Guidelines for enhanced SPI communication on STM32 MCUs and MPUs

Introduction

The serial peripheral interface (SPI) enables easy data transfer between peripherals and the microcontroller. It has a wide range of specific modes and possible configurations, hence the need for specific handling and settings. An optimized peripheral handling decreases the overall system load.

This document provides useful information to users targeting specific configurations, and provides tips on how to prevent and manage the most frequent difficulties encountered when handling SPI communication.

The comparison of different versions implemented on STM32 products can help users considering migration to a new MCU/MPU.

The reader must be already familiar with the basic SPI principles and peripheral configuration options.

The information given here does not replace that in reference manuals and datasheets available at www.st.com, but rather complements it. The available SPI features are product dependent, refer to reference manuals and datasheets for their detailed description.

1 General information

This document applies to STM32 Arm®-based devices.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Released versions

The SPI peripheral for STM32 devices has evolved over time. The following table summarizes the main differences between active versions.

Table 1. Main SPI features on STM32 devices

Feature/Version	1.2.x	1.3.x	2.x.x(1)	3.x.x(1)
STM32 series	F1, F2, F4, L0, L1	C0, F0, F3, F7, G0, G4, L4, L4+, L5, U0, WB, WL	H7, MP1	H5, MP2, U3, U5, WBA
Data size	8- or 16-bit	4- to 16-bit	4- to 16/32-bit	
Tx & Rx FIFOs	No	2x 4-byte	2x 4- to 32-byte	
Data packing by data-register access	No	Yes		
Data packing by packets	No		Yes	
Dual clock domain (APB and kernel)	No		Yes	
Programmable transfer counters	No		Yes	
Underrun detection/configuration	No		Yes	
Autonomous operation in low-power modes	No		Yes(2)	Yes(3)
Master transfer suspension	No		Yes	
Master automatic suspension	No		Yes	
Flushing content of FIFOs	No		Yes	
Master data/SS interleaving	No		Yes	
GPIOs alternate function control when SPI is disabled	No		Yes	
Configuration protection by locking	No		Yes	
Swap of MOSI/MISO, reversed SS logic	No		Yes	
RDY signal option with suspension	No			Yes
Master input data sampling delay	No			Yes(3)

1. *Derivative instances exist where data size configuration or some other features can be limited. Size of the FIFOs and I2S audio specific protocol support always depend on the instance implementation.*
2. *Limited capability; not fully supported.*
3. *Dependent upon implementation, not available on all products.*

2.1 Differences between versions

The different SPI versions have some similarities, as an example, all of them feature DMS and the capability to wake up from low-power mode. Some features, such as the inter-IC sound (I2S) support and the enhanced slave-select modes, are supported as option. The main differences are related to data size, data buffering, dual-clock domain, and programmable transfer counters, described in the following subsections.

Data size

Data size can be fixed and multiplied by 8-bit, or can be adjustable by bit.

Data buffering of Tx and Rx streams

The earlier versions support only single register pair to handle both the streams, while more recent versions use a set of registers collected at dedicated FIFOs.

Dual-clock domain

The older versions use a single-peripheral clock source, which feeds both the peripheral interface and the kernel.

More recent versions feature an autonomous run at low-power mode under kernel or also under external clock when the system peripheral interface clock is stopped (refer to Figure 1). This capability is enabled because in recent versions the kernel clock is separated from the clock feeding the system peripheral bus (APB) interface.

Programmable transfer counters

The specific control related to "end of transaction" actions such as slave-select (SS) management, CRC (cyclic redundancy check) append, update of the FIFO data threshold, or termination of data streams can be performed by a proper software action, but ideally it should be performed automatically by hardware using predefined counters.

Earlier versions of SPI do not feature the programmable counters and DMA overtakes the task, using its settings. The latest versions feature embedded counters, hence SPI takes over control of programmable counters action via the SPI configuration. In these cases, the DMA role is limited to manage data transfers.

2.2 Frequency constraints

The bus bandwidth depends upon the frequency(ies) applied to the associated clock domain(s), supposing that there is enough margin to handle all the fast data flow in time (see Section 4.1).

In systems featuring a single clock domain, the theoretical limit is up to half the APB clock applied for the peripheral in both master and slave configurations.

On dual clock domain systems, there are no specific constraints concerning the ratio between clock feeding APB and the kernel domains. In case of significant differences, the user must respect the number of clock periods needed to synchronize and propagate the signals shared between the domains (such as changes of flags or registers or data transfers). For the same reason, some functionalities can be limited when the data size is too short, for example the underrun protection logic at slave or handling an optional RDY status signal between master and slave.

The main factor impacting the bandwidth is the distortion of fast signals due to the capacitance of their external connections versus the throughput of the associated internal GPIO paths and associated alternate function logic. Temperature and low-supply voltage have an impact too: significant differences at the highest accessible communication speed can be observed between SPI instances (or their mapping options), SPI modes, power supply or temperature ranges. The real limits (verified by characterization or guaranteed by design) are detailed in the datasheets.

3 Data flow handling principle

The SPI hardware interface is based on a pair of physically separated internal shift registers, which handle the input and output serial bit streams on MOSI and MISO pins, synchronously clocked by the SCK signal.

