
Quad-SPI (QSPI) interface on STM32 microcontrollers

Introduction

In order to manage a wide range of multimedia, richer graphics and other data-intensive content, embedded applications evolve to offer more sophisticated features. These sophisticated features require extra demands on the often limited MCU on-chip memory.

External parallel memories are used to extend the MCU on-chip memory and solve the memory size limitation. Usually this action compromises the pin count and implies a need of more complex designs.

To face these requirements, STM32 MCUs (listed in [Table 1](#)) embed an external memory interface named Quad-SPI (see more details on [Table 2 on page 8](#)). This interface allows to connect external compact-footprint QSPI high-speed memories. This Quad-SPI interface can be used for data storage such as images, icons, or code execution.

This application note describes the Quad-SPI interface on the STM32 microcontrollers and explains how to use the module to configure, program, and read external Quad-SPI memories. It describes some typical use cases to use Quad-SPI interface based on some software examples from the STM32Cube firmware package and from the STM32F7 application notes.

Related documents

Available from STMicroelectronics web site www.st.com:

- *STM32L4x6 advanced ARM[®]-based 32-bit MCUs* (RM0351)
- *STM32F75xxx and STM32F74xxx advanced ARM[®]-based 32-bit MCUs* (RM0385)
- *STM32F446xx advanced ARM[®]-based 32-bit MCUs* (RM0390)
- *STM32F496xx and STM32F479xx advanced ARM[®]-based 32-bit MCUs* (RM0386)
- STM32L4x6, STM32F7x5/STM32F7x6, STM32F469/STM32F479 and STM32F446 datasheets.

Table 1. Applicable products

Type	Product series and lines
Microcontrollers	STM32L4 Series, STM32F7 Series
	STM32F446 line, STM32F469/479 line

Contents

1	Overview	8
1.1	Quad-SPI availability and features across STM32 families	8
1.2	Quad-SPI benefits against classic SPI and parallel interfaces	8
1.2.1	Main benefits of STM32 embedded Quad-SPI interface	9
1.3	Quad-SPI in a smart architecture	10
1.3.1	System architecture: STM32L4x6	10
1.3.2	System architecture: STM32F446	10
1.3.3	System architecture: STM32F469 and STM32F479	11
1.3.4	System architecture: STM32F7x5 and STM32F7x6	12
2	Quad-SPI interface description	13
2.1	Flexible frame format	13
2.1.1	Instruction phase	13
2.1.2	Address phase	15
2.1.3	Alternate-byte phase	15
2.1.4	Dummy-cycle phase	17
2.1.5	Data phase	17
2.2	Multiple hardware-configurations	19
2.2.1	Single-SPI mode (classic SPI)	19
2.2.2	Dual-SPI mode	20
2.2.3	Quad-SPI mode	21
2.2.4	Dual-Flash memory mode	21
2.2.5	DDR and SDR mode	25
2.3	Three operating modes	25
2.3.1	Indirect mode	25
2.3.2	Status-flag polling mode	26
2.3.3	Memory-mapped mode	26
2.4	Special features	28
2.4.1	Send instruction only-once (SIOO)	28
2.4.2	Delayed data sampling	28
2.4.3	Timeout counter	29
2.4.4	Additional status bits	29
2.4.5	Busy bit and abort functionality	29
2.4.6	4-byte address mode	30

2.5	Interrupts and DMA usage	32
2.5.1	Interrupts usage	32
2.5.2	DMA usage	32
2.6	Low-power modes	33
2.6.1	STM32L4x6 low-power mode	34
2.6.2	STM32F446, STM32F469, STM32F479, STM32F7x5 and STM32F7x6 low-power mode	34
3	Quad-SPI configuration	35
3.1	GPIOs configuration	35
3.1.1	GPIOs configuration using STM32CubeMX tool	35
3.2	Quad-SPI peripheral configuration and clock	38
3.2.1	Quad-SPI peripheral configuration (QUADSPI_CR register)	38
3.2.2	QSPI Flash memory parameters configuration (QUADSPI_DCR register)	38
3.2.3	QSPI memory device configuration	40
3.2.4	Starting a communication (QUADSPI_CCR register)	40
3.3	Hardware considerations	41
3.3.1	Pull-up resistance	41
3.3.2	Good PCB design allows maximum Quad-SPI speed	41
3.3.3	Chip-select high time (CSHT)	42
3.3.4	CKMODE	42
3.3.5	Some considerations when using Quad-SPI in classical SPI mode	42
4	Programming QSPI Flash memory	44
4.1	Programming code or data for an end application	44
4.1.1	Programming QSPI Flash memory using the STM32 ST-LINK utility	45
4.1.2	Programming QSPI Flash memory using IDE	49
4.2	Storing and erasing data on the fly during running application	53
4.2.1	Storing data	53
4.2.2	Erasing data	54
5	QSPI application examples	56
5.1	Reading data from QSPI memory: graphical application	56
5.1.1	Frame buffer content generation from QSPI memory	56
5.1.2	Displaying images directly from the QSPI memory	59
5.2	Executing from external QSPI memory: extend internal memory size	61

5.2.1	Configuring QSPI in memory-mapped mode during system initialization	63
5.2.2	Placing application's code in external QSPI memory	67
5.3	Storing (programming) data on the fly during a running application	72
5.3.1	Quad-SPI indirect write: programming QSPI memory using DMA	72
5.3.2	Quad-SPI indirect write: programming QSPI memory using interrupts	75
5.4	Erasing-data example	76
5.5	Hardware implementation example	77
6	Performance and power	79
6.1	How to get the best performances	79
6.1.1	Write performance	79
6.1.2	Read performance	79
6.2	Decreasing power consumption	82
6.2.1	Use timeout counter	82
6.2.2	Put the QSPI memory in deep power-down mode	82
6.2.3	Quad-SPI Flash memories supporting DPD mode	83
7	Supported devices	84
8	Conclusion	85
9	Revision history	86

List of tables

Table 1.	Applicable products	1
Table 2.	Quad-SPI availability and features across STM32 families	8
Table 3.	Benefits of using STM32 Quad-SPI interface	9
Table 4.	Instruction phase configurations	14
Table 5.	Address-phase configurations	15
Table 6.	Alternate-byte phase configurations	16
Table 7.	Data phase configuration versus Quad-SPI functional modes	18
Table 8.	Hardware configurations versus used GPIO number	19
Table 9.	Dual-Flash memory hardware configurations	23
Table 10.	Additional status bits	29
Table 11.	BUSY bit reset in different QSPI modes	30
Table 12.	Address mode versus maximum addressable memory space	31
Table 13.	Quad-SPI interrupts summary	32
Table 14.	DMA requests mapping and transfer directions versus STM32 series	33
Table 15.	QSPI peripheral in different power modes on STM32L4	34
Table 16.	QSPI peripheral in different power modes on STM32F446, STM32F469, STM32F479, STM32F7x5 and STM32F7x6	34
Table 17.	Execution performances versus configuration	71
Table 18.	Different STM32 boards embedding QSPI Flash memory	77
Table 19.	Document revision history	86

List of figures

Figure 1.	System architecture: STM32L4x6	10
Figure 2.	System architecture: STM32F446	11
Figure 3.	System architecture: STM32F469/F479	11
Figure 4.	System architecture: STM32F7x5 and STM32F7x	12
Figure 5.	Reading sequence in quad I/O SDR	13
Figure 6.	Alternate-byte phase: sending a nibble in dual-SPI mode	16
Figure 7.	Dummy-cycle: IO2 maintained low and IO3 maintained high by hardware	17
Figure 8.	Hardware configuration: single-SPI mode	20
Figure 9.	Hardware configuration: dual-SPI mode	20
Figure 10.	Hardware configuration: Quad-SPI mode	21
Figure 11.	Read sequence in dual-Flash memory Quad I/O SDR mode	22
Figure 12.	Executing non-sequential code from Quad-SPI	27
Figure 13.	Executing non-sequential code from Quad-SPI with SIOO enabled	28
Figure 14.	STM32CubeMX: Quad-SPI GPIOs configuration	35
Figure 15.	STM32CubeMX: PF8 pin configuration to QUADSPI_BK1_IO0 alternate function	36
Figure 16.	STM32CubeMX: Dual-Flash memory Quad-SPI with chip-select 1 configuration	36
Figure 17.	STM32CubeMX: Quad-SPI configuration	37
Figure 18.	STM32CubeMX: enabling Quad-SPI global interrupt	37
Figure 19.	Quad-SPI clock configuration on QUADSPI_CR register	38
Figure 20.	STM32CubeMX: Quad-SPI peripheral configuration	39
Figure 21.	Write enable sequence (command 0x06)	40
Figure 22.	Connecting chip-select to a pull-up resistance	41
Figure 23.	Chip select high time: CSHT = two clock cycles	42
Figure 24.	Quad-SPI in classical SPI-mode frame example	43
Figure 25.	Programming QSPI memory through debug interface	45
Figure 26.	STM32 ST-LINK utility: adding QSPI Flash memory loader	46
Figure 27.	STM32 ST-LINK utility: selecting QSPI Flash memory loader	46
Figure 28.	STM32 ST-LINK utility: error message	47
Figure 29.	STM32 ST-LINK utility: programming QSPI Flash memory	47
Figure 30.	STM32 ST-LINK utility: selecting HEX file for programming	48
Figure 31.	STM32 ST-LINK utility: erasing sectors	48
Figure 32.	Adding QSPI Flash memory loader to Keil MDK-ARM project	50
Figure 33.	Adding QSPI Flash memory loader to Keil MDK-ARM project	50
Figure 34.	Selecting QSPI Flash memory programming algorithm	51
Figure 35.	QSPI Flash memory loader programming algorithm configuration	51
Figure 36.	Quad I/O page program sequence (command 0x38)	53
Figure 37.	Read status register sequence (command 0x05)	54
Figure 38.	Sector erase sequence	55
Figure 39.	Example: full chip-erase sequence	55
Figure 40.	Quad-SPI usage in a graphical application	57
Figure 41.	DMA2D reading images from Quad-SPI to build frame buffer content	59
Figure 42.	LTDC reading an image directly from QSPI memory	60
Figure 43.	Project configurations: executing code from QSPI Flash memory	62
Figure 44.	Changing Quad-SPI configuration in the project's settings	63
Figure 45.	QSPI Flash memory connection in STM32756-EVAL board	64
Figure 46.	6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in QSPI memory	68
Figure 47.	6_2-Quad-SPI_rwRAM-DTCM project configuration: only code in QSPI memory	69
Figure 48.	Indirect write mode: programming QSPI memory using DMA	73

Figure 49.	Indirect write mode: programming QSPI memory using interrupt.	75
Figure 50.	QSPI memory connection on the STM32F746G-DISCO discovery board	78
Figure 51.	QSPI memory connection on the STM32L476G-EVAL board	78
Figure 52.	Deep Power-down (DPD) Sequence (Command B9).	82
Figure 53.	Release from deep power-down (RDP) sequence (Command AB)	83

1 Overview

The Quad-SPI is a serial interface allowing communication on four lines between a host (STM32) and an external QSPI memory. The Quad-SPI supports the traditional SPI (serial peripheral interface) as well as the dual-SPI mode which allows to communicate on two lines. Quad-SPI uses up to six lines in quad mode: one line for chip select, one line for clock and four lines for data in and data out.

This interface is integrated on STM32 MCU to fit memory-hungry applications, to simplify PCB (printed circuit board) designs and to reduce costs.

1.1 Quad-SPI availability and features across STM32 families

All STM32 microcontrollers shown in [Table 2](#) have mainly the same Quad-SPI features, except for STM32L4 series that does not support dual-Flash memory.

Table 2. Quad-SPI availability and features across STM32 families

QUADSPI features	Maximum speed (MHz) ⁽¹⁾		Dual-Flash memory	Max addressable space ⁽²⁾	
	SDR	DDR		Memory mapped	Indirect mode
STM32L4x6	60	48	No	256Mbytes	4Gbytes
STM32F446 ⁽³⁾	90	60	Yes		
STM32F469/F479	90	80	Yes		
STM32F7x5/F7x6	108	80	Yes		

1. Maximum Quad-SPI speed from datasheet. For more details on the Quad-SPI maximum speed refer to the relevant device's datasheet.
2. 32-bits address mode should be used to reach 256Mbytes in memory mapped mode and 4Gbytes in indirect mode.
3. Only single-SPI and dual-SPI modes are supported for STM32446 LQFP64 package.

