode, it is important to do this before the system enters a low-power mode when the peripheral clock is stopped, otherwise, ongoing transactions may be corrupted.

In Slave mode, the SPI communication can continue when the **spi_pclk** and **spi_ker_ck** clocks are stopped, without interruption, until any end of communication or data service request condition is reached. The **spi_pclk** can generally be stopped by setting the system into Stop mode. Refer to the RCC section for further information.

The master in full-duplex or transmit-only mode can finish any transaction when it stops providing data for transmission. In this case, the clock stops after the last data transaction. TXC flag can be polled (or interrupt enabled with EOTIE = 1) in order to wait for the last data frame to be sent.

When the master is in any receive-only mode, in order to stop the peripheral, the SPI communication must first be suspended, by setting the CSUSP bit to 1.

The data received but not read remain stored in RxFIFO when the SPI is suspended.

After such a software suspension, SPI must be always disabled to restart the internal state machine properly.

When SPI is disabled, RxFIFO is flushed. To prevent losing unread data, the user must ensure that RxFIFO is empty when disabling the SPI, by reading all remaining data (as indicated by the RXP, RXWNE and RXPLVL fields in the SPI_SR register).

The standard disable procedure is based on polling EOT and/or TXC status to check if a transmission session is (fully) completed. This check can be done in specific cases, too, when it is necessary to identify the end of ongoing transactions, for example:

- When the master handles SS signal by a GPIO not related to SPI (for example at case of multislave star topology) and it has to provide proper end of SS pulse for slave, or
- When transaction streams from DMA or FIFO are completed while the last data frame or CRC frame transaction is still ongoing in the peripheral bus.

When TSIZE>0, EOT and TXC signals are equal so polling of EOT is reliable at whatever SPI communication mode to check end of the bus activity. When TSIZE = 0, the user has to check TXC, SUSP or FIFO occupancy flags in according with the applied SPI mode and the way of the data flow termination.

The correct disable procedure in Master mode, except when receive-only mode is used, is:

1. Wait until TXC = 1 and/or EOT = 1 (no more data to transmit and last data frame sent). When CRC is used, it is sent automatically after the last data in the block is processed. TXC/EOT is set when CRC frame is completed in this case. When a transmission is suspended the software has to wait till CSTART bit is cleared.
2. Read all RxFIFO data (until RXWNE = 0 and RXPLVL = 00).
3. Disable the SPI (SPE = 0).

The correct disable procedure for master receive-only modes is:

1. Wait on EOT or break the receive flow by suspending SPI (CSUSP = 1).
2. Wait until SUSP = 1 (the last data frame is processed) if receive flow is suspended.
3. Read all RxFIFO data (until RXWNE = 0 and RXPLVL = 00).
4. Disable the SPI (SPE = 0).

In Slave mode, any on going data are lost when disabling the SPI.

Controlling the I/Os

As soon as the SPI is disabled, the associated and enabled AF outputs can still be driven by the device depending on the AFCNTR setting. When active output control is applied (AFCNTR = 1) and SPI is just been disabled (SPE = 0), the enabled outputs associated with SPI control signals (like SS and SCK at master and RDY at slave) can toggle immediately to inactive level (according to SSIOP and CPOL settings at master and RDIOP at slave respectively). The data line output (MOSI at master and MISO at slave) can instead change its level immediately at dependency on the actual TxFIFO content with the effect of potentially making invalid and no more guaranteed the value of the latest transacted bit on the bus. If necessary, the user has to take care about proper data hold time at the data line and avoid any eventual fast SPI disable just after the last data transaction is completed.

Note: Despite stability of the latest bit is guaranteed by design during the sampling edge of the clock, some devices can require even extension of this data bit stability interval during the sampling. It can be done for example by inserting small software delay between EOT event occurrence and SPI disable action.

68.4.14 Data packing

From user point of view there are two ways of data packing which can overlay each other:

- Type of access when data are written to TxFIFO or read from RxFIFO
  Multiple data can be pushed or fetched effectively by single access if data size is multiplied less than the access performed upon SPI_TXDR or SPI_RXDR registers.
- Number of data to be handled during the single software service
  It is convenient to group data into packets and cumulate the FIFO services overall the data packet content exclusively instead of handling data frame by frame separately. The user can define packets by FIFO threshold settings. Then all the FIFO occupancy events are related to that threshold level while required services are signalized by proper flags with interrupt and/or wake up capabilities.

When the data frame size fits into one byte (less than or equal to 8 bits), the data packing is used automatically when any read or write 16-bit or 32-bit access is performed on the SPI_RXDR/SPI_TXDR register. The multiple data frame pattern is handled in parallel in this case. At first, the SPI operates using the pattern stored in the LSB of the accessed word, then with the other data stored in the MSB.

Figure 848 provides an example of data packing mode sequence handling at full feature set instance. While DSIZE[4:0] is configured to 4-bit there, two or four data frames are written in the TxFIFO after the single 16-bit or 32-bit access the SPI_TXDR register of the transmitter. When the data frame size is between 9-bit and 16-bit, data packing is used automatically when a 32-bit access is done. Least significant half-word is used first. (regardless of the LSBFRST value)

This sequence can generate two or four RXP events in the receiver if the RxFIFO threshold is set frame (and data is read on a frame basis, unpacked), or it can generate a single RXP event if the FTHLV[3:0] field in the SPI_CFG1 register is programmed to a multiple of the frames to be read in a packed mode (16-bit or 32-bit read access).

The data are aligned in accordance with *Figure 847*. The valid bits are performed on the bus exclusively. Unused bits are not cared at transmitter while padded by zeros at receiver.

When short data frames (< 8-bit or < 16-bit) are used together with a larger data access mode (16-bit or 32-bit), the FTHLV value must be programmed as a multiple of the number

of frames/data access (multiple of 4 if 32-bit access is used to up to 8-bit frames or multiple of 2 if 16-bit access is used to up to 8-bit frames or 32-bit access to up to 16-bit frames.).

The RxFIFO threshold setting must always be higher or equal at least than the following read access size, as spurious extra data would be read otherwise.

The FIFO data access less than the configured data size is forbidden. One complete data frame must be always accessed at minimum.

A specific problem appears if an incomplete data packet is available at FIFO: less than threshold set at FTHLV bits.

There are two ways of dealing with this problem:

A. without using TSIZE field

On transmitter side, writing the last data frame of any odd sequence with an 8-bit/16-bit access to SPI_TXDR is enough.

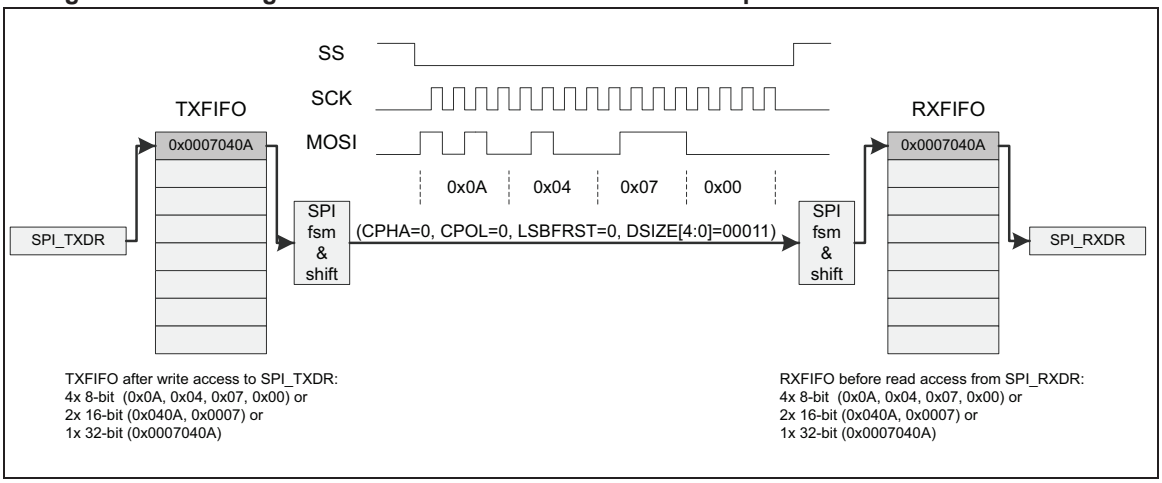
On receiver side, the remaining data may be read by any access. Any extra data read are padded with zeros. Polling the RXWNE and RXPLVL may be used to detect when the RX data are available in the RxFIFO. (A time out may be used at system level in order to detect the polling)

B. using the TSIZE field

On transmitter side, the transaction is stopped by the master when it faces EOT event.