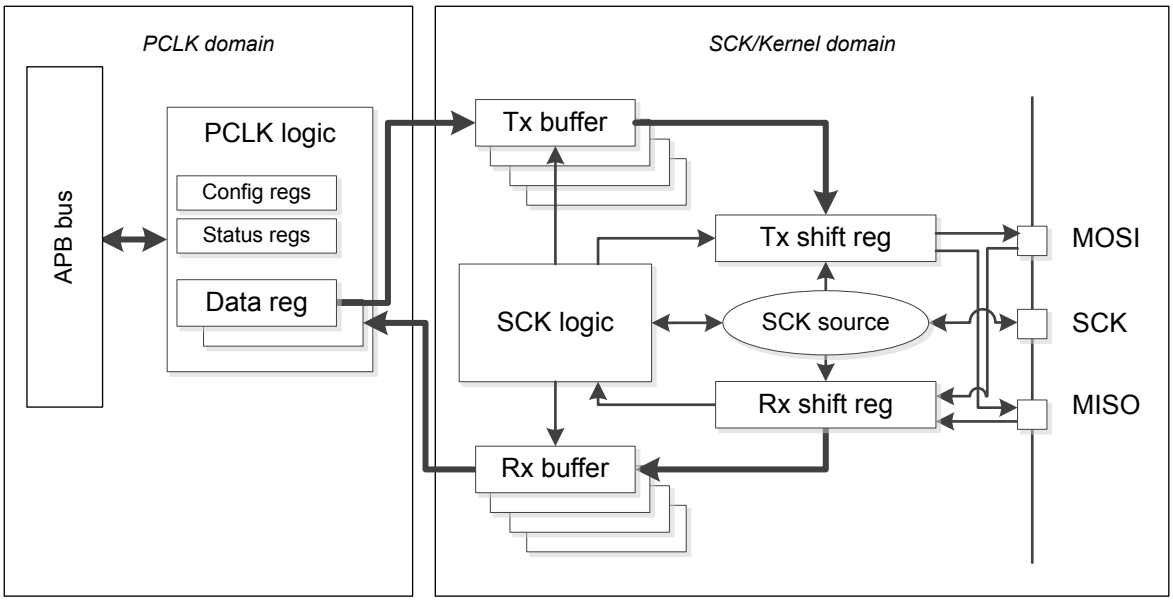
The SPI configuration (if it operates as a slave or a master) determines which pin works as input or as output, and whether the SCK clock source is external or internal. As shown in the figure below, data are clocked in and out through the MISO and MOSI pins via shift registers associated with the RAM storage area providing separated Tx and Rx buffers. In early SPI versions, each of these buffers is made by a single-data register. More recent SPI versions feature an extended set of registers, which work under the FIFO principle. The content of the buffers is accessible via read or write accesses to physically doubled SPI data registers allocated under a sigle common address in the APB (PCLK) clock domain. All the other peripheral configurations and status registers are allocated to this domain.

During transmission, once the content of the Tx shift register dedicated to data output is shifted out completely, the oldest data is moved into it from the associated Tx buffer. The space formerly occupied is released, and the buffer is ready to accept new data.

During reception, once the Rx shift register dedicated to data input is clocked in completely, the new data frame is moved from it into the associated Rx buffer.

When the peripheral is configured as slave, the SCK clock is always provided by an external source. When configured as master, the SCK clock is sourced internally from kernel (providing either the peripheral interface APB bus clock or a specific separated kernel clock). This clock signal divided by embedded clock baud rate generator feeds the outer serial interface of the SCK signal mastering communication with the slave nodes.

Figure 1. Simplified data flow scheme



In full-duplex communication, the data handshake for transmission and reception is done in parallel. Each new data frame write access precedes reading of the corresponding pattern transferred at the same time, because the transmitted data frame must be available at the Tx buffer before its transfer begins. When no new data are available, SPI master suspends the communication, but the slave forced to continue operation faces data underrun. It provides invalid data even without any detection if underrun fetaure is not supported. The next data can be written into the buffer once there is enough space to store them.

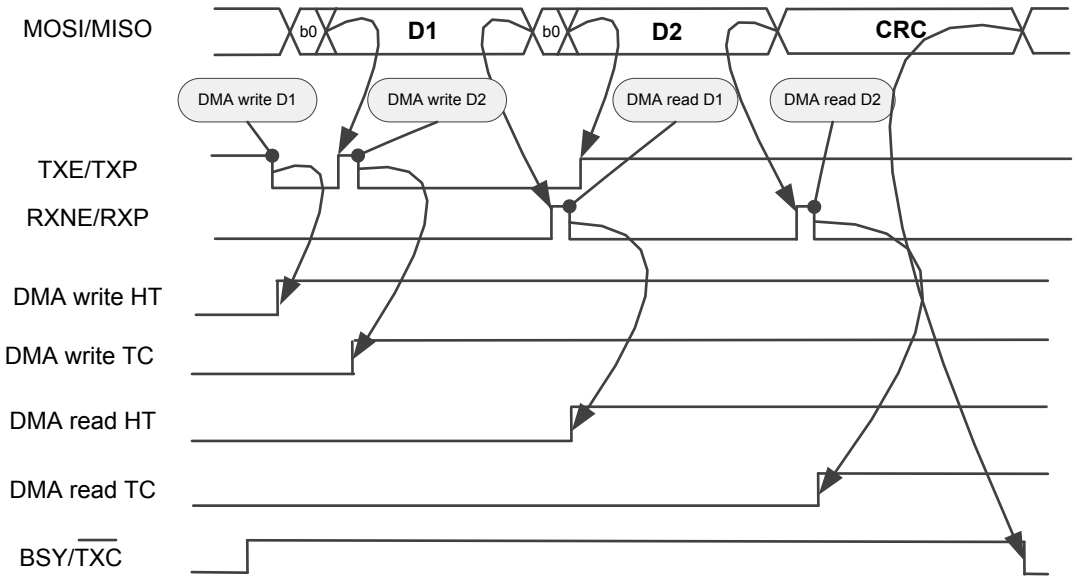
The first bit is always provided out from the Tx buffer, and the data buffered for transmission are moved into to internal Tx shift register after its first bit is fully transferred out. That is why the value of the first transferred bit is always dependent on the actual buffer content, while the rest of the data-frame already provided from the associated shift register is fixed. As a consequence, data output signal can toggle independently on SCK signal prior and during the first bit transfer period, depending upon the Tx buffer access and its content, and upon clock configuration.

The transfer between the buffer and the shift register can take place almost immediately after the initial data is written into the buffer at transfer start, because the buffer becomes empty as soon as the first bit of the data is transferred. When the buffer has a FIFO with sufficient capacity, it can accept the next data almost immediately. The receive buffer initially is always empty, as all data are copied into it after the ongoing data frames are completed.

The following figure provides a simplified example of a continuous SPI communication handled by DMA: the size of the data corresponds to a single DMA access of the SPI data register. The DMA write stream is completed earlier than the read stream, while the transfer on the bus can continue after both read and write DMA streams are completed (if CRC is applied).

Even if not necessary, it is advised to monitor the BSY or EOT/TXC flags at the end of the transfer to prevent either a premature SPI disable or the kernel clock-source removal. In this case the ongoing flow can be corrupted or wrongly terminated. The BSY flag becomes low between data frames if the flow is not continuous and it always drops at slave for a single SCK clock-period between data frames (even if the SCK flow is continuous). The latest SPI versions replace the BSY signal by the EOT and TXC flags. Single-data handling events signalized by TXE and RXNE flags are replaced by TXP and RXP or DXP flags, which handle the data packets events. The timing of these signals is almost the same across all versions. BSY flag monitoring is not suggested to handle the data flow.