1.2 Quad-SPI benefits against classic SPI and parallel interfaces

The Quad-SPI brings more performance in terms of throughput compared to classical SPI. The classical SPI uses only one data line while the Quad-SPI uses four data lines which multiplies the data throughput by almost four times.

Compared to FMC (flexible memory interface) and other parallel interfaces, Quad-SPI allows to connect a lower cost external Flash memory to small packages, reducing PCB area, simplifying PCB design and reducing GPIOs (general-purpose input/output) usage. In Quad-SPI mode only six GPIOs are used: four lines for data plus one line for clock and another for chip select. In dual-Flash Quad-SPI mode only 10 GPIOs are used, where eight lines for data.

1.2.1 Main benefits of STM32 embedded Quad-SPI interface

[Table 3](#) summarizes the major advantages of using STM32 embedded Quad-SPI interface:

Table 3. Benefits of using STM32 Quad-SPI interface

Benefits	Comments
Low pin count	Supports single, dual and QSPI memories. Uses six pins in Quad-SPI and four pins for single or dual SPI. Saves GPIOs to be used for other purposes.
Easier PCB design	Allows easier and faster PCB design thanks to a reduced pin count.
Save space for smaller size applications	Can be used in small size applications due to small footprint QSPI memories.
Save cost	Easier and faster design allows a lower development cost. Lower PCB cost, as it is possible to reduce PCB layers due to low pin count. Low cost memory solution.
Executable	Extends limited on-chip Flash memory allowing QSPI memory to be seen as an internal memory. Allows code execution (XIP mode) from QSPI Flash memory. Supports SIOO mode also named Continuous Read Mode by some memory manufacturers (see Section 2.4.1: Send instruction only-once (SIOO) on page 28) for higher execution performance.
Extended size for data storage	Memory-mapped mode allows QSPI memory to be accessed autonomously by any AHB (advanced high-performance) master. 32-bits address mode allows to address up to four Gbytes QSPI memory size. Dual-Flash memory mode allows using two QSPI Flash memories to double storage size ⁽¹⁾ .
High performances	Throughput is multiplied by four versus traditional SPI. The DDR mode doubles throughput. The dual-Flash memory mode doubles throughput. Perfect for graphical applications.
Multiple memory solutions	There are volatile QSPI SRAM (static random-access memory) available from Microchip, ON Semiconductor and others. Available non-volatile QSPI Flash (ROM) memories. NOR, NAND.
Supports any Quad-SPI memories available in the market	Its fully configurable and flexible frame format allows to support almost all QSPI devices available on the market.
Growing amount of manufacturers	Spansion, Windbond, Micron, Macronix, ONSemiconductors, Cypress, among others. Huge investment on higher densities QSPI Flash memories such as NAND.

1. 4 Gbytes maximal size can be reached with the 32-bits address mode.

1.3 Quad-SPI in a smart architecture

The Quad-SPI is mapped on a dedicated layer on AHB allowing it to be accessible as an internal memory thanks to the memory-mapped mode. In addition, the Quad-SPI interface is integrated in a smart architecture allowing:

- All masters to access the external QSPI memory without CPU intervention.
- All masters can read data from QSPI memory even in SLEEP mode when the CPU is stopped thanks to the STM32's smart architecture.
- CPU as a master can access Quad-SPI and execute code from the memory.
- GP DMA to do transfer from QSPI to other internal or external memories.
- Graphical DMA2D to directly build RAM video frames using QSPI Flash.

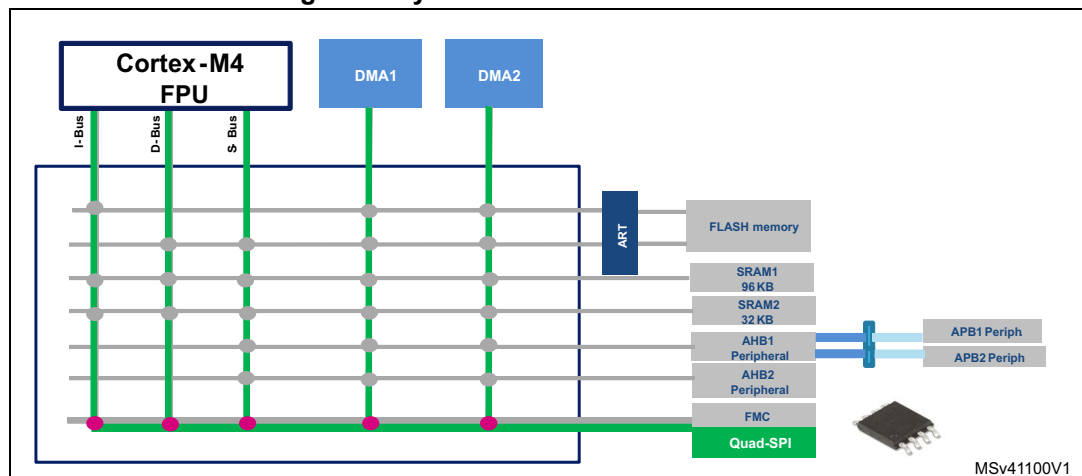
1.3.1 System architecture: STM32L4x6

The STM32L4x6's system architecture consists mainly of 32-bit multilayer AHB bus matrix that interconnects five masters and seven slaves.

The external QSPI memory can be accessed by the Cortex®-M4 either through the S-Bus or through I-bus and D-bus when physical remap is enabled, which boosts execution performances.

The Quad-SPI is also accessible by the masters on AHB bus matrix which are DMA1 and DMA2. [Figure 1](#) shows Quad-SPI interconnection in the STM32L4x6's system

Figure 1. System architecture: STM32L4x6

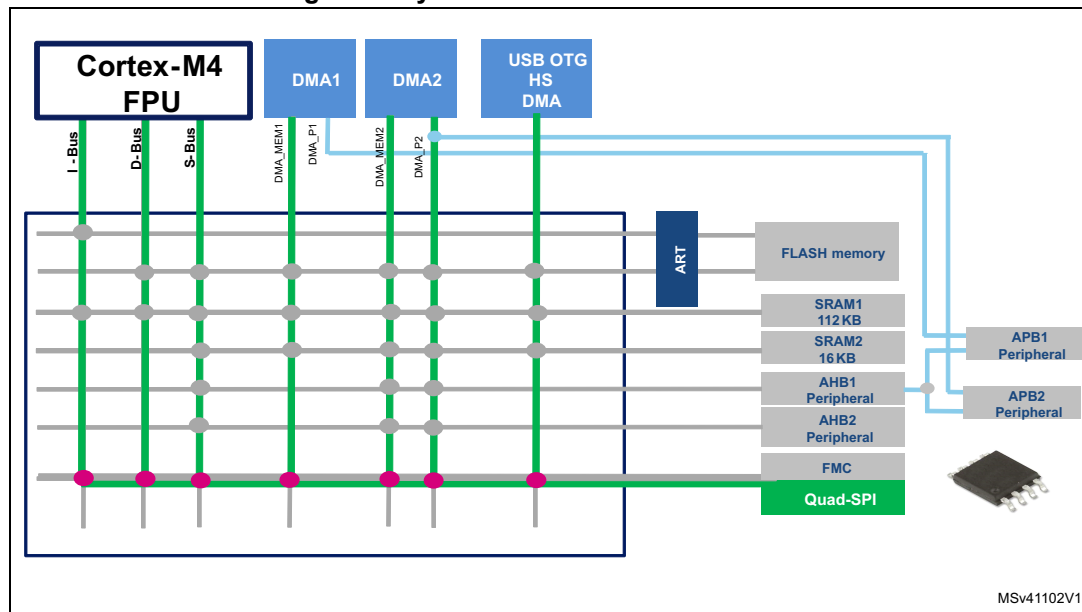


1.3.2 System architecture: STM32F446

The STM32F446's system architecture consists mainly of 32-bit multilayer AHB bus matrix that interconnects seven masters and seven slaves.

The external QSPI memory can be accessed by the Cortex®-M4 through the S-Bus. The Quad-SPI is also accessible by all the masters on AHB bus matrix, such as DMA1, DMA2 or USB OTG HS DMA. [Figure 2](#) shows the Quad-SPI's interconnection in the STM32F446's system.

Figure 2. System architecture: STM32F446



1.3.3 System architecture: STM32F469 and STM32F479

The STM32F469 and STM32F479's system architecture consists mainly of 32-bit multilayer AHB bus matrix that interconnects ten masters and nine slaves. The Quad-SPI is mapped on a dedicated layer on the AHB busmatrix.

The external QSPI memory can be accessed by the Cortex[®]-M4 through the S-Bus.

The Quad-SPI is also accessible by all the masters on AHB bus matrix, such as DMA1, DMA2, Chrom-ART accelerator or LCD-TFT, enabling an efficient data transfer like images for graphical applications. [Figure 3](#) shows the Quad-SPI interconnection in the STM32F469 and STM32F479 system.

Figure 3. System architecture: STM32F469/F479



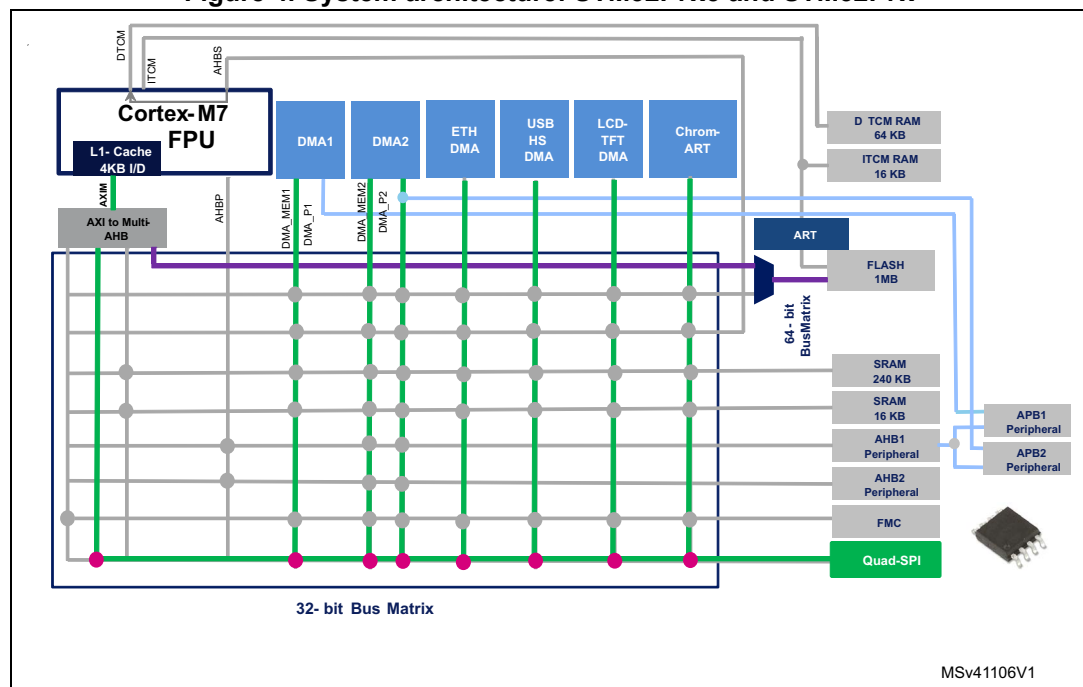
1.3.4 System architecture: STM32F7x5 and STM32F7x6

The main system architecture is based on two subsystems, an AXI (advanced extensible interface) to multi AHB bridge converting AXI4 protocol to AHB-Lite protocol and a multi-AHB bus matrix.

The multi AHB bus matrix interconnects twelve masters and eight slaves. There are four AXI bus accesses where the Quad-SPI is accessible through the second one, allowing the Cortex[®]-M7 to fetch code or data. The Quad-SPI is mapped on a dedicated layer on the AHB Busmatrix and can benefit from L1-Cache allowing 0-wait states access to the Cortex[®]-M7.

Quad-SPI is also accessible by all the masters on AHB bus matrix, such as DMA1, DMA2, Chrom-ART accelerator or LCD-TFT, enabling an efficient data-transfer in graphical applications. [Figure 4](#) shows Quad-SPI interconnection in the STM32F7x5 and STM32F7x6's system.