In reception, the RXP flag is not set when EOT is set. In the case when the number of data to be received (TSIZE) is not a multiple of packet size, the number of remaining data is indicated by the RXWNE and RXPLVL fields in the SPI_SR register. The remaining data can be read by any access. Any extra read is padded by zeros.

Figure 848. Packing data in FIFO for transmission and reception at full feature set instance



1. DSIZE[4:0] is configured to 4-bit, data is right aligned, valid bits are performed only on the bus, their order depends on LSBFRST, if it is set, the order is reversed at all the data frames.

68.4.15 Communication using DMA (direct memory addressing)

To operate at its maximum speed and to facilitate the data register read/write process required to avoid overrun, the SPI features a DMA capability, which implements a simple request/acknowledge protocol.

A DMA access is requested when the TXDMAEN or RXDMAEN enable bits in the SPI_CFG1 register are set. Separate requests must be issued to the Tx and Rx buffers to fulfill service of the defined packet.

- In transmission, a series of DMA requests is triggered each time TXP is set to 1. The DMA then performs series of writes to the SPI_TXDR register.
- In reception, a series of DMA requests is triggered each time RXP is set to 1. The DMA then performs series of reads from the SPI_RXDR register. When EOT is set at the end of transaction and last data packet is incomplete then DMA request is activated automatically in according with RXWNE and RXPLVL[1:0] setting to read rest of data.

If the SPI is programmed in receive-only mode, UDR is never set.

If the SPI is programmed in a transmit mode, TXP and UDR can be eventually set at slave side, because transmit data may not be available. In this case, some data are sent on the TX line according with the UDR management selection.

When the SPI is used at a simplex mode, the user must enable the adequate DMA channel only while keeping the complementary unused channel disabled.

If the SPI is programmed in transmit-only mode, RXP and OVR are never set.

If the SPI is programmed in full-duplex mode, RXP and OVR are eventually set, because received data are not read.

In transmission mode, when the DMA or the user has written all the data to be transmitted (the TXTF flag is set at SPI_SR register), the EOT (or TXC at case TSIZE = 0) flag can be monitored to ensure that the SPI communication is complete. This is required to avoid corrupting the last transmission before disabling the SPI or before disabling the **spi_pclk** in Master mode. The software must first wait until EOT = 1 and/or TXC = 1.

When starting communication using DMA, to prevent DMA channel management raising error events, these steps must be followed in order:

1. Enable DMA Rx buffer in the RXDMAEN bit in the SPI_CFG1 register, if DMA Rx is used.
2. Enable DMA requests for Tx and Rx in DMA registers, if the DMA is used.
3. Enable DMA Tx buffer in the TXDMAEN bit in the SPI_CFG1 register, if DMA Tx is used.
4. Enable the SPI by setting the SPE bit.

To close communication it is mandatory to follow these steps in order:

1. Disable DMA request for Tx and Rx in the DMA registers, if the DMA issued.
2. Disable the SPI by following the SPI disable procedure.
3. Disable DMA Tx and Rx buffers by clearing the TXDMAEN and RXDMAEN bits in the SPI_CFG1 register, if DMA Tx and/or DMA Rx are used.

Data packing with DMA

If the transfers are managed by DMA (TXDMAEN and RXDMAEN set in the SPI_CFG1 register) the packing mode is enabled/disabled automatically depending on the PSIZE value configured for SPI TX and the SPI RX DMA channel.

If the DMA channel PSIZE value is equal to 16-bit and the SPI data size is less than or equal to 8-bit, then the packing mode is enabled. Similarly, If the DMA channel PSIZE value is equal to 32-bit and the SPI data size is less than or equal to 16-bit, then the packing mode is

enabled. The DMA then automatically manages the write operations to the SPI_TXDR register.

Regardless data packing mode is used and the number of data to transfer is not a multiple of the DMA data size (16-bit or 32-bit) while the frame size is smaller, DMA completes the transfer automatically in according with the TSIZE field setting.

Alternatively, last data frames may be written by software, in the single/unpacked mode.

To configure any DMA data access less than the configured data size is forbidden. One complete data frame must be always accessed at minimum.

68.4.16 Autonomous mode

The SPI is capable to handle and initialize transactions autonomously requiring no specific system execution interaction till the ongoing transaction ends.
Such autonomous transactions can be handled not only in Run or Sleep modes but even in Stop mode when the SPI logic is able to provide temporary clock requests addressed to the reset and clock controller (RCC) to ensure clocking of those SPI domains just necessary for handling the data flow between the memory and the peripheral interface at dependency in the SPI mode.

In Stop mode, the APB clock is requested by the peripheral each time the SPI registers need to be updated based on specific traffic events (mainly TXP and RXP). The required clock is provided by RCC if SPI autonomous mode is enabled at the RCC configuration and the SPI is clocked by an internal oscillator available in Stop mode.

Interrupts or DMA requests are then generated, depending on the SPI configuration. If no interrupt is enabled, the device remains in Stop mode. If DMA requests are enabled, the data are directly transferred to/from the SRAM thanks to the DMA while the device remains in Stop mode. If an enabled interrupt occurs, the device wakes up from Stop mode.

Note: The peripheral clock request stays pending till flag with enabled interrupt stays set. This is why it is important to service these pending requests and clear their flag as soon as possible at system sensitive to the low-power consumption especially and the application must acknowledge all pending interrupts events before switching the SPI to low-power mode.

Slave mode

When the SPI is configured as a standard slave and device is at Stop mode, the SPI kernel clock and the SPI APB clock are not provided permanently. All the data flow between the SPI interface and associated FIFOs is handled by an external SCK clock provided by outer master device within the serial interface clock domain. APB clock temporal requests are then based upon specific traffic events at dependency on the SPI configuration. As slave never initializes a transaction, there is no need to synchronize any transaction start in this mode.

Note: The peripheral clock request stays pending till flag with enabled interrupt stays set. This is why it is important to service these pending requests and clear their flag as soon as possible at system sensitive to the low-power consumption especially and the application must acknowledge all pending interrupts events before switching the SPI to low-power mode.

Master mode

The SPI operating in Master mode provides the SCK signal for outer slaves until the transaction is completed. The SCK signal is derived from the SPI clock generator running within the kernel clock domain fed from RCC upon kernel clock request provided by the SPI

when the device is in Stop mode. Temporal requests for APB clock are then based upon specific traffic events, depending upon the SPI configuration. The SPI master always initializes a transaction.

To minimize consumption in Stop mode, it is suggested to combine communication starts triggered by hardware (TRIGEN bit set in the SPI_AUTOCR register) and transfers of predefined data size (TSIZE > 0 in the SPI_CR2 register). This ensures that any APB clock request is suppressed between EOT handling and the next trigger event.

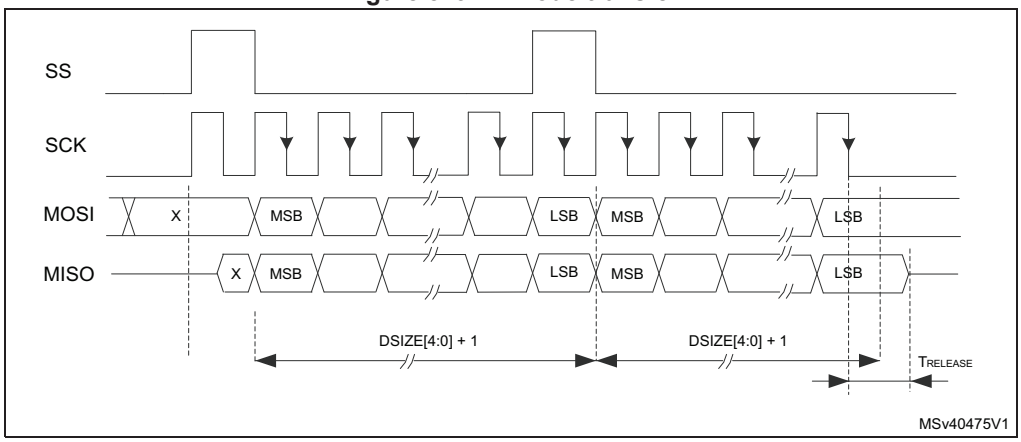
A transaction starts once the CSTART bit is set. In case of master transmitter, the TxFIFO must be filled by data, too. The CSTART bit can be written either by software or by hardware when a synchronous trigger is detected at Run, Sleep or Stop mode. The trigger source is selected by the TRIGSEL bits and enabled by the TRIGEN bit in the SPI_AUTOCR register. When the enabled trigger is detected, the transfer starts and continues by handling data till the EOT event or the transaction suspension. When the TRIGEN bit is changed, the user must prevent any trigger event occurrence. If the user cannot prevent that, the TRIGEN bit must be written while the SPI is disabled otherwise the peripheral behavior is not guaranteed.