Figure 2. Timing of SPI data events and DMA transfer complete flags during a transfer



4 Data flow potential problems

The most frequent problems are grouped in three categories:

- System performance and data flow
  - Ratio between APB and kernel clock
  - Ratio between master and slave performance
  - Frequency of the SPI and of the DMA events
- Specific modes handling
  - Data corruption by a premature termination
  - Handling exact number of data
  - SPI reconfiguration and handling of associated GPIOs
  - Handling half-duplex operations via a single bidirectional data line
- Specific signals handling
  - Internal interface signals
  - Optional external interface signals

Specific handling of autonomous SPI operation in low power modes is out of this document scope. Version 2.x.x is capable to drive serial bus clocking in stop mode while raising asynchronous interrupts to handle data exchange. Version 3.x.x, additionally, can handle the data exchange between the peripheral and memory via smart DMA if implemented in the product. This is done without need for system wake-up, due to temporal domain's clock requests handled by the peripheral while the data transfers can be synchronized with other peripherals by internal triggers. Refer to documents targeting LPBAM control, such as AN5645 "*STM32U5 Series power optimization using LPBAM*"

4.1 System performance and data flow issues

The SPI bus is potentially fast and the system must be high-performance to handle the data flow in time. Depending on the SPI mode, the system must prevent data overrun in reception, and data underrrun in transmission. The ability to handle all the data efficiently can become critical when the flow (based on the SCK clock signal) is continuous, the data frame is short, and the SCK clock is comparable or faster than the system core clock.

A practical example is a situation when the baud rate is PCLK / 2, and data frame is shortened to 4 bits. The system faces two SPI events within each interval of 8 PCLK periods in full Duplex-mode. In this case, if PCLK is similar (or equal to) the system clock, there is not enough margin at system level to handle such continuous bidirectional communication, even when all the available CPU performance is spent in an optimized program loop polling directly the SPI flags. The system has not enough margin because it must test the associated occupancy flags in the SPI status register, and handle the transfer between the SPI data register and the memory if the corresponding flag is active.

See below a simplified piece of C-code loop (it assumes that R1 and R2 CPU registers keep the Tx and Rx data pointers towards the memory, while R3 register keeps the base address of the SPI registers):

```
while (1) {
  if ((SPI->SR & SPI_SR_TXE) == 0) {
    SPI->DR = *tx_data++;
  }
  if ((SPI->SR & SPI_SR_RXNE) != 0) {
    *rx_data++ = SPI->DR;
  }
}
```

The assembled result of the compilation of the C-code is shown below:

```
??main_1:
   LDR      R0, [R3, #+08]    ;test TXE flag
   LSLS     R0,R0,#+30
   BMI.N    ??main_2
   LDRB     R0, [R1], #+1    ;read and send next data from a memory
   STRB     R0, [R3, #+12]
??main_2:
   LDR      R0, [R3, #+08]    ;test RXNE flag
   LSLS     R0,R0,#+31
   BPL.N    ??main_1
   LDRB     R0, [R3, #+12] ;store next data received to a volatile memory
   STRB     R0, [R2], #+1
   B.N         ??main_1
```

Even a simplified loop of code like this cannot be executed within eight CPU cycles, despite the C-compiler is configured for code speed optimization on this Cortex® M3 example. The execution of this loop is critical even within 16 cycles available with a standard 8-bit data-frame configuration.

If the SPI events are handled by interrupts, the system must perform the same number of cycles to detect the pending event and for the required memory transfer inside the interrupt service execution as in polling loop. It spends additional cycles for service entry and return, necessary to safe and restore the interrupted main execution context at stack. That is why a solution based on interrupts is not useful in this case.

A possible solution is to implement DMA, but the user must not overestimate the bandwidth to provide a real solution to the problem. DMA still needs a few system cycles to perform the required data transfer, while the execution of the related service can be postponed because another system operation is ongoing. Each DMA transfer must first go through the bus matrix arbitration process (winner takes the matrix path to make the transfer), hence the transfer and the associated arbitration takes up to seven system-clock cycles.

Note that due to a round-robin scheme principle applied to share the bus matrix bandwidth, the transfer request addressed to DMA can have significant latency after it is raised by the peripheral. The latency can be caused by a simultaneous request from another DMA channel or by the execution of a non-interruptible data-transfer sequence performed by the CPU. Examples of such sequences are the stack context handling (either at entry to or at return from an interrupt service, which normally takes eight cycles), or a simultaneous data-batch transfer such as a LDMIA instruction execution, which can take up to fourteen consecutive cycles. If all these factors add up, the delay to serve a pending DMA request can exceed few tens of system-clock cycles, and multiple 4-bit data frames can be missed on the SPI bus.

Software or DMA unable to handle data on time

There are three main indicators indicating that data management is not done on time (by software or by DMA), even if peripheral and system configurations are applied correctly. These are data corruption, data overrun and data underrun.

- **Data corruption (data flow is not as expected)**
  - Some data patterns are missing, wrongly repeated or in any way partially corrupted.
  - Note that when correct order of bits is shifted by one or more bits in the frames, this situation reflects a synchronization issue between master and slave rather than a late handling data by system.
  - Some actions that help to identify data-corruption problems are:
    - The implementation of a HW CRC checksum at the end of each transfer batch.
    - Any data redundancy check such as repetition of the messages, plausibility check (comparison of the message with the expected content) or any other similar predefined protocol.
- **Data overrun (RxFIFO space overflow)**
  - A system performance issue during reception is put in evidence when an OVR error flag is raised.
  - Data overrun is mainly a slave device problem.
  - Data overrun can also appear on a master device configured in Receive-only mode, because the SCK clock signal is provided continuously and independently on the data-buffering process, except for devices featuring master automatic suspension of the communication mode when the RxFIFO is full (see MASRX bit control in the reference manuals).
- **Data underrun (TxFIFO space underflow)**
  - The primary cause is low system performance during transmission.
  - Data underrun occurs only on slave devices because master always suspends the transmission when its Tx data buffer becomes empty and there are no more data to transmit. A master configured in I2S mode works differently in case of underrun.
  - The most recent devices feature configurable behaviour and specific indication (via UDR flag) in case of underrun. The device can repeat a constant pattern defined by the user or apply the latest received pattern, behaving as a simple shift register (applicable at a multi-slave daisy-chain circular topology). Repetition of the last transferred data frame is useful in I2S mode to achieve a smooth audio output.
  - For devices not featuring UDR flag, the underrun event is hidden. Some side effects can be visible, such as a flow corruption by repetition of a data pattern previously transferred (the value depends upon the design). For example, when FIFO is implemented, the oldest FIFO pattern is applied(1).
  - Data underrun depends upon the timing between the moments when data to transmit are written to the data register by slave and the start of its transfer by master. If slave manages to write the data within its first bit transfer period in spite of the underrun condition, the new data can be accepted for transmission, but there is a potential risk of corruption of the first transferred bit. This depends on the LSBFIRST bit setting if the most or the least significant bit of the data frame is considered. If the write comes later, the underrun pattern is completely transferred while new data are buffered for the next transfer.