Figure 4. System architecture: STM32F7x5 and STM32F7x



2 Quad-SPI interface description

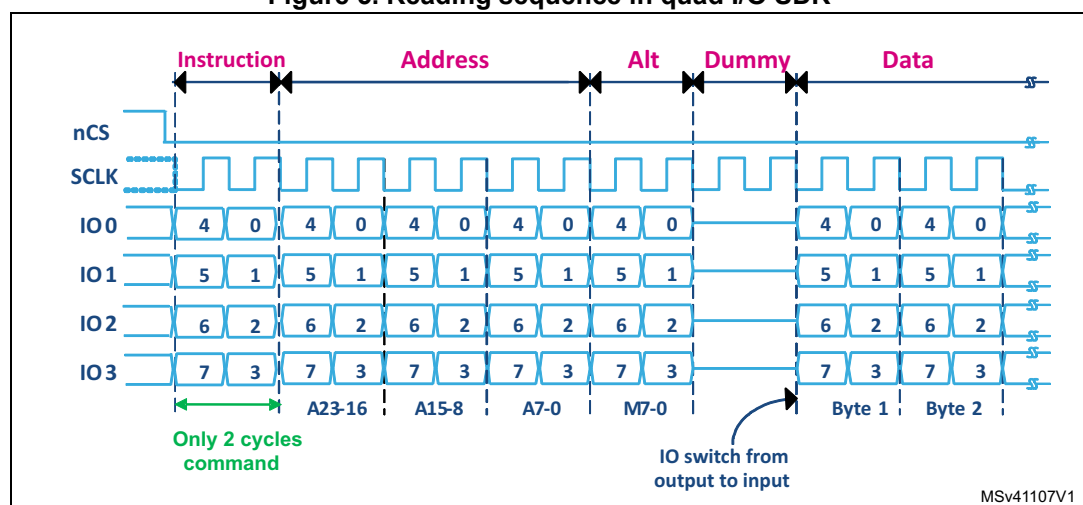
2.1 Flexible frame format

The QSPI interface provides a fully programmable frame composed of five phases where each phase is fully configurable, allowing it to be configured separately in terms of length and number of lines.

Note: *Instruction, address, alternate and data phases can only be configured in four lines if the hardware is configured in Quad-SPI mode.*

The frame format can be configured only in indirect mode or memory-mapped mode but not in status-flag polling mode. [Figure 5](#) reads the sequence in Quad I/O SDR mode.

Figure 5. Reading sequence in quad I/O SDR



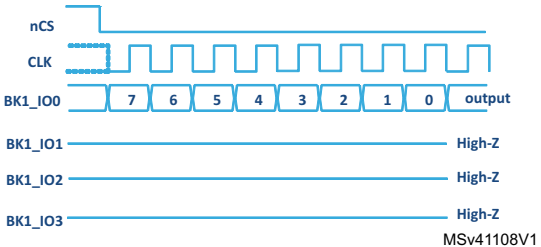
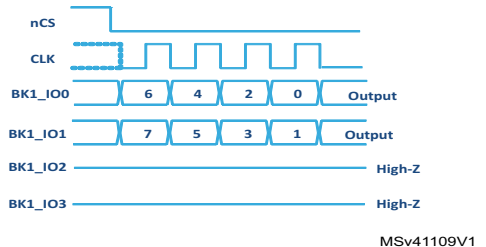
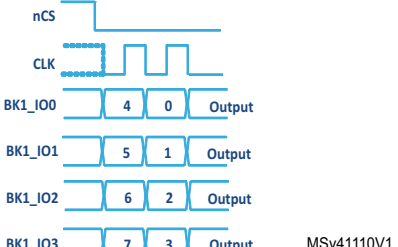
2.1.1 Instruction phase

In this phase a command (8-bits instruction) is sent to the Flash memory, specifying the type of operation to be performed. This command is fully configurable allowing to send any value, the user can simply write the desired command to be sent in the INSTRUCTION field of the QUADSPI_CCR[7:0] register.

Depending on the software and the hardware configurations, the instruction can be sent in one, two or four lines. In some use cases where only the address is sent, the instruction phase can be skipped. [Table 4](#) summarizes the different configurations for instruction phase.

Note: *The DDR mode is not supported in this phase, so even if the DDR mode is enabled the command is always sent in SDR mode.*

Table 4. Instruction phase configurations

Register configurations		Indirect mode Automatic-polling mode Memory-mapped mode	Command formats
Instruction to be sent in QUADSPI_CCR[7:0]		INSTRUCTION [7: 0]	NA
Instruction phase QUADSPI_CCR[9:8]	No Instruction: skipped	IMODE[1:0] = 00	The instruction phase is skipped
	Instruction on 1 line: Single SPI mode	IMODE[1:0] = 01	
	Instruction on 2 lines: Dual SPI mode	IMODE[1:0] = 10	
	Instruction on 4 lines: Quad-SPI mode	IMODE[1:0] = 11	

2.1.2 Address phase

In this phase an address is sent to the Flash memory, specifying the address of the data to be read or written. The address phase is fully configurable allowing to send one, two, three or four bytes address. In indirect mode and automatic-polling mode, the user can simply write the desired address in the QUADSPI_AR register.

Depending on the software and the hardware configurations, the address can be sent in one, two or four lines. In some use cases where the address is not needed such as in mass-erase operation, the address phase can be skipped

[Table 5](#) summarizes different address phase configurations.

Table 5. Address-phase configurations

Register configurations		Indirect mode	Automatic-polling mode	Memory-mapped mode
Address to be sent QUADSPI_AR[31:0]		ADDRESS[31:0]		Address is given directly via the AHB from any master on the bus matrix like Cortex® or DMA
Address size QUADSPI_CCR[13:12]	1-byte	ADSIZE[1:0] = 00		
	2-byte	ADSIZE[1:0] = 01		
	3-byte	ADSIZE[1:0] = 10		
	4-byte	ADSIZE[1:0] = 11		
Address phase QUADSPI_CCR[11:10]	No address: skipped	ADMODE[1:0] = 00		
	Address on 1 line: Single SPI mode	ADMODE[1:0] = 01		
	Address on 2 lines: Dual SPI mode	ADMODE[1:0] = 10		
	Address on 4 lines: Quad SPI mode	ADMODE[1:0] = 11		

Note: In dual-Flash memory mode when DFM = 1 the address to be sent to Flash1 is exactly the same address to be sent to Flash2.

2.1.3 Alternate-byte phase

This is an extra phase supported by the Quad-SPI interface offering more flexibility. It is generally used for controlling the mode of operation; for instance 1-byte can be sent continuously to keep the Quad-SPI device in an operating mode. This is supported for some memory manufacturers such as Spansion or Micron where an alternate byte is sent continuously to keep the memory in execute-in-place mode.

The alternate-byte phase is fully configurable allowing to send one, two, three or four bytes depending on the ABRSIZE [1:0] file configuration. The user can simply write the desired alternate bytes in the QUADSPI_ABR register.

Depending on the software and the hardware configurations, the alternate byte can be sent in one, two or four lines. If not needed, the alternate-byte phase can be skipped. [Table 6](#) summarizes different alternate-byte phase configurations.

Table 6. Alternate-byte phase configurations

Register configuration		Indirect mode	Automatic-polling mode	Memory-mapped mode
Alternate-byte to be sent QUADSPI_ABR		QUADSPI_ABR		
Alternate-byte size QUADSPI_CCR[17:16]	1-byte	ABSIZE [1:0] = 00		
	2-byte	ABSIZE [1:0] = 01		
	3-byte	ABSIZE [1:0] = 10		
	4-byte	ABSIZE [1:0] = 11		
Alternate-byte phase QUADSPI_CCR[15:14]	No alternate-byte: skipped	ABMODE [1:0] = 00		
	Alternate-byte on 1 line: single SPI mode	ABMODE [1:0] = 01		
	Alternate-byte on 2 lines: dual SPI mode	ABMODE [1:0] = 10		
	Alternate-byte on 4 lines: Quad SPI mode	ABMODE [1:0] = 11		

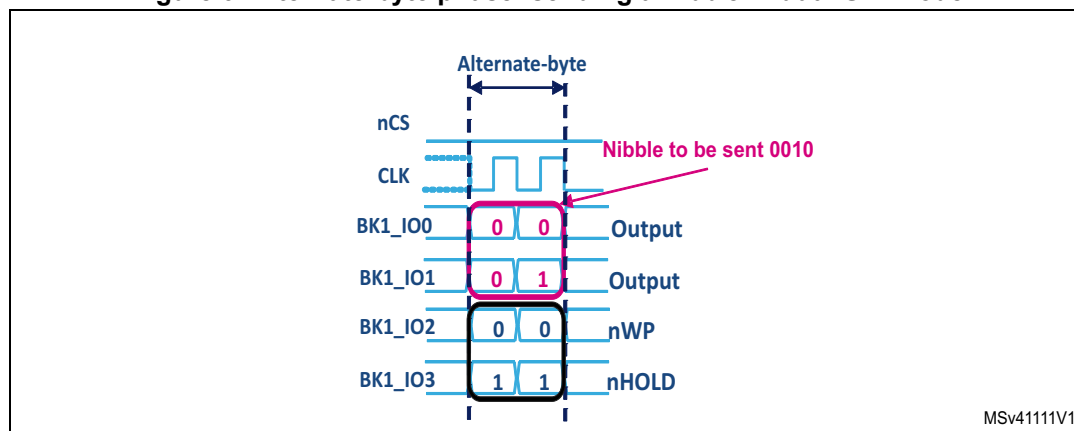
Note: In dual-Flash memory mode when DFM = 1, the alternate-byte to be sent to Flash1 is exactly the same as the ones to be sent to Flash2.

Alternate-byte phase: sending a nibble in dual-SPI mode

In some cases only one nibble needs to be sent at alternate-byte phase during two clock cycles only rather than a full byte during four clock cycles. For instance, when the dual-SPI mode is used and only two cycles are used for the alternate-byte phase.

The Quad-SPI mode can be activated only for alternate-byte phase in order to send an alternate byte where the nibble is sent over IO0 and IO1 while the other nibble have to be sent only to keep IO2 low and IO3 high during alternate-byte phase.

Figure 6. Alternate-byte phase: sending a nibble in dual-SPI mode



MSv41111V1

2.1.4 Dummy-cycle phase

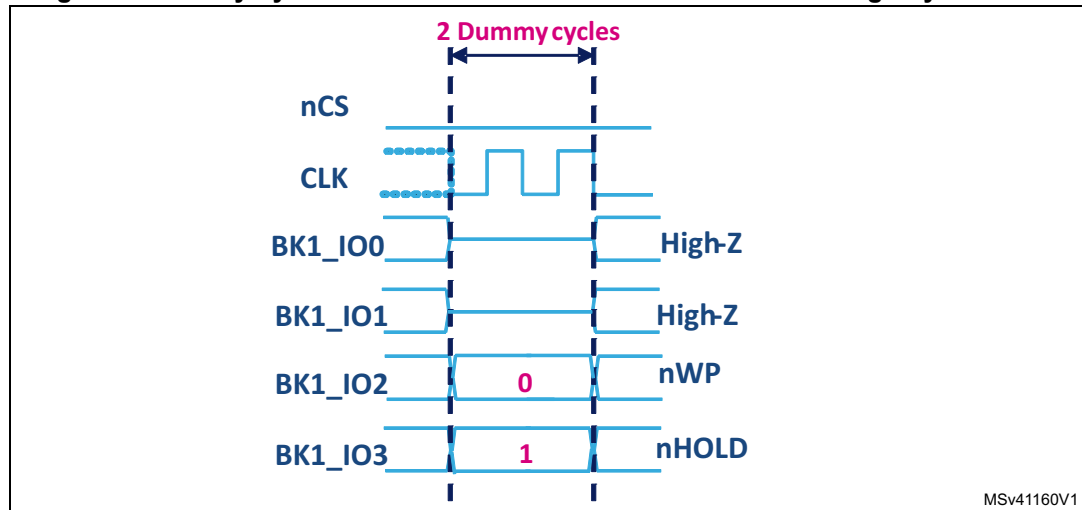
The dummy-cycle phase is needed in some cases when operating at high clock frequencies. This phase allows to ensure enough “turnaround” time for changing the data signals from output mode to input mode.