68.5 SPI specific modes and control

68.5.1 TI mode

With a specific SP[2:0] bit field setting of the SPI_CFG2 register, the SPI can be configured compliant with the TI protocol. The SCK and SS signals polarity, phase and flow as well as the bits order are fixed so the setting of CPOL, CPHA, LSBFRST, SSOM, SSOE, SSIOP, SSM, RDIOP, RDIOM, MSSI and MIDI is not required when the SPI is in TI mode configuration. The SS signal synchronizes the protocol by pulses over the LSB data bit as it is shown in *Figure 849*.

Figure 849. TI mode transfer



In Slave mode, the clock generator is used to define time when the slave output at MISO pin becomes to HiZ when the current transaction finishes. The master baud rate setting (MBR[2:0] at SPI_CFG1) is applied and any baud rate can be used to determine this moment with optimal flexibility. The delay for the MISO signal to become HiZ ($T_{RELEASE}$) depends on internal re-synchronization, too, which takes next additional 2-4 periods of the clock signal feeding the generator. It is given by formula:

$$\frac{Tbaud}{2} + 2 \times Tspi_ker_ck \leq Trelease \leq \frac{Tbaud}{2} + 4 \times Tspi_ker_ck$$

If the slave detects misplaced SS pulse during data transaction the TIFRE flag is set.

68.5.2 SPI error flags

An SPI interrupt is generated if one of the following error flags is set and interrupt is enabled by setting the corresponding Interrupt Enable bit.

Overrun flag (OVR)

An overrun condition occurs when data are received by a master or slave and the RxFIFO has not enough space to store these received data. This can happen if the software or the DMA did not have enough time to read the previously received data (stored in the RxFIFO).

When an overrun condition occurs, the OVR flag is set and the newly received value does not overwrite the previous one in the RxFIFO. The newly received value is discarded and all data transmitted subsequently are lost. OVR flag triggers an interrupt if OVRIE bit is set. Clearing the OVR bit is done by a writing 1 to the OVRC bit in the SPI_IFCR. To prevent any next overrun event the clearing should be done after RxFIFO is emptied by software reads. It is suggested to release the RxFIFO space as much as possible, this means to read out all the available data packets based on RXP flag indication.

In Master mode, the user can prevent the RxFIFO overrun by automatic communication suspend (MASRX bit).

Underrun flag (UDR)

In a slave-transmitting mode, the underrun condition is captured internally by hardware if no data is available for transmission in the slave TxFIFO commonly. The UDR flag setting is then propagated into the status register by hardware (see note below). UDR triggers an interrupt if the UDRIE bit is set.

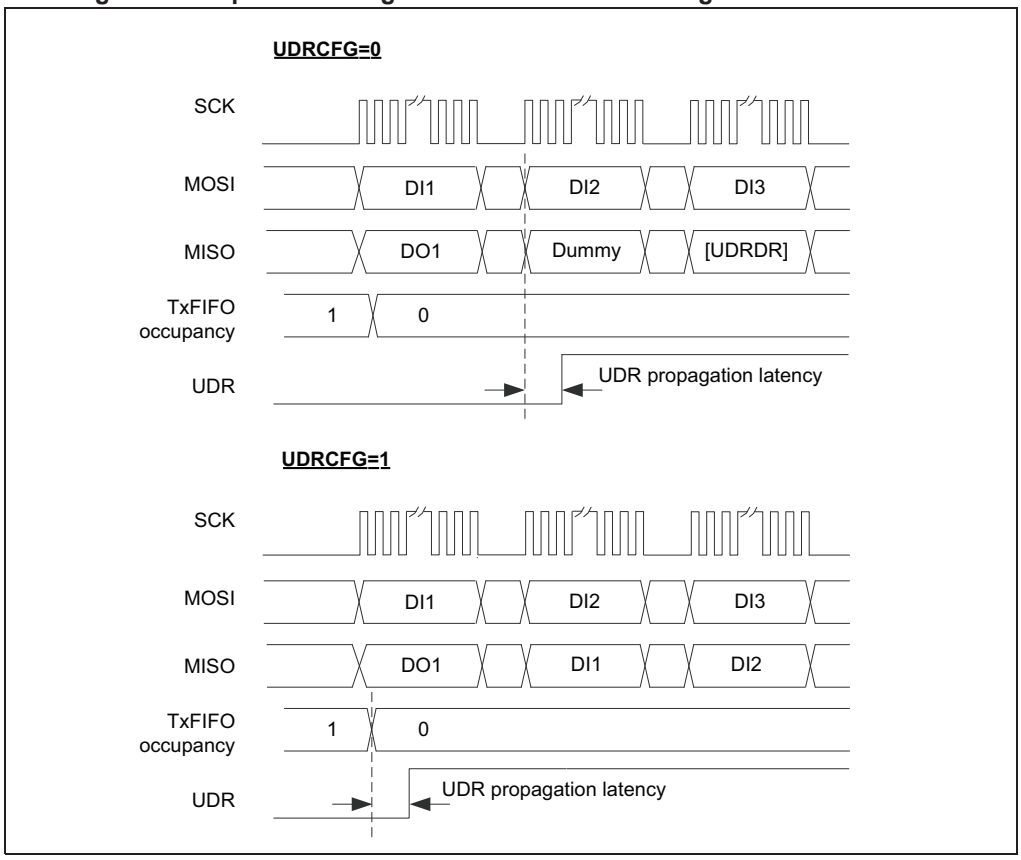
Underrun detection logic and system behavior depends on the UDRCFG bit. When an underrun is detected by slave, it can provide out either a constant pattern stored by the user at the UDRDR register or the data received previously from the master. When the first configuration (UDRCFG = 0) is applied, the underrun condition is evaluated whenever master starts to communicate a new data frame while TxFIFO is empty. Then single additional dummy (accidental) data is always inserted between last valid data and constant pattern defined at the UDRDR register (see *Figure 850*). Assuming that TxFIFO is not empty when master starts the communication, the underrun condition is evaluated just once the FIFO becomes empty during the next data flow. Valid data from TxFIFO is then upended by the lastly received data immediately.

The standard transmission is re-enabled once the software clears the UDR flag and this clearing is propagated into SPI logic by hardware. The user should write some data into TxFIFO prior clearing the UDR flag to prevent any next underrun condition occurrence capture.

The data transacted by slave is unpredictable especially when the transaction starts or continues while TxFIFO is empty and underrun condition is either not yet captured or just cleared. Typically, this is the case when SPI is just enabled or when a transaction with a defined size just starts. First bits can be corrupted in this case, as well, when slave software

writes first data into the empty TxFIFO too close prior the data transaction starts (propagation of the data into TxFIFO takes few APB clock cycles).

Figure 850. Optional configurations of slave detecting underrun condition



Note: The hardware propagation of an UDR event needs additional traffic on the bus. It always takes few extra SPI clock cycles after the event happens (both underrun captured by hardware and cleared by software). If clearing of the UDR flag by software is applied close to the end of data frame transaction or when SCK line is at idle in between the frames, next extra underrun pattern is sent initially by slave prior the valid data from TxFIFO becomes transacted again. The user can prevent this by SPI disable/enable action between sessions to restart the underrun logic and so initiate the next session by the valid data.

Mode fault (MODF)

Mode fault occurs when the master device has its internal SS signal (SS pin in SS hardware mode, or SSI bit in SS software mode) pulled low. This automatically affects the SPI interface in the following ways:

- The MODF bit is set and the SPI interrupt is triggered if the MODFIE bit is set.
- The SPE bit is forced to zero till MODF bit is set. This disables SPI and blocks all the peripheral outputs except the MODF interrupt request if enabled.
- The MASTER bit is cleared, thus forcing the device into Slave mode.