1. *When a 32-bit TxFIFO of a slave is fully filled by 0xAA, 0xBB, 0xCC, 0xDD. and a transfer of five 8-bit patterns is applied by master without any other update of the slave's TxFIFO, the slave output is 0xAA, 0xBB, 0xCC, 0xDD, 0xAA.*

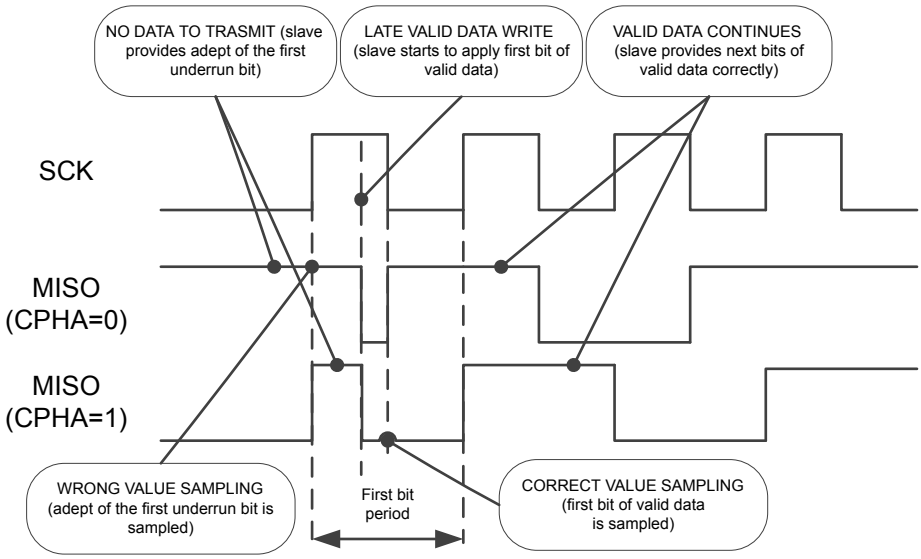
In the SPI slave configuration, when a last valid data bit is processed out and the data register of the TxFIFO is empty (node does not have next data to transfer), the value of the first underrun bit is taken from a previously transferred pattern. The data correctness is unsure, as the behaviour of the MISO slave output depends on many factors (clock-phase setting, timing between an occasional late-data write versus the beginning of the next transfer handled by the SCK signal, value of the underrun candidate bit versus the first new valid one to transmit).

If the CPHA bit is set, the MISO keeps the level of the last valid bit transferred, and the first underrun bit appears on the bus with the leading SCK edge of the next transaction if no new data are applied. If the CPHA bit is cleared, the value of potential underrun bit appears on the bus right after the last data transfer is completed, even if the underrun condition is not yet filled (valid data can still be written in time before the next data-frame transaction starts). In this case, the MISO line propagates (with possible toggle) the correct level immediately once the valid data is written into the data register and propagated into the empty SPI Tx buffer while node becomes ready for the data transfer in this configuration.

A typical observation of a late data-write correlated with the next data frame transfer start is the slave output peak toggle. This output is present prior or during the transfer of the first bit of a frame still provided by the buffer. This situation supposes that the value of the adept underrun bit and the corresponding bit at the new data are different, else there is no toggle and the switch between the underrun and the valid data source is not visible.

The following figure shows a typical deformation of the first bit when the software or the DMA writes late the new data to be transmitted, while the transfer of the first bit is already ongoing (or close to the moment when the data transfer starts). For illustration purposes, the valid bit of the next-data frame has opposite value than the underrun candidate to highlight the change of the MISO signal. The valid bit value is sampled correctly only if the write of the valid data preceeds the bit sampling edge, else, the underrun adept bit value is wrongly sampled. Wrong sampling is performed at configuration of CPHA = 0 when the data write comes too late after the sampling is performed at the beginning of the period.

Figure 3. A late write (underrun) of data while first-bit transfer is ongoing



A correct management of value sampling is observed when CPHA = 1, despite the data are written with significant delay. This is possible due to the fact that sampling is provided later at the middle of the bit period. Correct data transaction continues for the rest of the data frame at both configuration cases because data is written at the first-bit transaction period, so the transfer between Tx buffer and Tx shift register is still accepted.

4.1.1 Recommended methods to handle frequent SPI and DMA events

There are different possible ways to handle frequent SPI and DMA events, but their applicability depends upon the design. Older devices do not necessarily feature data FIFOs or widely configurable data size and they have limited configuration capabilities compared to recent devices.

The handling all SPI data events by interrupts is not the better method with fast baud rates, as repetition of these requests is very frequent.

Optimize performance is achieved when DMA takes care of all the data flow and other specific events associated with the transfer are polled by software or monitored by interrupts enabled by selective flags. The user must always check the status of error flags to detect problems with bus throughput or with system performance. In some cases, even optimized system performance is not enough for a fast SPI flow, and the user must apply additional methods to secure sufficient bandwidth.

Some effective actions to improve the management of frequent SPI and DMA events are:

- **Decrease the bus rate**: the ratio between the system and the kernel clock increases.
- **Increase the data size.**
- **Collect data into packets** when the software fills in and reads out the data.
- **Balance** the data register access with the data size and the threshold level of the FIFOs or of the data configuration setting. More information is detailed in the following section.
- **Slow down the data rate** leading to discontinuous clock by using fixed or enhanced adaptable methods of the temporal communication suspension between provided data frames or sessions.

These actions decrease the frequency and the number of data handling events. The system performance improves, thanks to a better management of their detection.

To decrease the latency when serving the raised events, temporarily disable interrupts or DMA channels from other peripherals, and set up and order priorities of the DMA channels.

Suppressing interrupts minimizes the CPU contribution in the round-robin bus matrix distribution scheme. The user must avoid automatic register refresh or memory dump windows during debug to avoid decreasing the BUS matrix throughput. When DMA interrupts are enabled, the half DMA transfer-completion interrupt must be suppressed if not used. Servicing this event becomes critical when a small number of continuous data is transferred by the DMA session or when frequent DMA data-handling services face a bad system performance (refer to the DMA HT and TC events on Figure 2).