The dummy phase is enabled by setting the number of dummy cycles in the DCYC[4:0] field QUADSPI_CCR register. The number defined in DCYC[4:0] field can reach 31 cycles and it does not depend on the used hardware configuration.

Note: Either in SDR or DDR mode, one dummy represents always one Quad-SPI clock cycle.

During this phase, if the Quad-SPI hardware configuration is used and if communication phases are either in Quad-SPI or dual-SPI modes, IO2 is forced to ‘0’ to disable the “write protect” function while IO3 is forced to ‘1’ to disable the “hold” function of the QSPI memory. This is fully managed by hardware (Quad-SPI peripheral) so nothing needs to be configured by the user.

Figure 7. Dummy-cycle: IO2 maintained low and IO3 maintained high by hardware



2.1.5 Data phase

In this phase; the data is sent or received from or to the QSPI memory. The data phase is fully configurable allowing to send, receive or both any number of bytes to or from the QSPI memory device.

In indirect mode and in the automatic-polling mode, the number of bytes to be sent, received or both is specified in the QUADSPI_DLR register.

In the indirect-write mode the data to be sent to the Flash memory must be written to the QUADSPI_DR register, while in the indirect-read mode the data received from the Flash memory is obtained by reading from the QUADSPI_DR register.

In the memory-mapped mode the data can only be read from the memory device but not written, then the data is accessed directly from the Quad-SPI FIFO. All masters on the bus matrix can read data from QSPI memory device as it is an internal memory.

Depending on the software and the hardware configurations, the data transfer can be in one, two or four lines. In some use cases where data is not needed such as erasing operation, the data phase can be skipped. [Table 7](#) summarizes the data phase configuration in different functional modes.

Table 7. Data phase configuration versus Quad-SPI functional modes

Register configuration		Indirect mode	Automatic-polling mode	Memory-mapped mode
Data	Read data	QUADSPI_DR		Data read is sent back directly over the AHB to any master on the bus matrix requesting for reading operation (Cortex [®] , DMA LTDC, DMA2D...).
	Write data	QUADSPI_DR		Not supported
Number of data to be sent/received	QUADSPI_DLR	1-byte 0x00000000 to undefined ⁽¹⁾ 0xFFFFFFFF	1-byte 0x00000000 to 4-bytes 0x00000003	QUADSPI_DLR has no meaning in this mode. If DMA is used, number of data to be read is set only in DMA's DMA_SxNDTR register.
Data phase QUADSPI_CCR[15:14]	No Data: skipped	DMODE[1:0] = 00 ⁽²⁾		
	Data on 1 line: Single SPI mode	DMODE[1:0] = 01		
	Data on 2 lines: Dual SPI mode	DMODE[1:0] = 10		
	Data on 4 lines: Quad SPI mode	DMODE[1:0] = 11		

1. When QUADSPI_DLR = 0xFFFFFFFF, the number of bytes to be sent or received is undefined so the transfer continues until the end of memory as defined in FSIZE. When QUADSPI_DLR = 0xFFFFFFFF and FSIZE = 0x1F then the transfer continue indefinitely, stopping only after an abort request or after the Quad-SPI is disabled. After the last memory address is read (at address 0xFFFFFFFF), the reading continues with address = 0x00000000.

2. This mode should be used only in the indirect mode.

2.2 Multiple hardware-configurations

The STM32 offers a very flexible Quad-SPI interface allowing to connect external the QSPI memory in different hardware configurations, allowing the user to choose its own configuration.

Depending on the used hardware configuration, the number of the used GPIOs can be up to 11. [Table 8](#) summarizes different use cases.

Table 8. Hardware configurations versus used GPIO number

		Single-Flash mode		Dual-Flash memory mode
Single/Dual SPI mode	Used GPIOs	Bank1 CLK BK1_IO0/SO BK1_IO1/SI BK1_nCS	Bank2 CLK BK2_IO0/SO BK2_IO1/SI BK2_nCS	CLK BK1_IO0/SO BK1_IO1/SI BK2_IO0/SO BK2_IO1/SI BK1_nCS ⁽¹⁾ BK2_nCS ⁽¹⁾
	GPIOs number	4 GPIOs		6 or 7 GPIOs
Quad-SPI mode	Used GPIOs	Bank1 CLK BK1_IO0/SO BK1_IO1/SI BK1_IO2 BK1_IO3 BK1_nCS	Bank2 CLK BK2_IO0/SO BK2_IO1/SI BK2_IO2 BK2_IO3 BK2_nCS	CLK BK1_IO0/SO BK1_IO1/SI BK1_IO2 BK1_IO3 BK2_IO0/SO BK2_IO1/SI BK2_IO2 BK2_IO3 BK1_nCS ⁽¹⁾ BK2_nCS ⁽¹⁾
	GPIOs number	6 GPIOs		10 or 11 GPIOs

1. In dual-Flash mode it is possible to use one chip select, either BK1_nCS or BK2_nCS. For more details on dual-flash mode, refer to [Section 2.2.4: Dual-Flash memory mode on page 21](#).

Note: *If none of the phases are configured to use Quad-SPI mode, then the GPIOs corresponding to IO2 and IO3 can be used for other functions even while the Quad-SPI is active.*

2.2.1 Single-SPI mode (classic SPI)

This is the classic SPI where only four GPIOs are used and the data is sent on SO line and received on SI line.

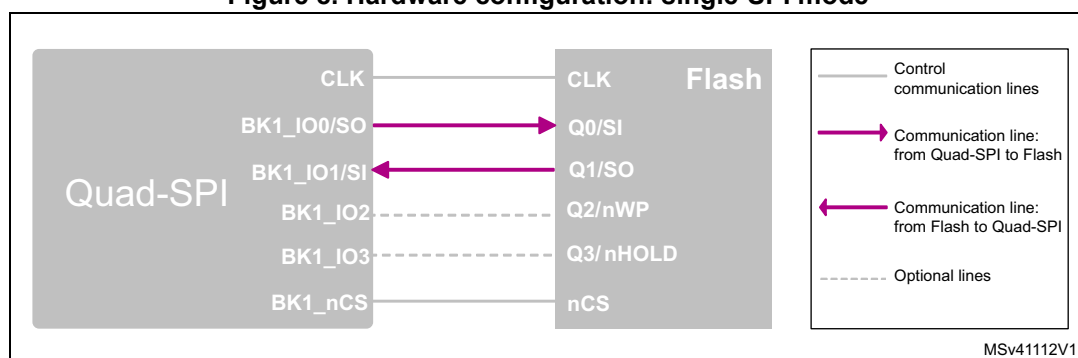
Note: *Full duplex transfer is not supported.*

The IO2 and IO3 lines are optional and can be used for other purposes such as:

- When used (IO2 and IO3 are connected to the QSPI memory): IO2 and IO3 pins should be configured as for IO0 and IO1. To allow communication with memory device:
 - IO2 is in output mode and forced to '0' to deactivate the "write protect" function
 - IO3 is in output mode and forced to '1' to deactivate the "hold" function
 - This is managed by the hardware (Quad-SPI peripheral) during all communication phases
- When not used, the nWP and nHOLD memory device pins have to be connected respectively to VDD and VSS while IO2 and IO3 pins could be used for other purposes.

In this mode, all phases as instruction, address, alternate-byte and data have to be configured in single-SPI mode by setting the IMODE/ADMODE/ABMODE/DMODE fields in QUADSPI_CCR to 01.

Figure 8. Hardware configuration: single-SPI mode

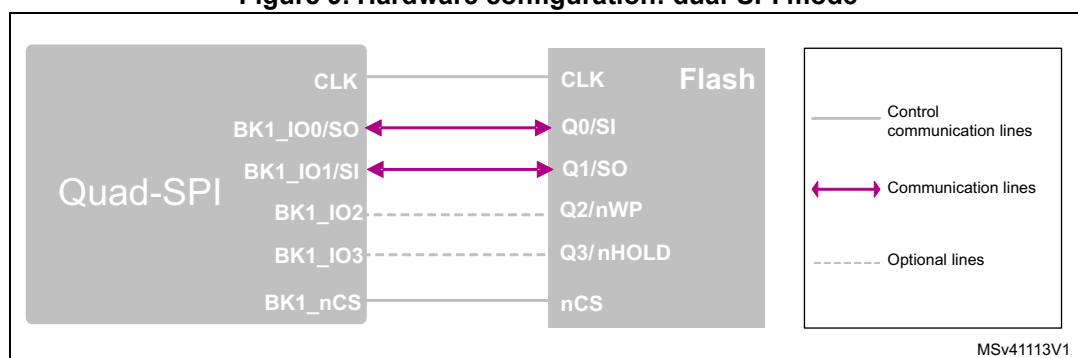


2.2.2 Dual-SPI mode

In dual-SPI mode the hardware configuration is similar to the one in single mode, but here two lines are used for data, it means that data is sent and received in two lines. As for single-SPI mode, the IO2 and IO3 lines are optional, if not used the nWP and nHOLD device pins have to be connected respectively to VDD and VSS.

In this mode all the instruction, address, alternate-byte and data phases have to be configured in dual-SPI mode by setting the IMODE/ADMODE/ABMODE/DMODE fields in QUADSPI_CCR to 10.

Figure 9. Hardware configuration: dual-SPI mode



2.2.3 Quad-SPI mode

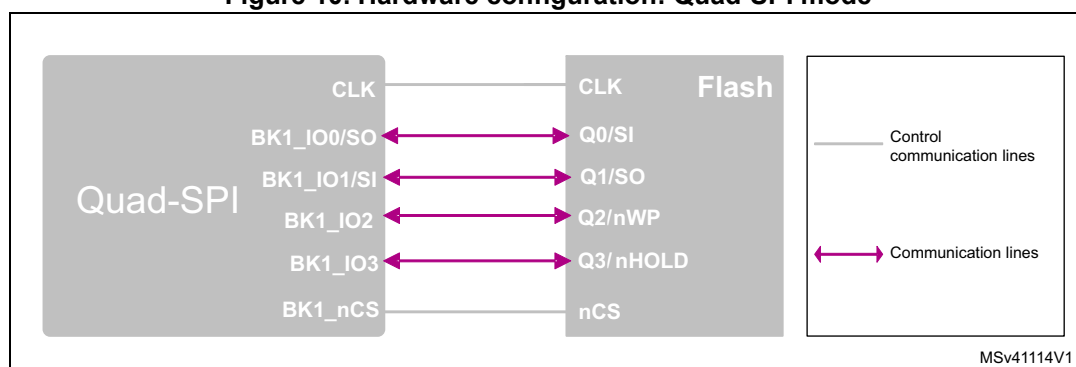
In the Quad-SPI mode six pins are used: four pins for data and two pins for clock and chip select. In this hardware configuration it is possible to use either single or dual-SPI mode. In Quad-SPI mode the data is transferred, received or both over four lines.

Note: In this mode, the hold and WP features are no longer available as IO3 and IO4 are used for communications.

In this mode, depending on the QSPI memory brand, the user can choose to send each phase in single, dual or quad mode. Many memory manufacturers are supporting the following configurations where Command-Address-Data: 1-1-4; 1-4-4; 4-4-4.

In general, if the address is sent in four lines then the alternate-byte should be sent in four lines too.

Figure 10. Hardware configuration: Quad-SPI mode



2.2.4 Dual-Flash memory mode

In dual-Flash memory mode, the MCU communicates with two external memory devices at the same time. This mode is useful to double throughput and to double size while using only 10 GPIOs: eight for data, one chip select for both devices and one for CLK. We can attain a throughput of two bytes per cycle with dual-Flash memory in DDR Quad-SPI mode.

In this mode, only one chip-select could be used for both devices and then save one GPIO for other usages and either nCS_BK1 or nCS_BK2 can be connected to both devices. The clock has to be connected to both devices.

Different hardware configurations are allowed, offering a high flexibility to the user. [Table 8 on page 19](#) illustrates all possible hardware configurations.

The dual-memory mode allows doubling the throughput, as one byte can be sent, received or both at every cycle. This mode is very interesting when more performance is needed; not only throughput is doubled in dual-Flash memory mode, but also the external memory size is doubled.