MODF is cleared by writing 1 to the MODFC bit in the SPI_IFCR.

To avoid any multiple slave conflicts in a system comprising several MCUs, the SS pin must be pulled to its non-active level before re-enabling the SPI, by setting the SPE bit.

As a security, hardware does not allow the SPE bit to be set while the MODF bit is set. In a slave device the MODF bit cannot be set except as the result of a previous multi master conflict.

A correct software procedure when master overtakes the bus at multi master system should be the following one:

- Switch into Master mode while SSOE = 0 (potential conflict can appear when another master occupies the bus. MODF is raised in this case, preventing any next node switching into Master mode)
- Put GPIO pin dedicated for another master SS control into active level
- Perform data transaction
- Put GPIO pin dedicated for another master SS control into non active level
- Switch back to Slave mode

CRC error (CRCE)

This flag is used to verify the validity of the value received when the CRCEN bit in the SPI_CFG1 register is set. The CRCE flag in the SPI_SR register is set if the value received in the shift register does not match the receiver SPI_RXCRC value, after the last data is received (as defined by TSIZE). The CRCE flag triggers an interrupt if CRCEIE bit is set. Clearing the bit CRCE is done by a writing 1 to the CRCEC bit in the SPI_IFCR.

TI mode frame format error (TIFRE)

A TI mode frame format error is detected when an SS pulse occurs during an ongoing communication when the SPI is operating in Slave mode and configured to conform to the TI mode protocol. When this error occurs, the TIFRE flag is set in the SPI_SR register. The SPI is not disabled when an error occurs, the SS pulse is ignored, and the SPI waits for the next SS pulse before starting a new transfer. The data may be corrupted since the error detection may result in the loss of few data frames.

The TIFRE flag is cleared by writing 1 to the TIFREC bit in the SPI_IFCR. If the TIFREIE bit is set, an interrupt is generated on the SS error detection. As data consistency is no longer guaranteed, communication should be re-initiated by software between master and slave.

68.5.3 CRC computation

Two separate 33-bit or two separate 17-bit CRC calculators are implemented to check the reliability of transmitted and received data. For instances with full feature set, the SPI offers CRC polynomial length from 5 to 33 bits when maximum data size is 32-bit and from 9 to 17 bits for the peripheral instances with data size limited to 16 bits. For instances with limited set of features, the CRC polynomial length can be set either to 9 or 17 only when data size is limited to 16 bit and optionally to 33 when data size is extended to 32-bit. The length of the polynomial is defined by the most significant bit of the value stored at the CRCPOLY register. It must be set greater than data frame length defined at DSIZE field. When

maximum data size is applied, the CRC33_17 bit must be set additionally to define the most significant bit of the polynomial string while keep its size always greater than data. The CRCSIZE field in the SPI_CFG1 then defines how many the most significant bits from CRC calculation registers are transacted and compared as CRC frame. It is defined independently from the data frame length, but it must be either equal or an integer multiple of the data frame size while its size cannot exceed the maximum data size of the instance.

To fully benefit from the CRC calculation capability, the polynomial length setting must correspond to the CRC pattern size, else the bits unused at the calculation are transacted and expected all zero at the end of the CRC pattern if its size is set greater then the polynomial length.

CRC principle

The CRC calculation is enabled by setting the CRCEN bit in the SPI_CFG1 register before the SPI is enabled (SPE = 1). The CRC value is then calculated using the CRC polynomial defined by the CRCPOLY register and CRC33_17 bit. When SPI is enabled, the CRC polynomial can be changed but only in case when there is no traffic on the bus.

The CRC computation is done, bit by bit, on the sampling clock edge defined by the CPHA and CPOL bits in the SPI_CR1 register. The calculated CRC value is checked automatically at the end of the data block defined by the SPI_CR2 register exclusively.

When a mismatch is detected between the CRC calculated internally on the received data and the CRC received from the transmitter, a CRCE flag is set to indicate a data corruption error. The right procedure for handling the CRC depends on the SPI configuration and the chosen transfer management.

CRC transfer management

Communication starts and continues normally until the last data frame must be sent or received in the SPI_DR register.

The length of the transfer must be defined by TSIZE. When the desired number of data is transacted, the TXCRC is transmitted and the data received on the line are compared to the RXCRC value.

No matter what is the CRCSIZE configuration, TSIZE cannot be set neither to 0xFFFF at full feature set instance nor to 0x3FF value at limited feature one if CRC is enabled.

In transmission, the CRC computation is frozen during CRC transaction and the TXCRC are transmitted, in a frame of length equal to the CRCSIZE field value.

In reception, the RXCRC is also frozen when desired number of data is transacted. Information to be compared with the RXCRC register content is then received in a frame of length equal to the CRCSIZE value.

Once the CRC frame is completed, an automatic check is performed comparing the received CRC value and the value calculated in the SPI_RXCRC register. The software has to check the CRCE flag in the SPI_SR register to determine if the data transfers were corrupted or not. Software clears the CRCE flag by writing 1 to the CRCEC.

The user takes no care about any flushing redundant CRC information, it is done automatically.

Resetting the SPI_TXCRC and SPI_RXCRC values

The SPI_TXCRC and SPI_RXCRC values are initialized automatically when new data is sampled after a CRC phase. This allows the use of DMA circular mode in order to transfer data without any interruption (several data blocks covered by intermediate CRC checking phases). Initialization patterns for receiver and transmitter can be configured either to zero or to all ones in dependency on setting bits TCRCINI and RCRCINI at SPI_CR1 register.

The CRC values are reset when the SPI is disabled.

68.6 SPI low-power modes

The SPI supports autonomous operation down to Stop mode, refer to *Section 68.4.16: Autonomous mode*.

Table 695. Effect of low-power modes on the SPI

Mode	Description
Sleep	No effect. SPI interrupts cause the device to exit Sleep mode.
Stop(1)	The SPI registers content is kept. If the autonomous mode is enabled at RCC configuration and SPI is clocked by an internal oscillator available in Stop mode, transfers are functional. The DMA requests are functional and the interrupts in these modes cause the device to exit Stop mode.
Standby(1)	The SPI instance not functional when this mode is powered down and must be reinitialized after exiting Standby mode. The SPI instance capable to work in this mode features DMA requests and wake-up capabilities.

1. Refer to *Section 68.3: SPI implementation* for information about wake-up from Stop mode support per instance as well as Standby mode availability. If an instance is not functional in a Stop mode, it must be disabled before entering this Stop mode.

68.7 SPI interrupts

Table 696 gives an overview of the SPI events capable to generate interrupts if enabled. Some of them feature wake up from Low-power mode capability, additionally. Most of them can be enabled and disabled independently while using specific interrupt enable control bits. The flags associated with the events are cleared by specific methods. Refer to description of SPI registers for more details about the event flags. When SPI is disabled, all the pending interrupt requests are blocked to prevent their propagation into the interrupt services, except the MODF interrupt request.

Table 696. SPI wake-up and interrupt requests

Interrupt vector	Interrupt event	Event flag	Enable Control bit	Event clear method	Exit from Stop and Standby modes capability(1)(2)
SPI	TxFIFO ready to be loaded (space available for one data packet - FIFO threshold)	TXP	TXPIE	TXP cleared by hardware when TxFIFO contains less than FTHLV empty locations	Yes
	Data received in RxFIFO (one data packet available - FIFO threshold)	RXP	RXPIE	RXP cleared by hardware when RxFIFO contains less than FTHLV samples	Yes
	Both TXP and RXP active	DXP	DXPIE	When TXP or RXP are cleared	Yes
	Transmission Transfer Filled	TXTF	TXTFIE	Writing TXTFC to 1	No
	Underrun	UDR	UDRIE	Writing UDRC to 1	Yes
	Overrun	OVR	OVRIE	Writing OVRC to 1	Yes
	CRC Error	CRCE	CRCEIE	Writing CRCEC to 1	Yes
	TI Frame Format Error	TIFRE	TIFREIE	Writing TIFREC to 1	No
	Mode Fault	MODF	MODFIE	Writing MODFC to 1	No
	End Of Transfer (full transfer sequence completed - based on TSIZE value)	EOT	EOTIE	Writing EOTC to 1	Yes
	Master mode suspended	SUSP		Writing SUSPC to 1	Yes
	TxFIFO transmission complete (TxFIFO empty)	TXC(3)		TXC cleared by hardware when a transmission activity starts on the bus	No

1. All the interrupt events are capable to wake up system from Sleep mode at each instance. For detailed information about instances capabilities to exit from concrete Stop and Standby mode refer to 'functionalities depending on the working mode' table.
2. Refer to *Section 68.3: SPI implementation* for information about Standby mode availability.
3. The TXC flag behavior depends on the TSIZE setting. When TSIZE>0, the flag fully follows the EOT one including its clearing by EOTC.