At master side, the data rate flow can be slowed down by opting out from a continuous SCK flow (if applicable). The latest SPI versions feature optional programmable fixed interleaving gaps between frames, or between the SS signal that becomes active and first data transfer at session begin, and they can also implement an automatic suspension of the master. This temporary suspension of the master's transfers prevents data underflow and overflow, and can be achieved by monitoring the RxFIFO occupancy (MASRX feature) or by monitoring additional specific status-ready signal (RDY), which can be optionally provided by the slave. In oldest designs, the handling of master continuous simplex receiver mode is problematic. It can be replaced by full duplex mode with ignored data output when communication timing control is done by filling dummy transmission data.

Other features present in latest SPI designs, which simplify the data-handling processes are:

- Capability to set up data counters
- Capability to use an optional coupled dual-flag event to control both transmission and reception of a common event with a single software service

A dual flag puts in evidence that both transmission and reception flags are active. When a double event like this one is handled at master level, there is a risk that the flow becomes discontinuous. Discontinuity may occur when new data is written and its corresponding transfer is started after the previous one is finished, hence the bus becomes idle.

The dual-flag handling must be avoided in slave mode when the master is continuously providing a clock signal and the packet size is set close to half of FIFO space. There is a high risk of data underrun, due to a postponed service of the pending TXP event (which always precedes a RXP event).

The main sense of the DXP appears when targeting low-power applications, especially when the data-handling services are cumulated and their number is decreased to minimum, which prevents system from its frequent wake ups. The counterpart for that is slowing down the data rate.

4.1.2 Balanced handling of communication events

A correct and optimized data handling is achieved by balancing the data access with the configured data size and packing. The user configures the threshold levels of the data buffer, which determine the frequency of data handling events.

The data register can be read or write and can be single or multiple access (the amount of access correspond to the number of data at the packet). The data register access is always equal to or higher than the selected data size.

A wider access of the SPI data register can be provided by software or by DMA; the access corresponds to a multiple of the data size. In this case, data packing is automatically applied during the read or write of its content. For example, four data with size up to 8 bits can be read or written in parallel by a single 32-bit access of the data register. The type of access is defined by casting the address of the data register at code level.

In recent devices, the user can additionally define longer data packets and cumulate series of data-register accesses on a single event. The number of accesses must correspond to defined threshold of the FIFO. The events then signal that just the packet service is available and a next sequence of repeated data accesses is expected. For example, if a 32-bit access is applied to data register, a sequence of three of them must be applied upon a single packet event when data packet is defined as 12x 8-bit data. This feature helps to limit the number of data handling events and decreases the system's load in a significant way as data batch is always serviced in a single run. Internal data exchange between data buffer and shift register is automatically handled based on *one by one* data access sequences.

A TXE corresponding to a TXP event is raised once the buffer for transmission can accept the next data respective to the complete data packet. The transmission buffer is ready once the first bit of the data which is releasing the necessary space is transferred out.

The Tx buffer may become empty and the SCK clock signal can stop and become non continuous if the master faces a significant latency when servicing a raised event that releases space in the transmission buffer. When the same situation happens while the master continues the transfer, the slave faces a data underrun condition.

There is a risk of first-bit corruption whenever the service latency is close to the time needed for a single packed data transfer, where the next data are written to the empty buffer of the slave when its first bit is already sampled. Refer to Figure 3 for a related example. This case can also be observed at MISO toggle.

Ideally the packet size should not exceed half of the FIFO space. This configuration allows that a subsequent additional packet to be applied early after the current packet transfer starts. This allows a big potential margin for servicing next packet close to a whole single-packet transfer duration.

If the data packet size configuration is equal or almost equal to the overall FIFO space and if there is a continuous flow on the bus, the available time to store next data or to read the received data becomes similar to a single-data transfer duration. In this situation, the user must wait until the space corresponding to a complete packet is either released at the TxFIFO during a transfer or that the space corresponding to a complete packet is fully completed at receive data side at RxFIFO.

Under the above configuration, even a small service latency may generate a transfer problem and data packing becomes useless. That type of configuration makes sense for interleaved SPI batches with a number of data to be completely accommodated at FIFO space. In that case, the application has enough room to handle data once a batch is completed before the next batch starts.

When the Full-duplex mode is applied, the number of the overall data transferred (transmitted and received) is balanced. The transferred data goes in parallel with the common clock signal and the handling of both data directions is shifted each time. Transmission handling must precede the reception handling, otherwise the master does not start or continue any transaction, and the slave faces underrun whenever there is not available data ready at the Tx buffer while a transfer is ongoing.

In recent devices, a specific dual flag DXP is applied to enable common services to handle both transmission and reception flows together. This functionality is especially useful in low-power modes.

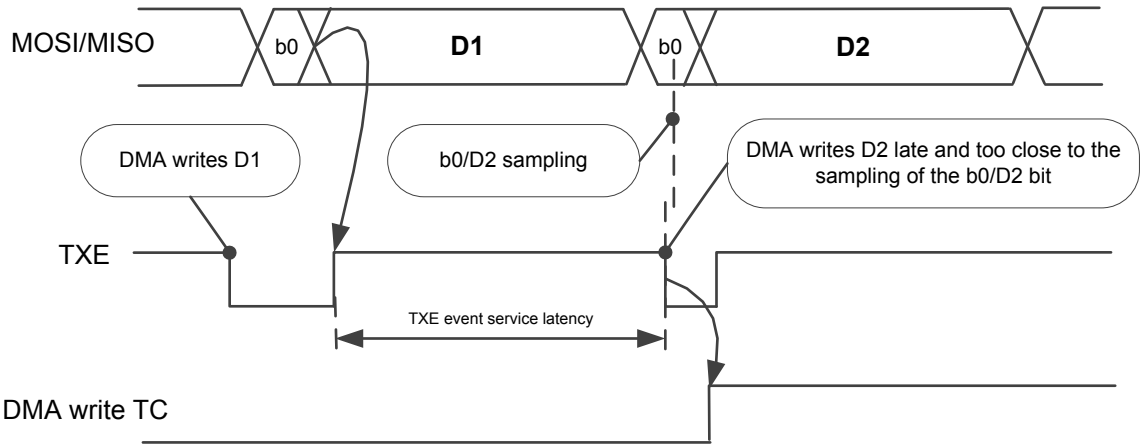
Some actions to cope with different timing of the both data flows are:

- Store some data for transmission in advance
- Handle the middle of the transfer by common DXP events
- Finish the session by handling the remaining received data at the end of the session (for example by an EOT event).