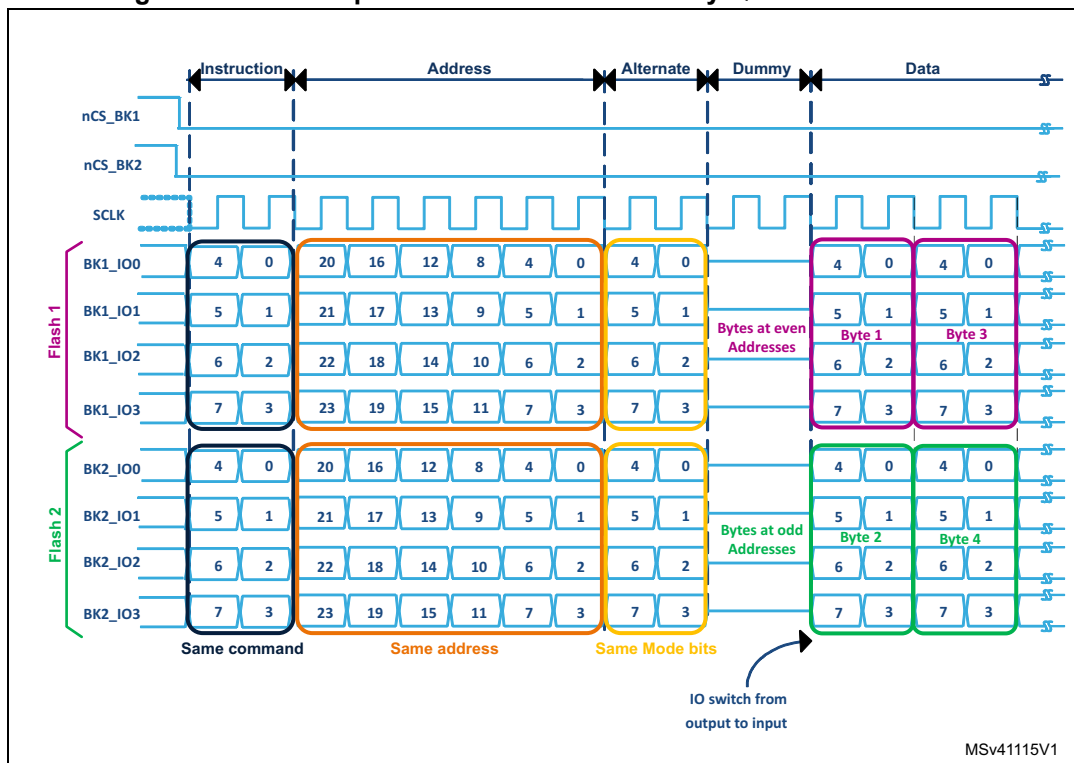
When using two external QSPI memories the size to be configured in FSIZE[4:0] should reflect the total Flash memory capacity, which is double the size of one individual component.

To support dual die packages with two chip-selects and dual QSPI devices, the FIFO size is always 32-bytes either in single Flash or dual-Flash memory mode.

Note: Either in single-Flash memory mode or dual-Flash memory mode, the addressable space in memory-mapped mode is always up to 256Mbytes. To go up to 256Mbytes, the 32-Bits address mode should be used.

Figure 11 shows an example of a read sequence in dual-Flash memory Quad I/O SDR mode.

Figure 11. Read sequence in dual-Flash memory Quad I/O SDR mode



Note that all bytes at even addresses are stored in Flash 1 while all bytes at odd addresses are stored in Flash 2. As described in Figure 11, in dual-Flash mode the same command, address and alternate are sent to both Flash 1 and Flash 2. For example to read the first four bytes in dual-Flash memory-mapped mode from 0x90000000 to 0x90000003 the following sequence is done by Quad-SPI peripheral:

- The address 0x00000000 is sent to both flashes and Byte 1 (at even address 0x90000000) is read from Flash 1 while Byte 2 (at odd address 0x90000001) is read from Flash 2.
- Then the address 0x00000001 is sent to both Flashes and Byte 3 (at even address 0x90000002) is read from Flash 1 while Byte 4 (at odd address 0x90000003) is read from Flash 2.

Cautions:

- In dual-Flash memory mode both device models must be identical, because in this mode the same commands and addresses are issued in parallel to both Flash memories. In the case that the two Flash-memory devices are different, the dual-Flash mode must be disabled (DFM = 0) and each Flash memory could be used in standalone, allowing either Flash 1 or Flash 2 to be enabled using QUADSPI_CR[7] FSEL bit.
- For all hardware configurations listed in [Table 9 on page 23](#), each memory device is configured in Quad-SPI mode. It is possible to connect each device in single or dual-SPI mode. If DFM = 1, both devices must be configured in the same way, otherwise each device could have different hardware configuration when DFM = 0.
- The size must include both flashes: the Flash memory size, as specified in FSIZE[4:0] (QUADSPI_DCR[20:16]) should reflect the total Flash memory capacity, which is the double of the size of one individual component.

Table 9. Dual-Flash memory hardware configurations

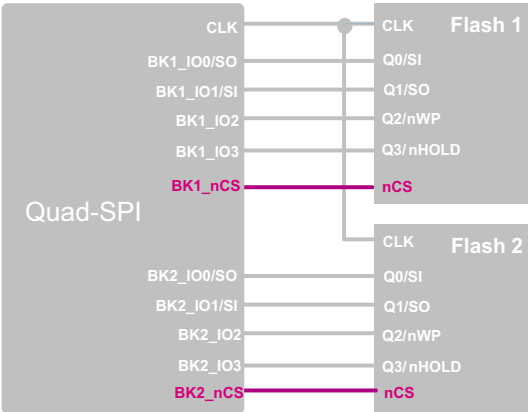
Used nCS	nCS configuration	Flash mode		Hardware configuration
2 nCS enabled	Both nCS_BK1 and nCS_BK2 not connected together	Single Flash DFM = 0 ⁽¹⁾	FSEL = 0 Flash 1 enabled	
			FSEL = 1 Flash 2 enabled	
		Dual-Flash memory DFM = 1		

Table 9. Dual-Flash memory hardware configurations (continued)

Used nCS	nCS configuration	Flash mode	Hardware configuration
1 nCS enabled	nCS_BK1 connected to both devices	Dual-Flash memory DFM = 1	<p>MSv41118V1</p>
	nCS_BK2 connected to both devices	Dual-Flash memory DFM = 1	<p>MSv41119V1</p>

1. When single-Flash memory mode is selected DFM = 0, the user can switch between Flash 1 or Flash 2 using FSEL bit.

Pink lines highlight the used chip select.

2.2.5 DDR and SDR mode

The SDR mode is activated by default, DDR mode allows to sample at rising and falling edge of each clock cycle and allows to double the throughput. DDR mode allows boosting the throughput and the execution performances; it is also very useful when the system clock (HCLK) is low and does not allow the Quad-SPI operating at maximum speed.

- SDR: data sent on CLK falling edge and sampled on CLK rising edge.
- DDR: data sent on both CLK edges, during the address, alternate or data phases. The sample is done one half CLK later.

When using DDR (Dual Data Rate), also known as DTR (Dual Transfer Rate) the user should consider the following:

- The trigger points and the configuration procedure are the same as in SDR.
- The command is sent on a full cycle like in the SDR mode.
- The alternate, data and address phases are sent on both edges.
- The dummy cycles are counted on full cycles like in SDR mode.

2.3 Three operating modes

2.3.1 Indirect mode

The indirect mode is used in below cases:

- For reading, writing or erasing operations
- If there is no need for AHB masters to access autonomously the QSPI memory (available in memory-mapped mode)
- For all the operations to be performed through the Quad-SPI data registers using CPU or using DMA
- To configure the QSPI Flash memory.

In the indirect mode, all operations are performed through the Quad-SPI registers where both read and write operations are available and managed by software. The Quad-SPI interface behaves like a classical SPI interface. The transferred data goes through the data register with FIFO. The data exchanges are driven by software or by DMA, using related interrupt flags in the Quad-SPI status registers.

The read and write operations are always performed in burst unless the amount of data is equal to one. The amount of data to be transferred is set in the QUADSPI_DLR register. In this mode it is possible to read or write data from or to external Flash memory with sizes up to four Gbytes.

The automatic-polling mode is available to generate an interrupt when the status-register inside the Flash memory is changing (useful for checking the end of the erase or the end of programming).

In case of an erase or programming operation, the indirect mode has to be used and all the operations have to be handled by software. In this case, it is recommended to use the status-polling mode and then poll the status register inside the Flash memory to know when the programming or the erase operation is completed.

2.3.2 Status-flag polling mode

The status-flag polling mode is used in below cases:

- To read QSPI Flash memory status register
- To poll autonomously for the end of an operation: Quad-SPI polls the status register inside the memory

The interface can automatically poll a specified register inside the memory and relieve the CPU from this task (useful when polling end of programming flag for example). This is a mode to check for example when an erase operation is completed and to know that an interrupt could be generated.

The Quad-SPI interface can also be configured to periodically read at a defined rate a register in the QSPI Flash memory. The returned data can be masked to select the bits to be evaluated. The selected bits are compared bit per bit with their required values stored in the match register. The result comparison can be treated in two ways:

- ANDed mode: if all the selected bits are matching, an interrupt is generated when it succeeds (stop on match flag)
- ORed mode: if one of the selected bits is matching, an interrupt is generated when it succeeds (stop on match flag)

When a match occurs, the Quad-SPI interface can stop automatically. The READ STATUS REGISTER command is used by many memory manufacturers as Micron or Spansion to read continuously the status register.

2.3.3 Memory-mapped mode

The memory-mapped mode is used in below cases:

- For reading operations
- To use external QSPI Flash memory like an internal memory, so any AHB master can read data autonomously
- For code execution from external QSPI Flash memory.

In memory-mapped mode the external memory is seen by the system as it was an internal memory. This mode allows all AHB masters to access to QSPI memory as an internal memory. The CPU can execute code from the QSPI memory as well.

When memory-mapped mode is used, a prefetching mechanism fully managed by the hardware allows to optimize the read and the execution performances from external the QSPI memory. Given that all the communication phases such as sending opcode or address are managed by QSPI peripheral, a 32-bytes FIFO is used for prefetching; this optimized prefetch mechanism avoids software overhead.

The programmed instructions and frame are sent automatically when an AHB master is accessing the memory-mapped space. Once the Quad-SPI peripheral is configured, the QSPI memory is accessed as soon as there is a read request on the AHB; this is done in the QSPI memory mapped address range. This action is totally transparent for the user.

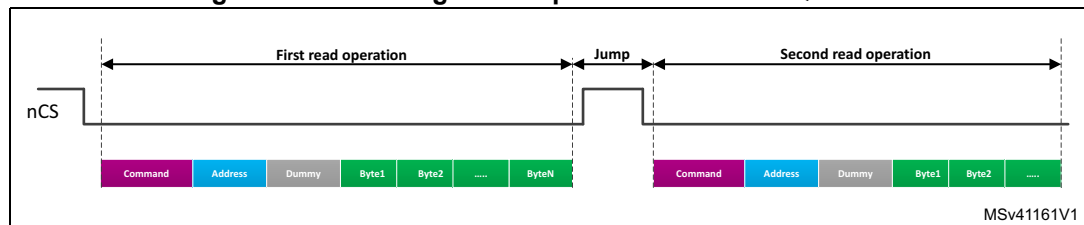
The LTDC master can access autonomously to the external Flash memory where all the access operations are fully managed by the QSPI interface. In the meanwhile, the Cortex[®]-M CPU is executing code from the internal Flash memory.

The Quad-SPI interface is able to manage up to 256Mbytes Flash memory starting from 0x90000000 to 0x9FFFFFFF in the memory mapped mode.

EXecute in place (XIP)

The prefetch buffer supports execution in place, therefore the code can be executed directly from the external QSPI memory. The Quad-SPI anticipates the next CPU access and loads in advance the byte at the following address. If the subsequent access is indeed made at a continuous address, the access will be completed faster since the value is already prefetched.

Figure 12. Executing non-sequential code from Quad-SPI



Bootling from QSPI Flash memory

Boot from the QSPI memory is not supported but the user can boot from the internal Flash memory and then configure the Quad-SPI in memory-mapped mode and then the execution starts from the QSPI memory. For more details on how to execute from the external QSPI memory, please refer to [Section 5.2 on page 61](#).