68.8 SPI registers

68.8.1 SPI control register 1 (SPI_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	IOLOCK
															rs

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TCRCINI	RCRCINI	CRC33_17	SSI	HDDIR	CSUSP	CSTART	MASRX	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SPE
rw	rw	rw	rw	rw	w	rs	rw								rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **IOLOCK**: locking the AF configuration of associated I/Os
This bit can be set by software only when SPI is disabled (SPE = 0). It is cleared by hardware exclusively whenever the SPE bit is changed from 1 to 0, either by hardware or software.
0: AF configuration is not locked
1: AF configuration is locked
When this bit is set, SPI_CFG2 register content cannot be modified. This bit is write-protected when SPI is enabled (SPE = 1).

Bit 15 **TCRCINI**: CRC calculation initialization pattern control for transmitter
0: all zero pattern is applied
1: all ones pattern is applied

Bit 14 **RCRCINI**: CRC calculation initialization pattern control for receiver
0: All zero pattern is applied
1: All ones pattern is applied

Bit 13 **CRC33_17**: 32-bit CRC polynomial configuration
0: Full size (33-bit or 17-bit) CRC polynomial is not used
1: Full size (33-bit or 17-bit) CRC polynomial is used

Bit 12 **SSI:** internal SS signal input level
This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the peripheral SS input internally and the I/O value of the SS pin is ignored.

Bit 11 **HDDIR:** Rx/Tx direction at Half-duplex mode
In Half-Duplex configuration the HDDIR bit establishes the Rx/Tx direction of the data transfer. This bit is ignored in Full-Duplex or any Simplex configuration.
0: SPI is receiver
1: SPI is transmitter

Bit 10 **CSUSP:** master suspend request
This bit reads as zero.
In Master mode, when this bit is set by software, the CSTART bit is reset at the end of the current frame and communication is suspended. The user has to check SUSP flag to check end of the frame transaction.
The Master mode communication must be suspended (using this bit or keeping TXDR empty) before going to Low-power mode.
After software suspension, SUSP flag must be cleared and SPI disabled and re-enabled before the next transaction starts.

Bit 9 **CSTART:** master transfer start
This bit can be set by software if SPI is enabled only to start an SPI communication. it is cleared by hardware when end of transfer (EOT) flag is set or when a transaction suspend request is accepted.
0: master transfer is at idle
1: master transfer is ongoing or temporary suspended by automatic suspend
In SPI mode, the bit is taken into account at master mode only. If transmission is enabled, communication starts or continues only if any data is available in the transmission FIFO.

Bit 8 **MASRX:** master automatic suspension in Receive mode

This bit is set and cleared by software to control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition.

0: SPI flow/clock generation is continuous, regardless of overrun condition. (data are lost)

1: SPI flow is suspended temporary on RxFIFO full condition, before reaching overrun condition. The SUSP flag is set when the SPI communication is suspended.

When SPI communication is suspended by hardware automatically, it may happen that few bits of next frame are already clocked out due to internal synchronization delay.

This is why, the automatic suspension is not quite reliable when size of data drops below 8 bits. In this case, a safe suspension can be achieved by combination with delay inserted between data frames applied when MIDI parameter keeps a non zero value; sum of data size and the interleaved SPI cycles should always produce interval at length of 8 SPI clock periods at minimum. After software clearing of the SUSP bit, the communication resumes and continues by subsequent bits transaction without any next constraint. Prior the SUSP bit is cleared, the user must release the RxFIFO space as much as possible by reading out all the data packets available at RxFIFO based on the RXP flag indication to prevent any subsequent suspension.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **SPE:** serial peripheral enable

This bit is set by and cleared by software.

0: Serial peripheral disabled.

1: Serial peripheral enabled

When SPE = 1, SPI data transfer is enabled, SPI_CFG1 and SPI_CFG2 configuration registers, CRCPOLY, UDRDR, part of SPI_AUTOCR register and IOLOCK bit in the SPI_CR1 register are write protected. They can be changed only when SPE = 0.

When SPE = 0 any SPI operation is stopped and disabled, all the pending requests of the events with enabled interrupt are blocked except the MODF interrupt request (but their pending still propagates the request of the spi_plck clock), the SS output is deactivated at master, the RDY signal keeps not ready status at slave, the internal state machine is reseted, all the FIFOs content is flushed, CRC calculation initialized, receive data register is read zero.

SPE is cleared and cannot be set when MODF error flag is active.

68.8.2 SPI control register 2 (SPI_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIZE[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **TSIZE[15:0]**: number of data at current transfer

When these bits are changed by software, the SPI must be disabled.

Endless transaction is initialized when CSTART is set while zero value is stored at TSIZE.

TSIZE cannot be set to 0xFFFF respective 0x3FFF value when CRC is enabled.

Note: TSIZE[15:10] bits are reserved at limited feature set instances and must be kept at reset value.

68.8.3 SPI configuration register 1 (SPI_CFG1)

Address offset: 0x08

Reset value: 0x0007 0007

The content of this register is write protected when SPI is enabled, except TXDMAEN and RXDMAEN bits.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BPASS	MBR[2:0]			Res.	Res.	Res.	Res.	Res.	CRCEN	Res.	CRCSIZE[4:0]				
rw	rw	rw	rw						rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXDMAEN	RXDMAEN	Res.	Res.	Res.	Res.	UDRCFG	FTHLV[3:0]				DSIZE[4:0]				
rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **BPASS:** bypass of the prescaler at master baud rate clock generator

0: bypass is disabled

1: bypass is enabled

Bits 30:28 **MBR[2:0]:** master baud rate prescaler setting

000: SPI master clock/2

001: SPI master clock/4

010: SPI master clock/8

011: SPI master clock/16

100: SPI master clock/32

101: SPI master clock/64

110: SPI master clock/128

111: SPI master clock/256

Note: MBR setting is considered at slave working at TI mode, too (see Section 68.5.1: TI mode).

Bits 27:23 Reserved, must be kept at reset value.

Bit 22 **CRCEN:** hardware CRC computation enable

0: CRC calculation disabled

1: CRC calculation enabled

Bit 21 Reserved, must be kept at reset value.

Bits 20:16 **CRCSIZE[4:0]:** length of CRC frame to be transacted and compared

Most significant bits are taken into account from polynomial calculation when CRC result is transacted or compared. The length of the polynomial is not affected by this setting.

00000: reserved
00001: reserved
00010: reserved
00011: 4-bits
00100: 5-bits
00101: 6-bits
00110: 7-bits
00111: 8-bits
.....
11101: 30-bits
11110: 31-bits
11111: 32-bits

The value must be set equal or multiply of data size (DSIZE[4:0]). Its maximum size corresponds to DSIZE maximum at the instance.

Note: The most significant bit at CRCSIZE bit field is reserved at the peripheral instances where data size is limited to 16-bit.

Bit 15 **TXDMAEN:** Tx DMA stream enable

0: Tx DMA disabled
1: Tx DMA enabled

Bit 14 **RXDMAEN:** Rx DMA stream enable

0: Rx-DMA disabled
1: Rx-DMA enabled

Bits 13:10 Reserved, must be kept at reset value.

Bit 9 **UDRCFG:** behavior of slave transmitter at underrun condition

0: slave sends a constant pattern defined by the user at the SPI_UDRDR register

1: Slave repeats lastly received data from master. When slave is configured at transmit only mode (COMM[1:0] = 01), all zeros pattern is repeated.

For more details see *Figure 850: Optional configurations of slave detecting underrun condition*.

Bits 8:5 **FTHLV[3:0]:** FIFO threshold level

Defines number of data frames at single data packet. Size of the packet should not exceed 1/2 of FIFO space.