Note: A DXP event interrupt is suppressed by hardware once all the data for transmission is submitted.

The following figure is an example of a TXE/TXP event service latency limit available at slave for DMA. The example shows a continuous transfer of two consequent data configured with respect to available buffer size what involves to write D2 in time after D1 data is released from the buffer. The data is released from the buffer once its first bit is transferred on the bus. If D1 is a data packet, TXP signal (replacing the TXE one then) can raise later during D1 transfer once a space large enough to accommodate a full D2 packet is released and available at the buffer to accept new data.

Figure 4. **DMA late service of TXE/TXP event**



4.2 Handling specific SPI modes

SPI transfers are based on a predefined exact number of data. Some enhanced SPI modes are sensitive to deterministic data-counting and require specific control-handling related actions close to the end of the data transfer.

4.2.1 End of the bus activity detection

Specific situations that require proper detection of the bus activity-end to prevent corruption of ongoing data flow by premature fatal terminating accesses are:

- SS signal control
- Necessity to disable the peripheral after the session is completed due to system entry into low-power mode, the peripheral clock removal or change the peripheral configuration (for example, change of the data direction at Half-duplex mode or setup another configuration of the SCK signal or baud rate, or simply reset the internal state machine before next session start).

The end of the bus activity can be detected easily by RXNE or RXP flags, signaling end of last data reception. Exceptions are when the CRC frames are upended at the end of a transfer to validate the data (refer to Figure 1), or when SPI is used exclusively in transmission mode. In such cases, the signals to monitor the end of the bus activity are BSY, EOT, or TXC flags. Any monitoring of DMA complete events is not applicable, as SPI can be still transmitting (refer to Figure 2).

For BSY monitoring, additional specific measurements (such as timeout monitoring) are needed, because this flag is not fully reliable on old devices. BSY should be cleared by hardware between data frames when communication is not continuous or when it is fully completed; otherwise it highlights a synchronization problem between master and slave.

4.2.2 Specific aspects when SPI is disabled

Once the end of the bus activity is detected correctly, SPI can be disabled, but consider some specific aspects when doing so.

The FIFO content handling while SPI is disabled depends upon the design.

- On versions 1.3.x, the content of the Rx FIFO is preserved and accessible. Disabling SPI is a standard procedure to terminate the simplex-receive modes.
- On versions 2.x.x and higher, software suspension is used to terminate whatever ongoing flow. Tx and Rx FIFO contents are flushed, lost and impossible to access when SPI is disabled.

Another problem related to peripheral disable is the control of the associated GPIOs:

- Versions 1.x.x: the peripheral takes no control of the associated GPIOs when it is disabled. If the SPI signals are kept in alternate function configuration, they float if not supported by external resistors.
- Versions 2.x.x and higher: the peripheral can keep the GPIO control even when it is disabled (depending on IOs alternate function configuration managed by the AFCNTR bit) to prevent unexpected changes of the associated master outputs especially. In the contrary, when user apply SPI configurations affecting the IOs configured in the control keeping mode (such as change of the SCK signal polarity), new default levels are propagated out immediately despite SPI is still disabled. Be careful with desynchronization between master and slave, and perform the configuration change exclusively when no slave is selected.

SPI must be disabled before the system entries into Halt mode and between the transactions when SPI must be reconfigured or to restart the internal state machine properly. Even if SPI disable between sessions is not mandatory when configuration is not changed, it is recommended to ensure correct functionality, especially when reconfiguration or complex functionalities (CRC calculation, counting data configured to a non standard data size, counting data whose number is not aligned with a configured data packet, recovery from transaction suspension done by software) are applied.

The SPI must not be disabled before the communication is fully completed, and it should be as short as possible at slave, especially between sessions, to prevent missing the next communication start.

The most recent SPI designs are edge sensitive for hardware SS signal detection; if the master provides an active edge of the SS while the SPI is still temporarily disabled, the slave might miss the start of the slave select-signal and the following session. This is not the case on older SPI design where the NSS detection is level sensitive.

4.2.3 Time-critical control at the end of transfer

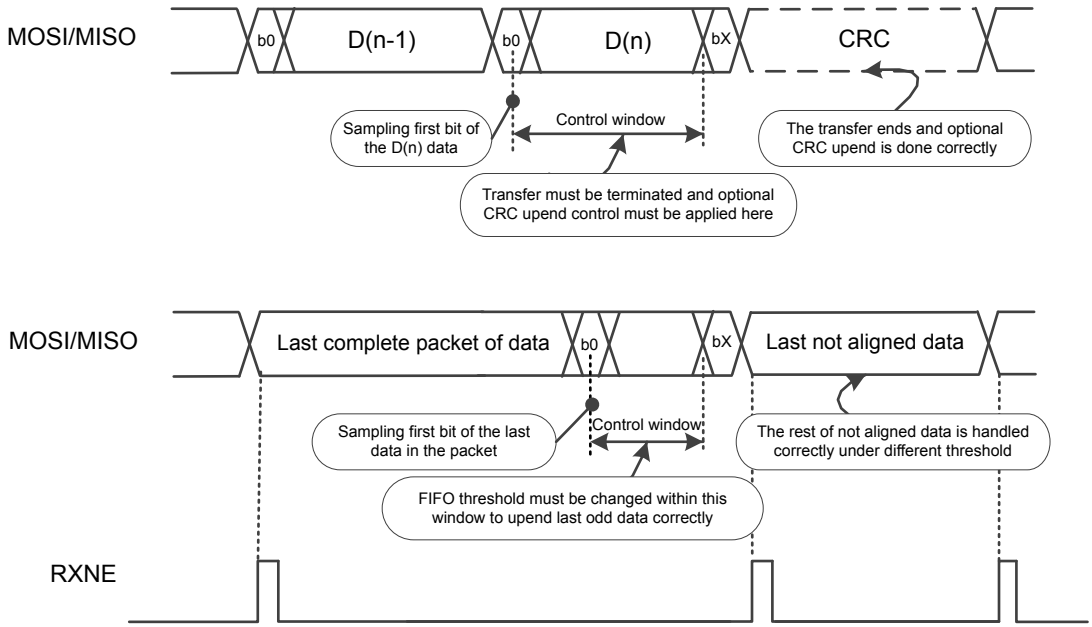
There are specific cases requiring a time-critical access before the transfer ends within a specific control window, just during last data handling:

- Change of the FIFO threshold configuration to handle data not aligned correctly with data or packet size
- At the end the CRC pattern control of data validation
- Proper termination of continuous data-flow like master configured in Receive-only mode to prevent any unexpected additional dummy-data transfer and ensure that the required data are transferred. In earlier versions, terminate this mode requires nonstandard SPI disable access during ongoing traffic. This is the case when SPI disable must be applied when communication is still ongoing on the bus.