Note: *Reading the QUADSPI_DR in memory-mapped mode has no meaning and returns 0.*

For all supported STM32 series, the QSPI memory is accessible by Cortex®-M through System bus. For the STM32L4, the Quad-SPI is also accessible through the I-Code and the D-Code buses when a physical remap is enabled at address 0, which allows better execution performances.

When the Quad-SPI is remapped at address 0x00000000, only 128 Mbytes are remapped. Even when aliased in the boot memory space, the QSPI memory is still accessible at its original memory space.

Note: *The data length register QUADSPI_DLR has no meaning in the memory-mapped mode.*

2.4 Special features

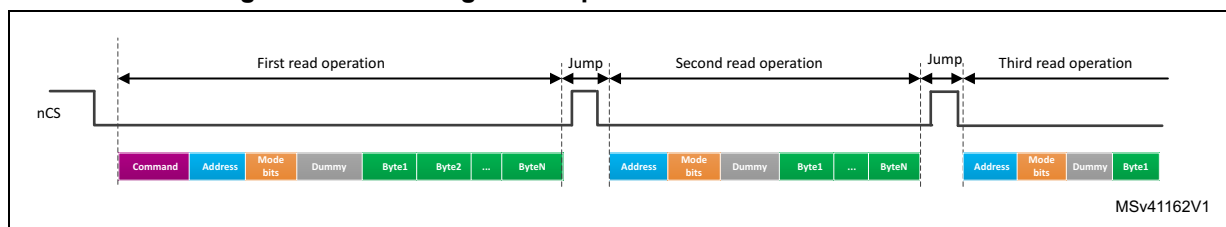
2.4.1 Send instruction only-once (SIOO)

The SIOO feature is named also by some memory manufacturers as “continuous read mode” or even as “burst mode”. This feature is available for all Quad-SPI modes: indirect, automatic-polling and memory-mapped. It is recommended to use this feature in order to reduce command overhead and boost the execution performances. When SIOO is enabled, the command is sent only once when starting the reading operation, then only the address is sent.

The command is sent only at first when starting the read operation. If a new read operation occurs, only one address is sent allowing to reduce up to eight cycles (in single I/O mode) for the command. This is a very interesting feature to reduce access overhead to the QSPI memory.

Data are prefetched continuously while the FIFO is empty, when a discontinuous access is detected, the Quad-SPI rises chip-select and starts a new read operation without sending the command but sending directly the new address.

Figure 13. Executing non-sequential code from Quad-SPI with SIOO enabled



The SIOO feature is supported by many QSPI memory manufacturers such as Micron or Spansion, nevertheless before using it, the user have to check if the feature is supported by the used memory.

To enable the SIOO mode, the user should:

- Configure the memory by entering the SIOO mode. Refer to relevant manufacturer's datasheet for more details on how to enter this mode (make sure that the read command to be used does support this mode). Note that an alternate byte (mode Bits) needs to be sent in order to keep the device in this mode. Refer to SIOO example on [Section 5.2 on page 61](#) for more details on enabling this feature.
- Configure the Quad-SPI peripheral by setting the SIOO bit in QUADSPI_CCR register.

2.4.2 Delayed data sampling

The delayed data sampling is useful when the signals are delayed due to constraints on the PCB layout optimizations allowing to compensate this delay. The sampling clock can be shifted by an additional half cycle after data is driven by the Flash memory, this is done to guarantee that the data is ready at the sampling moment. This feature is not supported in DDR mode. To enable sampling shift set the SSHIFT bit in QUADSPI_CR register.

In DDR mode, the output data can be shifted by one quarter of the Quad-SPI output clock cycle in order to relax the hold constraints. To enable this output data delay, set the DHHC bit in QUADSPI_CR register. For more details on Quad-SPI's timing characteristics refer to the relevant product's datasheet.

2.4.3 Timeout counter

The timeout counter can be used to reduce the QSPI memory power-consumption by releasing the nCS and then putting the memory in a lower consumption state.

After each access in the memory-mapped mode, the Quad-SPI prefetches the subsequent bytes and holds these bytes in the FIFO. When the FIFO is full, the communication clock is stopped but the nCS pin remains low to keep the Flash memory selected and not resend a complete command to read the next bytes when location will be available in the FIFO.

To avoid any extra power-consumption in the external Flash memory, when the clock is stopped for a long time, the timeout counter can release the nCS pin; this action puts the external Flash memory in a lower-consumption state after a period of timeout elapsed without any access. Once FIFO becomes empty, the nCS will be low allowing read operations.

To use the timeout counter the user should:

- Enable it by setting the TCEN bit in the QUADSPI_CR register
- Program the timeout period TIMEOUT[15:0] in the QUADSPI_LPTR register

Note: When the timeout counter is enabled, for example in memory-mapped mode, if timeout occurs, nCS is raised; and for any new read access, a new read operation is started.

2.4.4 Additional status bits

Other than status flags described in [Table 13](#) in [Section 2.5.1: Interrupts usage on page 32](#), the QUADSPI_SR status register includes additional status bits, [Table 10](#) summarizes status flags.

Table 10. Additional status bits

Name	Size	Description
FLEVEL	5 bits	Number of valid bytes being held in the FIFO (for indirect mode)
BUSY	1 bit	This bit is set when an operation is ongoing. Clears automatically when operations are finished and FIFO is empty.

2.4.5 Busy bit and abort functionality

BUSY bit

The BUSY bit is used to indicate the state of the QSPI interface, it is set in the QUADSPI_SR register when an operation is ongoing and clears automatically when the operations are finished or aborted.

Note: Some Quad-SPI registers could not be written when the BUSY bit is set, so the user have to check if it is reset before writing to registers. Refer to the relevant reference manual to check if the register could be written or not when BUSY is set.

ABORT bit

When an application is running, any ongoing Quad-SPI operation can be aborted by setting the ABORT bit in the QUADSPI_CR register.

Note: Some Flash memories might misbehave if a write operation to a status registers is aborted.

[Table 11](#) summarizes different cases when the BUSY bit is reset in different QSPI operating modes:

Table 11. BUSY bit reset in different QSPI modes

QSPI mode	BUSY bit reset
Indirect mode	<ul style="list-style-type: none"> – The Quad-SPI has completed the requested command sequence and the FIFO is empty – Due to an abort
Automatic-polling mode	<ul style="list-style-type: none"> – After the last periodic access is complete, due to a match when APMS =1 – Due to an abort
Memory-mapped mode	<ul style="list-style-type: none"> – On a timeout event – Due to an abort – Quad-SPI peripheral is disabled

2.4.6 4-byte address mode

This mode is named also by some brands “extended address mode”. In this mode a 4-byte address is sent allowing to address the Flash memories with sizes up to four Gbytes. This mode is supported by many QSPI memory manufacturers such as Micron or Spansion.

Before using the 4-byte mode, the user has to check if it is supported by the used device.

To enable this mode, the user should:

- Configure the memory by entering the 4-byte mode. Refer to relevant manufacturer’s datasheet for more details on how to enter this mode.
 - For Micron devices for example, the ENTER 4-BYTE ADDRESS MODE “B7h” command should be used, then the user should use dedicated 4-byte address commands for some operations, such as read, program or erase, from the device datasheet.
 - For Micron devices, if a read operation is needed, the user should use 4-BYTE READ command “13h” instead of READ “03”
- Configure the Quad-SPI peripheral in 32-Bits address mode by setting ADSIZE[1:0] = 11 field in QUADSPI_CCR register.

Note: *The memory size has to be configured according to the fixed address size respecting the following formula: number of bytes in Flash memory = $2^{[FSIZE+1]}$ where $[FSIZE+1]$ is effectively the number of address bits required to address the Flash memory.*

[Table 12](#) summarizes the different address modes versus the maximum addressable memory space.

Table 12. Address mode versus maximum addressable memory space

Address length	QUADSPI_CCR ADSIZE [1:0]	QUADSPI_DCR [20:16] FSIZE [4:0]	Memory size $2^{[FSIZE+1]}$
8-Bits address	00	00111	Up to 256bytes
16-Bits address	01	01111	Up to 64Kbytes
24-Bits address	10	10111	Up to 16Mbytes
32-Bits address	11	11111	Up to 4Gbytes ⁽¹⁾

1. Only the first 256Mbytes of the QSPI memory can be read in memory-mapped mode (from 0x90000000 to 0x9FFFFFFF).

Cautions on 4-byte address mode:

- In indirect mode, if the address plus the data length exceeds the Flash memory size, TEF flag will be set as soon as the access is triggered.
- In memory-mapped mode, if an access is made to an address outside of the range defined by FSIZE but still within the 256Mbytes range, then an AHB error is given. The effect of this error depends on the AHB master that attempted the access; if it is the Cortex® CPU, a hard fault interrupt is generated and if it is a DMA, a DMA transfer error is generated while the corresponding DMA channel is automatically disabled.

2.5 Interrupts and DMA usage

2.5.1 Interrupts usage

The QSPI peripheral supports five different interrupts where each one is useful in a particular case. To be used, each interrupt has to be enabled by setting its corresponding enable bit and enabling the QSPI global interrupt on the NVIC side. [Table 13](#) summarizes all the supported interrupts.

Table 13. Quad-SPI interrupts summary

Interrupt	Event Flag ⁽¹⁾	Enable control bit ⁽²⁾	Clear bits ⁽³⁾	Description
Timeout	TOF	TOIE	CTOF	Timeout occurred
Status Match	SMF	SMIE	CSMF	Matching of the masked received data with the match register (automatic-polling mode only)
FIFO Threshold	FTF	FTIE	-	FIFO threshold reached (indirect mode)
Transfer Complete	TCF	TCIE	CTCF	Indirect mode: the correct number of data set in the QUADSPI_DLR register has been transferred All modes: the transfer has been aborted
Transfer Error	TEF	TEIE	CTEF	Indirect mode: out-of-range address has been accessed

1. All event flags are available in the QUADSPI_SR register

2. All enable-control bits are available in the QUADSPI_CR register

3. All clear bits are available in the QUADSPI_FCR register

2.5.2 DMA usage

DMA can be used to perform data transfers from or to the Quad-SPI external memory, this is possible when the Quad-SPI interface is configured either in indirect read/write mode or in memory-mapped mode. In memory-mapped mode only-read from QSPI memory is allowed.

DMA usage with Quad-SPI in indirect mode

DMA requests can be generated only in indirect mode when the Quad-SPI's FIFO threshold is reached and when the DMAEN bit is enabled.

The transfer is started when

- The DMAEN bit is set and the QSPI and DMA are configured
- The FTF flag is set when FIFO threshold is attained

If DMAEN = 1 already, then the DMA controller must be disabled before changing the FTHRES/FMODE.

In indirect mode the QSPI is seen as a peripheral which should be considered when configuring DMA direction. [Table 14](#) summarizes the different DMA requests and transfer directions versus the STM32 series.

Table 14. DMA requests mapping and transfer directions versus STM32 series

Product	DMA	Stream/Channel	Reading from Quad-SPI with DMA	Writing to Quad-SPI with DMA
STM32L4x6	DMA1	Channel 5 request 5	peripheral-to-memory	memory-to-peripheral
	DMA2	Channel 7 request 3		
STM32F446	DMA2	Stream7 Channel3		
STM32F469/F479	DMA2			
STM32F7x5/F7x6	DMA2			

DMA usage with Quad-SPI in memory-mapped mode

When the memory-mapped mode is configured, the QSPI memory is seen as an internal memory that should be considered when configuring DMA direction. In this mode, the Quad-SPI peripheral can not generate a DMA transfer.

In memory-mapped mode either DMA1 or DMA2 can be used to transfer data from the external QSPI memory to any other memory or peripheral. To perform a transfer using DMA from external QSPI memory to any other memory or a peripheral, the user should configure the DMA by setting the source address (from 0x90000000), the destination address and the number of data to be transferred.

The DMAEN bit has no effect in memory-mapped mode, the transfer is started as soon as the DMA is accessing the Quad-SPI address range (from 0x90000000 to 0x9FFFFFFF).

Once the DMA configured transfer is started by software, the DMA will read the data from the QSPI memory exactly as an internal memory. The Quad-SPI peripheral manages the communication with the external memory and puts the read data in the FIFO.