0000: 1-data
0001: 2-data
0010: 3-data
0011: 4-data
0100: 5-data
0101: 6-data
0110: 7-data
0111: 8-data
1000: 9-data
1001: 10-data
1010: 11-data
1011: 12-data
1100: 13-data
1101: 14-data
1110: 15-data
1111: 16-data

SPI interface is more efficient if configured packet sizes are aligned with data register access parallelism:

– If SPI data register is accessed as a 16-bit register and DSIZE ≤ 8 bit, better to select FTHLV = 2, 4, 6.
– If SPI data register is accessed as a 32-bit register and DSIZE> 8 bit, better to select FTHLV = 2, 4, 6, while if DSIZE ≤ 8bit, better to select FTHLV = 4, 8, 12.

Note: FTHLV[3:2] bits are reserved at instances with limited set of features

Bits 4:0 **DSIZE[4:0]:** number of bits in at single SPI data frame

00000: not used
00001: not used
00010: not used
00011: 4 bits
00100: 5 bits
00101: 6 bits
00110: 7 bits
00111: 8 bits
.....
11101: 30 bits
11110: 31 bits
11111: 32 bits

Note: Maximum data size can be limited up to 16-bits at some instances. At instances with limited set of features, DSIZE[2:0] bits are reserved and must be kept at reset state. DSIZE[4:3] bits then control next settings of data size:
00xxx: 8-bits
01xxx: 16-bits
10xxx: 24-bits
11xxx: 32-bits.

68.8.4 SPI configuration register 2 (SPI_CFG2)

Address offset: 0x0C

Reset value: 0x0000 0000

The content of this register is write protected when SPI is enabled or IOLOCK bit is set at SPI_CR1 register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFCNTR	SSOM	SSOE	SSIOP	Res.	SSM	CPOL	CPHA	LSBFRST	MASTER	SP[2:0]			COMM[1:0]		Res.
rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IOSWP	RDIOP	RDIOM	Res.	Res.	Res.	Res.	Res.	MIDI[3:0]				MSSI[3:0]			
rw	rw	rw						rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **AFCNTR:** alternate function GPIOs control

This bit is taken into account when SPE = 0 only

0: The peripheral takes no control of GPIOs while it is disabled

1: The peripheral keeps always control of all associated GPIOs

When SPI must be disabled temporary for a specific configuration reason (for example CRC reset, CPHA or HDDIR change) setting this bit prevents any glitches on the associated outputs configured at alternate function mode by keeping them forced at state corresponding the current SPI configuration.

Bit 30 **SSOM**: SS output management in Master mode

This bit is taken into account in Master mode when SSOE is enabled. It allows the SS output to be configured between two consecutive data transfers.

0: SS is kept at active level till data transfer is completed, it becomes inactive with EOT flag

1: SPI data frames are interleaved with SS non active pulses when MIDI[3:0]>1

Bit 29 **SSOE:** SS output enable

This bit is taken into account in Master mode only

0: SS output is disabled and the SPI can work in multimaster configuration

1: SS output is enabled. The SPI cannot work in a multimaster environment. It forces the SS pin at inactive level after the transfer is completed or SPI is disabled with respect to SSOM, MIDI, MSSI, SSIOP bits setting

Bit 28 **SSIOP**: SS input/output polarity

0: low level is active for SS signal

1: high level is active for SS signal

Bit 27 Reserved, must be kept at reset value.

Bit 26 **SSM:** software management of SS signal input

0: SS input value is determined by the SS PAD

1: SS input value is determined by the SSI bit

When master uses hardware SS output (SSM = 0 and SSOE = 1) the SS signal input is forced to not active state internally to prevent master mode fault error.

Bit 25 **CPOL:** clock polarity

0: SCK signal is at 0 when idle

1: SCK signal is at 1 when idle

Bit 24 **CPHA:** clock phase

0: the first clock transition is the first data capture edge

1: the second clock transition is the first data capture edge

Bit 23 **LSBFRST:** data frame format

0: MSB transmitted first

1: LSB transmitted first

Bit 22 **MASTER:** SPI Master

0: SPI Slave

1: SPI Master

Bits 21:19 **SP[2:0]:** serial protocol

000: SPI Motorola

001: SPI TI

others: reserved, must not be used

Bits 18:17 **COMM[1:0]:** SPI Communication Mode
00: full-duplex
01: simplex transmitter
10: simplex receiver
11: half-duplex

Bit 16 Reserved, must be kept at reset value.

Bit 15 **IOSWP:** swap functionality of MISO and MOSI pins
0: no swap
1: MOSI and MISO are swapped
When this bit is set, the function of MISO and MOSI pins alternate functions are inverted. Original MISO pin becomes MOSI and original MOSI pin becomes MISO.

Bit 14 **RDIOP:** RDY signal input/output polarity
0: high level of the signal means the slave is ready for communication
1: low level of the signal means the slave is ready for communication

Bit 13 **RDIOM:** RDY signal input/output management
0: RDY signal is defined internally fixed as permanently active (RDIOP setting has no effect)
1: RDY signal is overtaken from alternate function input (at master case) or output (at slave case) of the dedicated pin (RDIOP setting takes effect)
Note: When DSIZE at the SPI_CFG1 register is configured shorter than 8-bit, the RDIOM bit must be kept at zero.

Bits 12:8 Reserved, must be kept at reset value.

Bits 7:4 **MIDI[3:0]**: master Inter-Data Idleness
Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in Master mode.
0000: no delay
0001: 1 clock cycle period delay
...
1111: 15 clock cycle periods delay
Note: This feature is not supported in TI mode.

Bits 3:0 **MSSI[3:0]**: Master SS Idleness
Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS opening a session and the beginning of the first data frame of the session in Master mode when SSOE is enabled.
0000: no extra delay
0001: 1 clock cycle period delay added
...
1111: 15 clock cycle periods delay added
Note: This feature is not supported in TI mode.
To include the delay, the SPI must be disabled and re-enabled between sessions.

68.8.5 SPI interrupt enable register (SPI_IER)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	MODFIE	TIFREIE	CRCEIE	OVRIE	UDRIE	TXTFIE	EOTIE	DXPIE	TXPIE	RXPIE
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:10 Reserved, must be kept at reset value.

Bit 9 **MODFIE**: mode Fault interrupt enable
0: MODF interrupt disabled
1: MODF interrupt enabled

Bit 8 **TIFREIE**: TIFRE interrupt enable
0: TIFRE interrupt disabled
1: TIFRE interrupt enabled

Bit 7 **CRCEIE:** CRC error interrupt enable
0: CRC interrupt disabled
1: CRC interrupt enabled

Bit 6 **OVRIE:** OVR interrupt enable
0: OVR interrupt disabled
1: OVR interrupt enabled

Bit 5 **UDRIE:** UDR interrupt enable
0: UDR interrupt disabled
1: UDR interrupt enabled

Bit 4 **TXTFIE:** TXTFIE interrupt enable
0: TXTF interrupt disabled
1: TXTF interrupt enabled

Bit 3 **EOTIE:** EOT, SUSP and TXC interrupt enable
0: EOT/SUSP/TXC interrupt disabled
1: EOT/SUSP/TXC interrupt enabled

Bit 2 **DXPIE**: DXP interrupt enabled
DXPIE is set by software and cleared by TXTF flag set event.
0: DXP interrupt disabled
1: DXP interrupt enabled

Bit 1 **TXPIE:** TXP interrupt enable
TXPIE is set by software and cleared by TXTF flag set event.
0: TXP interrupt disabled
1: TXP interrupt enabled

Bit 0 **RXPIE:** RXP interrupt enable
0: RXP interrupt disabled
1: RXP interrupt enabled

68.8.6 SPI status register (SPI_SR)

Address offset: 0x14

Reset value: 0x0000 1002

All the flags of this register are not cleared automatically when the SPI is re-enabled. They require specific clearing access exclusively via the flag clearing register as noted in the bits descriptions below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTSIZE[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXWNE	RXPLVL[1:0]		TXC	SUSP	Res.	MODF	TIFRE	CRCE	OVR	UDR	TXTF	EOT	DXP	TXP	RXP
r	r	r	r	r		r	r	r	r	r	r	r	r	r	r

Bits 31:16 **CTSIZE[15:0]**: number of data frames remaining in current TSIZE session

The value is not quite reliable when traffic is ongoing on bus or during autonomous operation in low-power mode.