It is practically impossible to provide such a control when it is based on software polling or interrupt services upon the associated events. There is a very short time frame available during the ongoing transfer of the last data frame, and there is a high risk of corruption when this window is missed and the required action is serviced due to a latency outside this interval.

The available control window starts after sampling the first bit of the frame, and ends before the begin of the last bit transfer time slot. The following figure is an example of an upend CRC frame pattern after the last data (or handling reception end) when data is not aligned with the FIFO threshold. In these cases, once the last complete data packet is processed, a final incomplete data must be sent. The same window is applied when continuous clock flow in receive-only mode must be terminated properly at master to receive the expected number of data while preventing next dummy data reception.

Figure 5. Examples of window available to perform specific control at the end of a transfer



If the device features programmable data counters, a specific control is done autonomously at the end of each transfer. In these cases, the user can operate with the last not-complete packet or with a standard complete packet. Only consistent data are applied (redundant reads give zero data, redundant writes are ignored). If the number of data is not known, it is better to use a single-data threshold (packet), assuming that the system has enough performance to handle on time all the data flow one by one. No packet event appears until the complete packet is received. The user must check the FIFO occupancy flags at the end of the transfer if a not aligned number of data is received.

As older devices does not feature embedded data counters, it is advised to use DMA and data counters instead. DMA performs the required control action autonomously at hardware level, hence the risk to miss the available control window is lower compared to an action performed by software. User must correctly set up the data or DMA counters to ensure handling of proper number of data on Tx and Rx side (especially when a CRC pattern has to be added). The size of the CRC pattern must be excluded from the data counting number and the CRC pattern is received to RxFIFO from which it must be always flushed out by software after the transfer upended by CRC is finished.

The following table provides an overview of the differences at the CRC settings and capabilities. More details can be found on the corresponding product's reference manual.

Table 2. Differences of CRC control between active SPI versions

Feature/Version	1.2.x	1.3.x	2.x.x	3.x.x
Setting of DMA data counters	Number of data excluding the CRC	Number of data excluding the CRC(1)	N/A	
Setting of TSIZE counter	N/A		Number of data	
Handling the end of a not aligned packed transfer	N/A	By software (via LDMA_RX bit)	Automatically by hardware	
CRC size	Fixed CRC8 for 8-bit data and CRC16 for 16-bit data	CRC8 or CRC16 can be applied for either 8- or 16-bit data	4-16/32 bit(2)	
Flushing of the CRC pattern from RxFIFO	Exclusively by software read access of RxFIFO	By software read access of RxFIFO(1)	Automatically by hardware	
Initialization CRC pattern	Fixed to all 0s		Selectable all 0s or all 1s	
Reset CRC calculation	Both SPI and CRC disable		By SPI disable	

1. *In Full duplex mode, DMA Rx counter can be set to include the number of the CRC patterns (1 or 2), as DMA Tx counter setting prevails to handle the pattern CRC transfer. The pattern is read out from the RxFIFO by DMA. This method cannot be used in Rx-only mode, where the CRC pattern flushing is handled exclusively by software.*
2. *The size of the CRC pattern is defined independently from the data size, but it must be either equal to the data size, or an integer multiple of it. It cannot exceed the maximum data size of the instance. The CRC polynomial size should be greater than the CRC pattern size to benefit from the CRC calculator capability, otherwise the pattern is upended by some invalid bits. To achieve full polynomial size coverage, specific additional CRC33_17 bit must be set when maximum data size is configured for 32- or 16-bit instances. TSIZE can never be set to its maximum value when CRC is enabled.*

4.2.4 Handling the Half-duplex mode

The most problematic SPI task is handling the Half-duplex mode, which requires common single-data line shared between master and slave alternately for bidirectional transfers. This mode cumulates all the problematics listed previously:

- **Handling exact number of data at particular simplex sessions**
  Terminating of master-receiver working at Receive-only mode is critical when the ongoing continuous flow must be stopped correctly. When handling CRC pattern upend or when the session is not aligned with the configured data threshold both data-flow directions require a specific control of the correct number of transferred data at the end of the flow.
- **Checking the end of the bus activity of a particular session**
  The bus has to be reconfigured between sessions in order to change direction of the data line, initialize CRC or configure the next session parameters. To do the configuration, SPI has to be temporary disabled. This disable might corrupt the end of the ongoing session, this is why the end of the bus activity has to be properly detected before disabling the SPI.
- **Reconfiguration of the bus between sessions**
  A change of the data-line direction must be synchronized on master and on slave sides, but these reconfigurations are independent, hence not fully synchronized. The reconfiguration timing is affected by how the involved nodes recognize the end of the bus activity and by the latencies of their SPI reconfiguration services. One node can initialize its GPIO output at common data-line while the opposite node is not yet able to manage the reconfiguration. Then it can happen that both GPIOs outputs propagate different output levels and compete for the line at the same time for a sure period (it depends on data being transferred and on the line clock phase setting). It is highly recommended to implement a serial resistor at this common data-line between the nodes. This action prevents a temporal short connection between the outputs and limits current blowing between them when just opposite levels are matching on the line.
- **Control of associated GPIOs to keep nodes synchronized**
  The necessity of the peripheral reconfiguration within the session requires an SPI disable. It brings additional effects like flushing the FIFO content or loosing control of the associated GPIOs. Continuity of SPI signals during reconfiguration of the bus must be ensured to prevent any synchronization issue between the involved nodes. Master has to avoid any glitches on SCK and SS signal. These signals might have to be managed by software as a general purpose outputs during the interface reconfiguration phase. An interactive session often starts by a command (write) sequence sent by master followed by an adequate response (writer or read) sequence when slave receives or sends data matched with the command content (except for cases when the protocol between master and slave is fixed). The master must always give sufficient time to the slave to reconfigure the bus, to recognize the command and to prepare the adequate answer. It mostly requires to avoid a continuous SCK clock signal and insert an adequate pause between such a command and data phases.

The Half-duplex mode handling is very complex. An analysis must be done to determine if the saved single data-line and the associated GPIO pin output are worth the effort, as Full-duplex communication mode is easier to handle. In Full-duplex communication mode, there is no need of any reconfiguration of the peripheral and the user can focus on handling the data content at the required direction while the data at opposite direction (not monitored) is either ignored or placed dummy.