The number of data items to be transferred is managed by the DMA so the user should configure the number of data in the DMA's DMA_SxNDTR register (or DMA_CNDTRx register for STM32L4x6). There is no need to configure the QUADSPI_DLR register as it has no effect in the memory-mapped mode where the DMA is the flow controller.

Note: The DMA's FIFO can be used for example if the DMA Burst mode is required to reduce the transfer overhead on the bus matrix.

2.6 Low-power modes

The STM32 power state is an important requirement that must be considered as it has a direct effect on the Quad-SPI interface state. For example, if the MCU is in standby mode then the Quad-SPI has to be reconfigured after wakeup from this mode.

Note: The QSPI memories can also be put in low-power mode. Depending on the memory brand, some devices support both standby mode and deep power-down mode while other devices support only standby mode.

In order to save more energy when the application is in low-power mode, it is recommended to put the QSPI memory in low-power mode before entering the STM32 in low-power mode. More information on reducing power consumption is available on [Section 6.2 on page 82](#).

It is possible to perform transfers in sleep mode while the CPU is stopped thanks to the STM32's smart architecture and to the fact that in sleep mode all peripherals can be enabled. This can fit wearable applications where the low-power consumption is a must.

An AHB master such as DMA could continue the transfers from the Quad-SPI (when memory-mapped mode is used) even after entering the MCU in SLEEP mode. Once the transfer is completed an interrupt can be generated to wake-up the STM32.

2.6.1 STM32L4x6 low-power mode

The Quad-SPI is active in run, sleep, low-power run and low-power sleep mode. A Quad-SPI interrupt can cause the device to exit sleep or low-power sleep mode. In Stop1 or Stop2 mode, the Quad-SPI is frozen, the content of its registers is kept. In standby or shutdown mode, the Quad-SPI is powered down and it must be reinitialized afterwards.

Table 15. QSPI peripheral in different power modes on STM32L4

Mode	Description
Run	Active
Low-power run	
Sleep	Active. Peripheral interrupts cause the device to exit sleep mode.
Low-power sleep	Active. Peripheral interrupts cause the device to exit low-power sleep mode.
Stop 1	Frozen. Peripheral registers content is kept.
Stop 2	
Standby	Powered-down. The peripheral must be reinitialized after exiting standby or shutdown mode.
Shutdown	

2.6.2 STM32F446, STM32F469, STM32F479, STM32F7x5 and STM32F7x6 low-power mode

The Quad-SPI is active in both run and sleep mode. A Quad-SPI interrupt can cause the device to exit the sleep mode. In stop mode, the Quad-SPI is frozen and the content of its registers is kept. In standby mode, the Quad-SPI is powered-down and it must be reinitialized afterwards.

Table 16. QSPI peripheral in different power modes on STM32F446, STM32F469, STM32F479, STM32F7x5 and STM32F7x6

Mode	Description
Run	Active.
Sleep	Active. Peripheral interrupts cause the device to exit sleep mode.
Stop	Frozen. Peripheral registers content is kept.
Standby	Powered-down. The peripheral must be reinitialized after exiting standby mode.

3 Quad-SPI configuration

This section describes all QSPI configuration steps required to perform either read, write or erase operations.

3.1 GPIOs configuration

The user should configure the GPIOs to be used for interfacing with the QSPI memory, and this is dependent on the preferred hardware configuration. For more details on the hardware configurations please refer to [Table 8 on page 19](#).

Note: *It is recommended to reset the Quad-SPI peripheral before starting a configuration and also to guarantee that the peripheral is in reset state.*

Depending on the GPIOs availability, the user can configure either Bank1 or Bank2 GPIOs. The user can also configure both of banks if two QSPI memories are connected.

Note: *All GPIOs have to be configured in high-speed mode.*

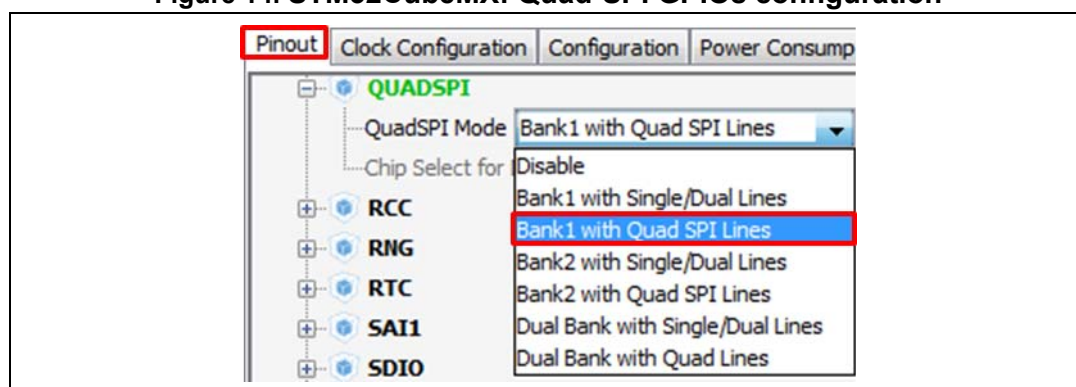
3.1.1 GPIOs configuration using STM32CubeMX tool

The following example shows how to configure Quad-SPI GPIOs in Quad I/O mode using Bank1 GPIOs.

Using the STM32CubeMX tool is a very simple, easy and rapid way to configure the Quad-SPI peripheral and its GPIOs as it allows to generate a project with preconfigured Quad-SPI.

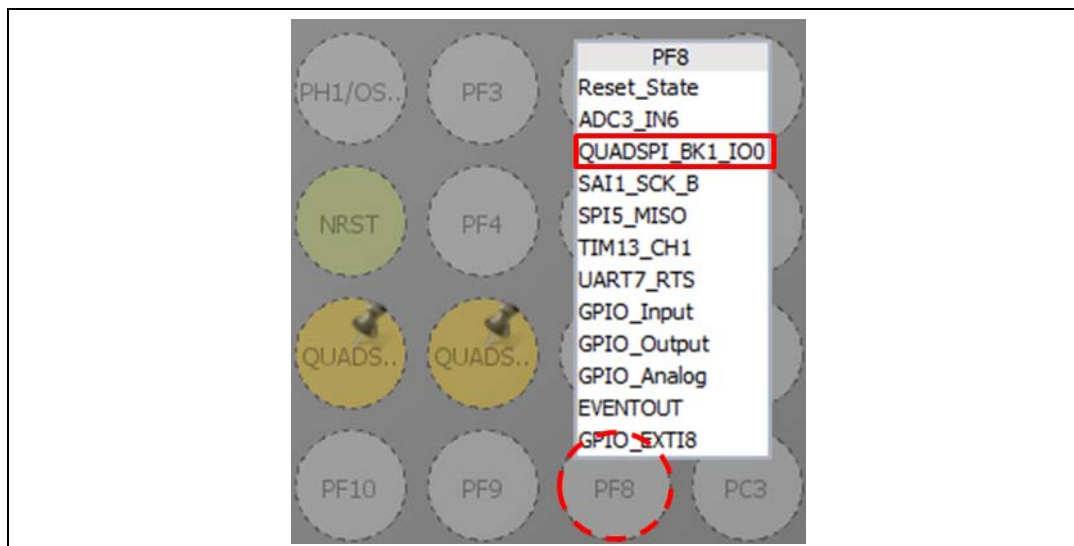
Once the STM32CubeMX project is created, in the Pinout tab choose one from the listed hardware configurations. [Figure 14](#) shows how to select the Quad-SPI hardware configuration with the STM32CubeMX where Bank1 GPIOs are used.

Figure 14. STM32CubeMX: Quad-SPI GPIOs configuration



If after selecting one hardware configuration (as shown in [Figure 14](#)) the used GPIOs does not match with the memory connection board, the user can configure the alternate function directly on the corresponding pins. For more details on Quad-SPI alternate functions availability versus GPIOs, refer to the alternate function mapping table in the relevant datasheet. [Figure 15](#) shows for instance how to configure manually a PF8 pin to QUADSPI_BK1_IO0 alternate function.

Figure 15. STM32CubeMX: PF8 pin configuration to QUADSPI_BK1_IO0 alternate function



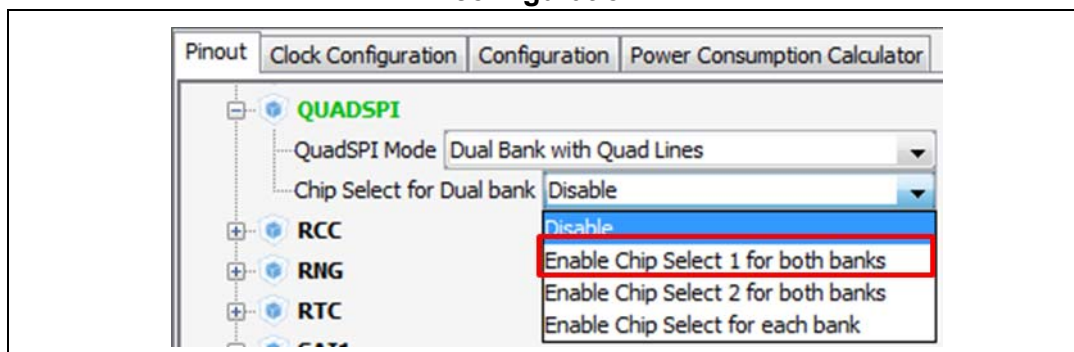
The used pins will be highlighted in green once the Quad-SPI interface GPIOs are configured correctly.

Dual-Flash memory case

If dual bank is selected, the user should select one from the listed chip-select configurations. For more details on different dual-Flash memory chip-select configurations please refer to [Section 2.2.4: Dual-Flash memory mode on page 21](#).

[Figure 16](#) shows how to configure chip-select 1 for both banks with STM32CubeMX.

Figure 16. STM32CubeMX: Dual-Flash memory Quad-SPI with chip-select 1 configuration

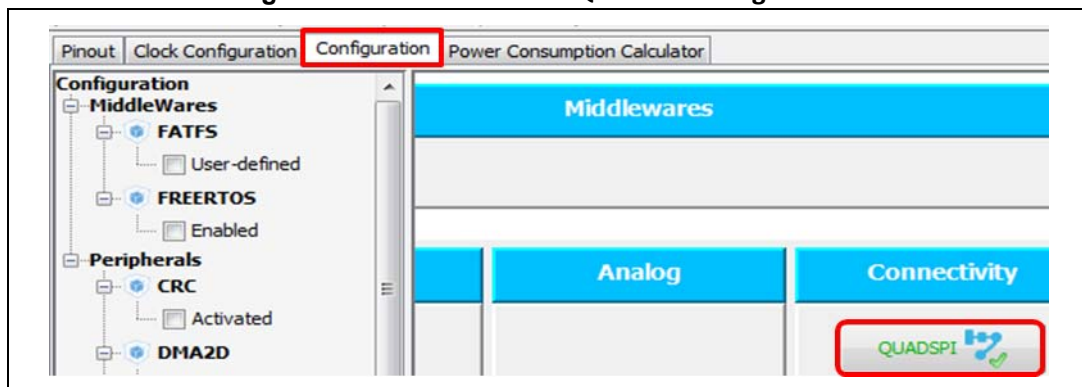


Enabling Quad-SPI interrupts

To be able to use Quad-SPI interrupts, the user should enable the QSPI global interrupt on the NVIC side. After that, each interrupt is enabled separately by enabling its corresponding enable bit (interrupt-enable bits are available in the QUADSPI_CR register described in [Table 13 on page 32](#)).