Note: CTSIZE[15:0] bits are not available in instances with limited set of features.

Bit 15 **RXWNE:** RxFIFO word not empty

0: less than four bytes of RxFIFO space is occupied by data

1: at least four bytes of RxFIFO space is occupied by data

Note: This bit value does not depend on DSIZE setting and keeps together with RXPLVL[1:0] information about RxFIFO occupancy by residual data.

Bits 14:13 **RXPLVL[1:0]:** RxFIFO packing level

When RXWNE = 0 and data size is set up to 16-bit, the value gives number of remaining data frames persisting at RxFIFO.

00: no next frame is available at RxFIFO

01: 1 frame is available

10: 2 frames are available*

11: 3 frames are available*

Note: (*): Possible value when data size is set up to 8-bit only.

When data size is greater than 16-bit, these bits are always read as 00. In that consequence, the single data frame received at the FIFO cannot be detected neither by RWNE nor by RXPLVL bits if data size is set from 17 to 24 bits. The user must then apply other methods to detect the number of data received, such as monitor the EOT event when TSIZE > 0 or RXP events when FTHLV = 0.

Bit 12 **TXC:** TxFIFO transmission complete

The flag behavior depends on TSIZE setting.

When TSIZE = 0, the TXC is changed by hardware exclusively and it raises each time the TxFIFO becomes empty and there is no activity on the bus.

If TSIZE ≠ 0 there is no specific reason to monitor TXC as it just copies the EOT flag value including its software clearing. The TXC generates an interrupt when EOTIE is set.

This flag is set when SPI is reset or disabled.

0: current data transaction is still ongoing, data is available in TxFIFO or last frame transmission is on going.

1: last TxFIFO frame transmission complete

Bit 11 **SUSP:** suspension status

In Master mode, SUSP is set by hardware either as soon as the current frame is completed after CSUSP request is done or at master automatic suspend receive mode (MASRX bit is set at SPI_CR1 register) on RxFIFO full condition.

SUSP generates an interrupt when EOTIE is set.

This bit must be cleared prior SPI is disabled and this is done by writing 1 to SUSPC bit of SPI_IFCR exclusively.

0: SPI not suspended (Master mode active or other mode).

1: Master mode is suspended (current frame completed).

Bit 10 Reserved, must be kept at reset value.

Bit 9 **MODF**: mode fault

0: no mode fault

1: mode fault detected.

When MODF is set, SPE and IOLOCK bits of SPI_CR1 register are reset and setting SPE again is blocked until MODF is cleared.

This bit is cleared by writing 1 to MODFC bit of SPI_IFCR exclusively.

Bit 8 **TIFRE**: TI frame format error

0: no TI Frame Error

1: TI frame error detected

This bit is cleared by writing 1 to TIFREC bit of SPI_IFCR exclusively.

Bit 7 **CRCE:** CRC error

0: no CRC error

1: CRC error detected

This bit is cleared when SPI is re-enabled or by writing 1 to CRCEC bit of SPI_IFCR optionally.

Bit 6 **OVR:** overrun

0: no overrun

1: overrun detected

This bit is cleared when SPI is re-enabled or by writing 1 to OVRC bit of SPI_IFCR optionally.

Bit 5 **UDR:** underrun

0: no underrun

1: underrun detected

This bit is cleared when SPI is re-enabled or by writing 1 to UDRC bit of SPI_IFCR optionally.

Note: In SPI mode, the UDR flag applies to Slave mode only.

Bit 4 **TXTF:** transmission transfer filled

0: upload of TxFIFO is ongoing or not started

1: TxFIFO upload is finished

TXTF is set by hardware as soon as all of the data packets in a transfer have been submitted for transmission by application software or DMA, that is when TSIZE number of data have been pushed into the TxFIFO.

This bit is cleared by software write 1 to TXTFC bit of SPI_IFCR exclusively.

TXTF flag triggers an interrupt if TXTFIE bit is set.

TXTF setting clears the TXPIE and DXPIE masks so to off-load application software from calculating when to disable TXP and DXP interrupts.

Bit 3 **EOT:** end of transfer

EOT is set by hardware as soon as a full transfer is complete, that is when SPI is re-enabled or when TSIZE number of data have been transmitted and/or received on the SPI. EOT is cleared when SPI is re-enabled or by writing 1 to EOTC bit of SPI_IFCR optionally.

EOT flag triggers an interrupt if EOTIE bit is set.

If DXP flag is used until TXTF flag is set and DXPIE is cleared, EOT can be used to download the last packets contained into RxFIFO in one-shot.

0: transfer is ongoing or not started

1: transfer complete

In master, EOT event terminates the data transaction and handles SS output optionally. When CRC is applied, the EOT event is extended over the CRC frame transaction.

To restart the internal state machine properly, SPI is strongly suggested to be disabled and re-enabled before next transaction starts despite its setting is not changed.

Bit 2 **DXP**: duplex packet

0: TxFIFO is Full and/or RxFIFO is Empty

1: both TxFIFO has space for write and RxFIFO contains for read a single packet at least

DXP flag is set whenever both TXP and RXP flags are set regardless SPI mode.

Bit 1 **TXP:** Tx-packet space available

0: not enough free space at TxFIFO to host next data packet

1: enough free space at TxFIFO to host at least one data packet

TXP flag can be changed only by hardware. Its value depends on the physical size of the FIFO and its threshold (FTHLV[3:0]), data frame size (DSIZE[4:0] in SPI mode), and actual communication flow. If the data packet is stored by performing consecutive write operations to SPI_TXDR, TXP flag must be checked again once a complete data packet is stored at TxFIFO. TXP is set despite SPI TxFIFO becomes inaccessible when SPI is reset or disabled.

Bit 0 **RXP:** Rx-packet available

0: RxFIFO is empty or an incomplete data packet is received

1: RxFIFO contains at least one data packet

The flag is changed by hardware. It monitors the total number of data currently available at RxFIFO if SPI is enabled. RXP value depends on the FIFO threshold (FTHLV[3:0]), data frame size (DSIZE[4:0] in SPI mode), and actual communication flow. If the data packet is read by performing consecutive read operations from SPI_RXDR, RXP flag must be checked again once a complete data packet is read out from RxFIFO.

68.8.7 SPI interrupt/status flags clear register (SPI_IFCR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SUSPC	Res.	MODFC	TIFREC	CRCEC	OVRC	UDRC	TXTFC	EOTC	Res.	Res.	Res.
				w		w	w	w	w	w	w	w			

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **SUSPC:** Suspend flag clear
Writing a 1 into this bit clears SUSP flag in the SPI_SR register

Bit 10 Reserved, must be kept at reset value.

Bit 9 **MODFC**: mode fault flag clear
Writing a 1 into this bit clears MODF flag in the SPI_SR register

Bit 8 **TIFREC**: TI frame format error flag clear
Writing a 1 into this bit clears TIFRE flag in the SPI_SR register

Bit 7 **CRCEC:** CRC error flag clear
Writing a 1 into this bit clears CRCE flag in the SPI_SR register

Bit 6 **OVRC:** overrun flag clear
Writing a 1 into this bit clears OVR flag in the SPI_SR register

Bit 5 **UDRC:** underrun flag clear
Writing a 1 into this bit clears UDR flag in the SPI_SR register

Bit 4 **TXTFC:** transmission transfer filled flag clear
Writing a 1 into this bit clears TXTF flag in the SPI_SR register

Bit 3 **EOTC:** end of transfer flag clear
Writing a 1 into this bit clears EOT flag in the SPI_SR register

Bits 2:0 Reserved, must be kept at reset value.

68.8.8 SPI autonomous mode control register (SPI_AUTOCR)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRIGEN	TRIGPOL	TRIGSEL[3:0]			
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **TRIGEN:** HW control of CSTART triggering enable
0: HW control disabled
1: HW control enabled

Note: if user cannot prevent trigger event during write, the TRIGEN must be changed when SPI is disabled

Bit 20 **TRIGPOL:** trigger polarity
0: trigger is active on raising edge
1: trigger is active on falling edge
Note: This bit can be written only when SPE = 0.

Bits 19:16 **TRIGSEL[3:0]**: trigger selection (refer *Section : Description of SPI interconnections*).
0000: spi_trg0 is selected
0001: spi_trg1 is selected
...
1111: spi_trg15 is selected
Note: these bits can be written only when SPE = 0.