4.3 Handling specific SPI signals

This section presents more details on handling specific internal-signals, interfaces and handling optional signals of the SPI interface

4.3.1 Handling specific internal signals and interfaces

In older versions, the DMA overtakes partial control of SPI functionalities due to the lack of transfer counters, therefore there is a wide interface between DMA and SPI. The most recent versions fully overtake the control while the simplified DMA interface handles only the data transfers. The number of data involved at SPI side must be correctly set and well balanced on both the SPI and DMA side, especially when enhanced SPI modes are used. Note that all the DMA access is based on the TXP and RXP or EOT data threshold events. If an event is missing (for example when incomplete data packet is done), no DMA transfer is performed.

On recent SPI versions, the Simplex SPI modes are strictly separated. There is no longer need to manage the unused RxFIFO flushing, nor to handle its OVR errors. DMA channel enable must be prevented in the unused direction.

When SPI is temporarily disabled, the associated GPIOs configured in alternate function output-mode at master side must remain at defined inactive levels. This need creates a problem on old SPI versions, as the peripheral leaves them undefined and floating.

Once the SPI is enabled, it may propagate active internal signal levels (such as empty transmit buffer status). This propagation may force a premature DMA stream request or create a series of initial interrupt services (if enabled in the configuration). Even a system without any FIFO could raise two consequent data requests when SPI is enabled and data is released from the buffer once the first bit is transferred out. The buffer raises the next data write request immediately after the transfer starts (refer to Figure 2).

The SPI enable sequence described in the product reference manual must be followed to ensure a proper events processing and to avoid missing any event service or a dysfunctional initialization of DMA streams.

Some signals can be evaluated even with significant latency if dual-clock domain is supported. For example, data underrun at slave side is evaluated under peripheral SCK clock domain, in consequence some traffic on the bus is necessary to evaluate the status (few additional SCK clock signal cycles are needed). This additional traffic makes the event to appear considerably later. Undesirable effects like unexpected data insertion may then occur once the underrrun flag is detected or cleared.

There is an internal synchronization between the APB bus clock and the kernel clock. If there is a big difference between them, some system responses may be processed with significant delay, causing unavailability of some features (refer to Section 2.2: Frequency constraints).

Most of the SPI instances can wake up the system from low-power modes. If dual-clock domain is available, SPI can autonomously communicate at low-power mode while all the necessary internal clock requests are automatically provided at dependency on applied SPI configuration and actual FIFO status.

Recent SPI versions can autonomously initialize transfers by system triggers. This action enables a data transfer synchronization with other peripheral events without system wake-up. The functionality and wake-up capabilities are product-dependent, refer to the reference manual for more details.

4.3.2 Handling optional signals

Slave select (NSS/SS) is an optional SPI interface signal. This signal is useful at multi slave or multi master topologies especially when it assigns just a single slave node for communication at time. This signal is useful even at single master/slave systems as it permits to separate and synchronize transfers between both nodes. Specific modes lake Slave Select pulse or TI one synchronize even each data frame.

Hardware support of the NSS master output at standard SPI Motorola mode is not ideal on old SPI versions. The signal logic copies the SPE status and causes problems to handle its GPIO alternate function. When the SPI is disabled, the signal is no longer propagated onto the associated GPIOs causing its floating once the SPE bit is set to zero.

On recent SPI versions, control over the alternate function can be kept even if the SPI is disabled. Also it is possible to select the polarity of the pin and to swap most of the MOSI and MISO signals capabilities. Recent SPI versions only support single NSS/SS pin. This alternate function has to be replaced by standard GPIO(s) control when handling a multi slave system or when a specific communication requires a reconfiguration of SPI within a single session (such at Half-duplex mode when the SPI has to be disabled temporarily while SS needs to stay active).

Recent SPI versions are edge selective in SS input hardware management mode (SSM = 0). A session can be missed when SPI is enabled after the master has provided the active edge of the SS signal. Older designs and SS software management mode (SSM = 1) are always level sensitive.

An optional RDY signal is featured in some SPI versions, which performs the slave's FIFOs occupancy status, and so its capability to continue at operation without any risk of data underrun or overrun. If the master disregards RDY and continues the transfer, the slave risks an underrun or overrun condition. On STM32 devices, the master communication is temporarily frozen when the slave provides a not-ready status.

4.4 SPI instances with different features

On STM32 devices the same peripherals can be implemented with different configurations. The main differences are:

- Maximum supported data size and its configurability (by bit or by multiple of bytes)
- Maximum supported polynomial or frame CRC size
- CRC configurability (by bit or by multiple of bytes)
- FIFOs capacity
- I2S support availability
- Maximum number of data in single session

- Low-power mode operation capabilities

The differences between the implemented instances are detailed in the SPI implementation table of reference manuals.

4.5 SPI synchronization between nodes

SPI is a synchronous interface and it demands proper synchronization between the nodes to ensure correct data handling. The only synchronization signals featured by SPI are NSS/SS, hence the user must check all desynchronization symptoms.

Possible causes of desynchronization are a glitch on SCK line due to signal disturbance, or a wrong handling of SCK alternate output during SPI reconfiguration.

Despite BSY flag can exceptionally hang on, monitoring its clearing can be helpful when communication becomes idle. If it stays set, it might signal possible synchronization issues between master and slave.

Correct data shifting by one or more bits and CRC errors are consequence of synchronization problems. In these cases, the bus must be inspected to find the cause of the abnormality, and to analyze the transferred data to find the root cause. If a desynchronization occurs, both the master and the slave nodes must be restarted.

Recent products can support half SCK clock period delay of the MISO master input sampling. This can help to compensate signal response delay raised, for example, when SPI bus signals are opto-isolated (refer to DRDS bit availability in configuration registers).

Apply SPI disable between sessions. This resets the internal state machine, and corrects desynchronization problems, if present.

Revision history

Table 3. **Document revision history**

Date	Version	Changes
30-Nov-2020	1	Initial release.
29-Jun-2021	2	Updated Table 1. Main SPI features on STM32 devices
12-Apr-2024	3	Updated document title, Introduction, Section 4: Data flow potential problems, and Section 4.5: SPI synchronization between nodes. Updated Table 1. Main SPI features on STM32 devices, and Table 2. Differences of CRC control between active SPI versions. Minor text edits across the whole document.
03-Oct-2024	4	Updated Table 1. Main SPI features on STM32 devices. Minor text edits across the whole document.

Contents

List of tables

List of figures