Bits 15:0 Reserved, must be kept at reset value.

68.8.9 SPI transmit data register (SPI_TXDR)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXDR[31:16]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXDR[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:0 **TXDR[31:0]:** transmit data register
The register serves as an interface with TxFIFO. A write to it accesses TxFIFO.
Note: data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read.
Note: DR can be accessed byte-wise (8-bit access): in this case only one data-byte is written by single access.
halfword-wise (16 bit access) in this case 2 data-bytes or 1 halfword-data can be written by single access.
word-wise (32 bit access). In this case 4 data-bytes or 2 halfword-data or word-data can be written by single access.
Write access of this register less than the configured data size is forbidden.

68.8.10 SPI receive data register (SPI_RXDR)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RXDR[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDR[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **RXDR[31:0]:** receive data register

The register serves as an interface with RxFIFO. When it is read, RxFIFO is accessed.

Note: data is always right-aligned. Unused bits are read as zero when the register is read. Writing to the register is ignored.

Note: DR can be accessed byte-wise (8-bit access): in this case only one data-byte is read by single access

halfword-wise (16 bit access) in this case 2 data-bytes or 1 halfword-data can be read by single access

word-wise (32 bit access). In this case 4 data-bytes or 2 halfword-data or word-data can be read by single access.

Read access of this register less than the configured data size is forbidden.

68.8.11 SPI polynomial register (SPI_CRCPOLY)

Address offset: 0x40

Reset value: 0x0000 0107

The content of this register is write protected when SPI is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CRCPOLY[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRCPOLY[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **CRCPOLY[31:0]:** CRC polynomial register

This register contains the polynomial for the CRC calculation.

The default 9-bit polynomial setting 0x107 corresponds to default 8-bit setting of DSIZE. It is compatible with setting 0x07 used in other ST products with fixed length of the polynomial string, where the most significant bit of the string is always kept hidden.

Length of the polynomial is given by the most significant bit of the value stored in this register. It must be set greater than DSIZE. CRC33_17 bit must be set additionally with CRCPOLY register when DSIZE is configured to maximum 32-bit or 16-bit size and CRC is enabled (to keep polynomial length grater than data size).

Note: CRCPOLY[31:16] bits are reserved at instances with data size limited to 16-bit. There is no constrain when 32-bit access is applied at these addresses. Reserved bits 31-16 are always read zero while any write to them is ignored.

68.8.12 SPI transmitter CRC register (SPI_TXCRC)

Address offset: 0x44

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXCRC[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **TXCRC[31:0]:** CRC register for transmitter

When CRC calculation is enabled, the TXCRC[31:0] bits contain the computed CRC value of the subsequently transmitted bytes. CRC calculation is initialized when the CRCEN bit of SPI_CR1 is set or when a data block is transacted completely. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPOLY register.

The number of bits considered at calculation depends on SPI_CRCPOLY register and CRCSIZE bits settings at SPI_CFG1 register.

Note: A read to this register when the communication is ongoing may return an incorrect value.

Note: TXCRC[31-16] bits are reserved at instances with data size limited to 16-bit. There is no constrain when 32-bit access is applied at these addresses. Reserved bits 31-16 are always read zero while any write to them is ignored.

Note: The configuration of CRCSIZE bit field is not taken into account when the content of this register is read by software. No masking is applied for unused bits in this case.

68.8.13 SPI receiver CRC register (SPI_RXCRC)

Address offset: 0x48

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RXCRC[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXCRC[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **RXCRC[31:0]:** CRC register for receiver

When CRC calculation is enabled, the RXCRC[31:0] bits contain the computed CRC value of the subsequently received bytes. CRC calculation is initialized when the CRCEN bit of SPI_CR1 is set or when a data block is transacted completely. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPOLY register.

The number of bits considered at calculation depends on SPI_CRCPOLY register and CRCSIZE bits settings at SPI_CFG1 register.

Note: A read to this register when the communication is ongoing may return an incorrect value.

RXCRC[31-16] bits are reserved at the peripheral instances with data size limited to 16-bit. There is no constrain when 32-bit access is applied at these addresses. Reserved bits 31-16 are always read zero while any write to them is ignored.

Note: The configuration of CRCSIZE bit field is not taken into account when the content of this register is read by software. No masking is applied for unused bits in this case.

68.8.14 SPI underrun data register (SPI_UDRDR)

Address offset: 0x4C

Reset value: 0x0000 0000

The content of this register is write protected when SPI is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UDRDR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UDRDR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **UDRDR[31:0]:** data at slave underrun condition

The register is taken into account in Slave mode and at underrun condition only. The number of bits considered depends on DSIZE bit settings of the SPI_CFG1 register. Underrun condition handling depends on setting UDRCFG bit at SPI_CFG1 register.

Note: UDRDR[31-16] bits are reserved at the peripheral instances with data size limited to 16-bit. There is no constraint when 32-bit access is applied at these addresses. Reserved bits 31-16 are always read zero while any write to them is ignored.

68.8.15 SPI register map

Table 697. SPI register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SPI_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	IOLOCK	TCRCINI	RCRCINI	CRC33_17	SSI	HDDIR	CSUSP	CSTART	MASRX	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SPE
	Reset value																0	0	0	0	0	0	0	0	0								0
0x04	SPI_CR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSIZE[15:0]															
	Reset value																	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	0	0	0	0	0	0	0	0	0	0
0x08	SPI_CFG1	BPASS	MBR[2:0]			Res.	Res.	Res.	Res.	Res.	CRCEN	Res.	CRCSIZE[4:0]					TXDMAEN	RXDMAEN	Res.	Res.	Res.	Res.	UDRCFG	FTHLV[3:0]				DSIZE[4:0]				
	Reset value	0	0	0	0						0		1	1	1	1	1	0	0					0	0(1)	0(1)	0	0	0	0(1)	1(1)	1(1)	1(1)
0x0C	SPI_CFG2	AFCNTR	SSOM	SSOE	SSIOP	Res.	SSM	CPOL	CPHA	LSBFRST	MASTER	SP[2:0]			COMM [1:0]		Res.	IOSWP	RDIOP	RDIOM	Res.	Res.	Res.	Res.	Res.	MIDI[3:0]				MSSI[3:0]			
	Reset value	0	0	0	0		0	0	0	0	0	0	0	0	0	0		0	0	0						0	0	0	0	0	0	0	0
0x10	SPI_IER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MODFIE	TIFREIE	CRCEIE	OVRIE	UDRIE	TXTFIE	EOTIE	DXPIE	TXPIE	RXPIE
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x14	SPI_SR	CTSIZE[15:0](1)																RXWNE	RXPLVL [1:0]		TXC	SUSP	Res.	MODF	TIFRE	CRCE	OVR	UDR	TXTF	EOT	DPXP	TXP	RXP
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		0	0	0	0	0	0	0	0	1	0
0x18	SPI_IFCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SUSPC	Res.	MODFC	TIFREC	CRCEC	OVRC	UDRC	TXTFC	EOTC	Res.	Res.	Res.
	Reset value																					0		0	0	0	0	0	0	0			

Table 697. SPI register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	SPI_AUTOCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRIGEN	TRIGPOL	TRIGSEL[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value											0	0	0	0	0	0																
0x20	SPI_TXDR	TXDR[31:16]																TXDR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24-0x2C	Reserved	Reserved																															
0x30	SPI_RXDR	RXDR[31:16]																RXDR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x34 - 0x3C	Reserved	Reserved																															
0x40	SPI_CRCPOLY	CRCPOLY[31:16](2)																CRCPOLY[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1
0x44	SPI_TXCRC	TXCRC[31:16](2)																TXCRC[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	SPI_RXCRC	RXCRC[31:16](2)																RXCRC[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x4C	SPI_UDRDR	UDRDR[31:16](2)																UDRDR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The bitfield is reserved for instances with limited set of features and it must be kept at reset value. For more details, refer to the concrete register description in *Section 68.8: SPI registers*.
2. The bits 31-16 are reserved for the peripheral instances with data size limited to 16-bit. There is no constrain when the 32-bit access is applied at these addresses. The bits 31-16, when reserved, are always read to zero while any write to them is ignored.

Refer to *Section 2.3* for the register boundary addresses.