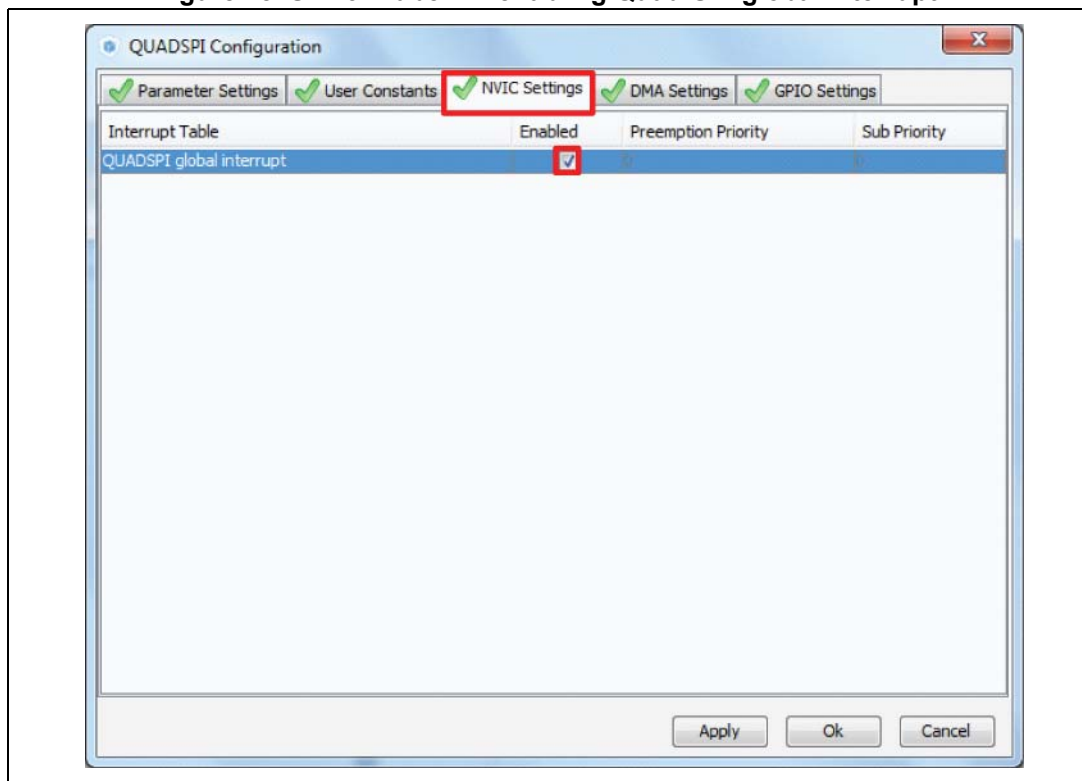
Select the configuration tab then click on the QUADSPI button as shown in [Figure 17](#).

Figure 17. STM32CubeMX: Quad-SPI configuration



In the window shown in [Figure 18](#) select the NVIC settings tab, check the Quad-SPI global interrupt then click on the OK button.

Figure 18. STM32CubeMX: enabling Quad-SPI global interrupt



3.2 Quad-SPI peripheral configuration and clock

3.2.1 Quad-SPI peripheral configuration (QUADSPI_CR register)

The Quad-SPI peripheral is configured using the QUADSPI_CR. The user must configure the clock prescaler division factor and the sample shifting settings for the incoming data. The DMA requests are enabled setting the DMAEN bit. In case of interrupt usage, their respective enable bit can also be set during this phase. The FIFO level either for DMA request generation or for interrupt generation is programmed in the FTHRES bits.

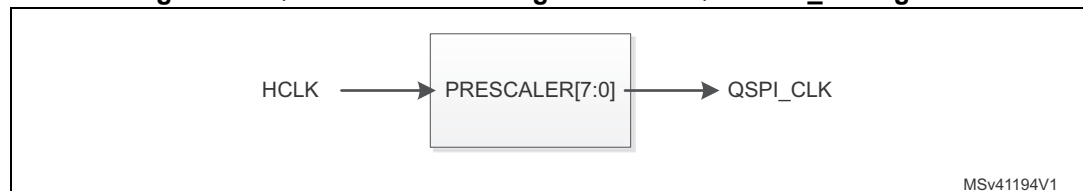
If timeout counter is needed, the TCEN bit can be set and the timeout value programmed in the QUADSPI_LPTR register.

The dual-Flash memory mode can be activated by setting DFM to 1.

The Quad-SPI clock source is the AHB where a prescaler is used to generate the Quad-SPI CLK. Depending on the programmed prescaler in the QUADSPI_CR register it is possible that both the CPU and the Quad-SPI work at the same speed (PRESCALER[7:0] = 0).

Figure 19 shows the Quad-SPI clock scheme where: $QSPI_Clk = F_{AHB} / (Prescaler + 1)$.

Figure 19. Quad-SPI clock configuration on QUADSPI_CR register



3.2.2 QSPI Flash memory parameters configuration (QUADSPI_DCR register)

The parameters related to the targeted external Flash memory are configured through the QUADSPI_DCR register. The user must program the Flash memory size in the FSIZE field, the chip-select minimum high-time in the CSHT field, and the functional mode (Mode 0 or Mode 3) in the MODE bit in the QUADSPI_DCR register.

Note: The Quad-SPI parameters can be changed when application is running but not during an ongoing transfer, in other words not when the busy bit is set. If it is needed to change during an ongoing transfer, the ABORT bit can be used to stop the ongoing operation then the configuration could be changed.

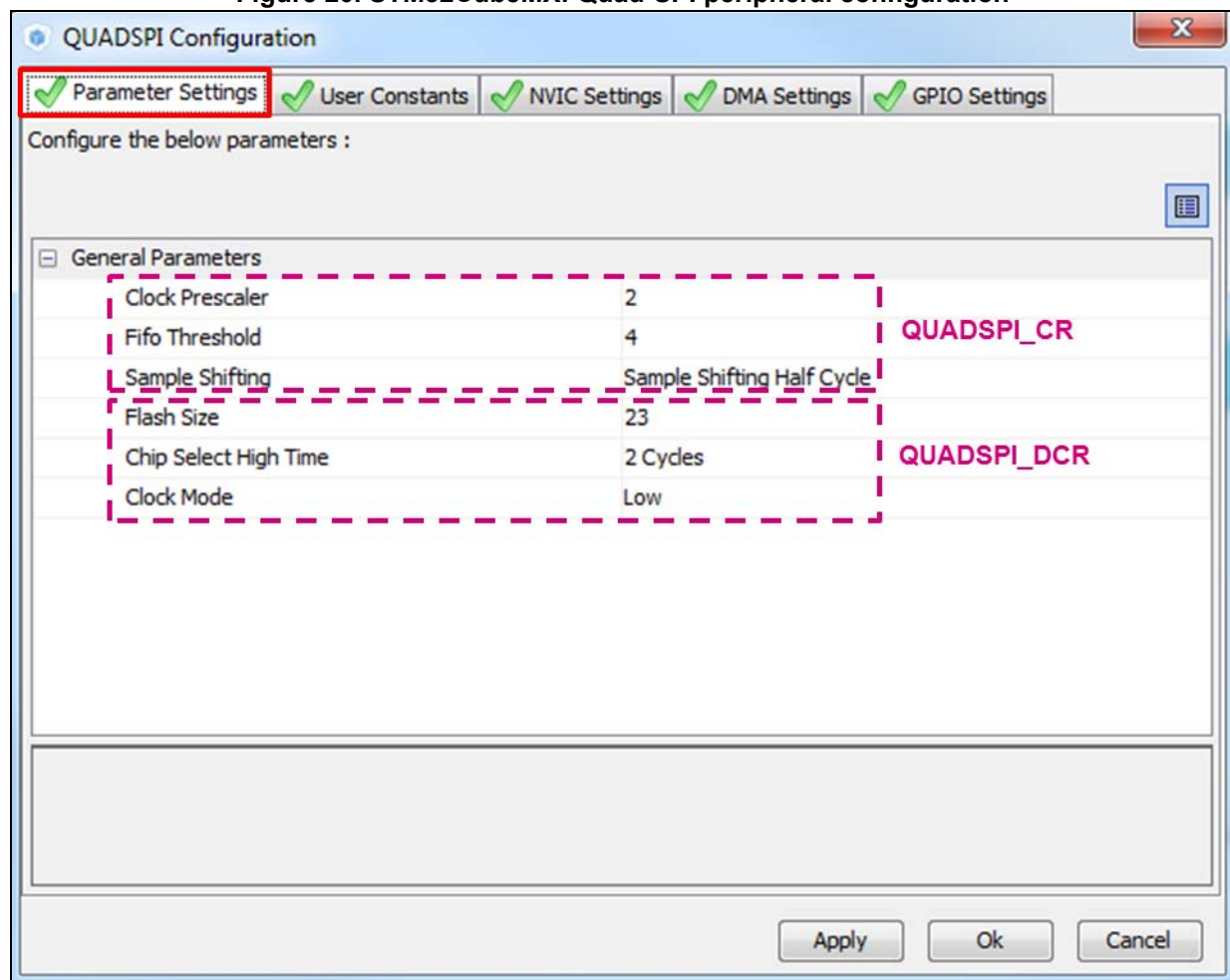
Quad-SPI configuration using STM32CubeMX

The STM32CubeMX tool can be used to configure the Quad-SPI peripheral. Once the GPIOs have been correctly configured, as already described, the Quad-SPI parameters can be configured. However, all configurations related to starting communication have to be added manually by the user in the project.

In the QUADSPI configuration window, select the Parameter Settings tab then configure the parameters. [Figure 20](#) shows an example of Quad-SPI configuration according to the following conditions:

- Clock Prescaler = 2 => $QSPI_CLK = F_{AHB}/3$
- FIFO Threshold = 4 => FTF flag is set as soon as there are five or more free bytes available to be written to the FIFO
- Sample shifting half cycle enabled
- Flash size is 16MByte => number of bytes in Flash memory = $2^{[FSIZE+1]} = 2^{[23+1]} => FSIZE = 23$
- Chip select high time = 2 cycles => $CSHT[2:0] = 1$
- Clock mode is low => clock mode 0 enabled

Figure 20. STM32CubeMX: Quad-SPI peripheral configuration



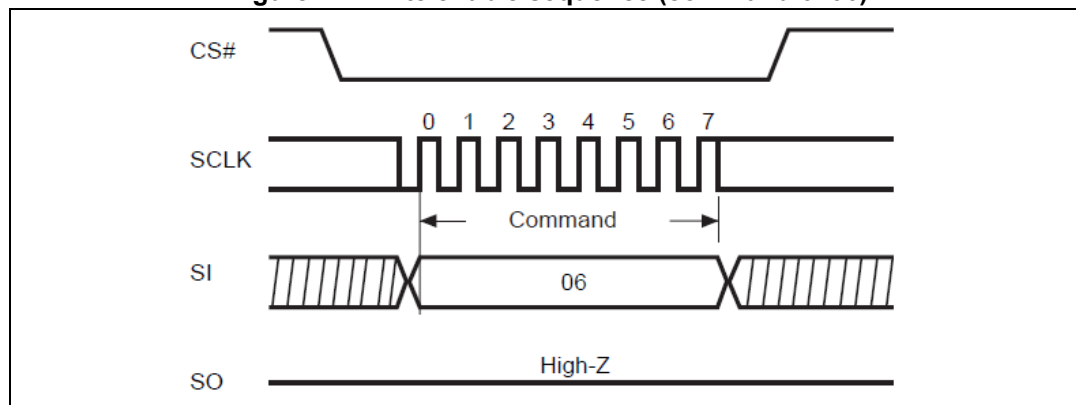
3.2.3 QSPI memory device configuration

Once the Quad-SPI peripheral is configured, the QSPI memory should be configured depending on the used Quad-SPI configuration using the indirect mode.

Enable writing to QSPI Flash memory

The write-enable command should be sent to the memory in order to set the write-enable latch (WEL) bit. This WEL bit must be set prior to every programming, Erasing, and write to the memory status register operation. This command is generally sent in one line without any address or data. The used Quad-SPI mode is 1-0-0 (instruction on 1 line - No Address - No Data).

Figure 21. Write enable sequence (command 0x06)



Configuring dummy cycle

The number of dummy cycles should be configured in the QSPI memory according to the operating clock speed, for that, the user should refer to the memory device's datasheet.

3.2.4 Starting a communication (QUADSPI_CCR register)

Once the Quad-SPI peripheral is configured, the communication can be started in one of the three modes: indirect-write or indirect-read mode, status-flag polling mode or memory-mapped mode. The operating mode is configured in the FMODE[1:0] field in QUADSPI_CCR register. Before starting the communication, the user should configure the frame format in the QUADSPI_CCR register. Refer to [Section 2.1 on page 13](#) for more details on frame format configuration.

Indirect-write mode (FMODE = 00)

Communication starts immediately if:

- A write is performed to INSTRUCTION[7:0] (QUADSPI_CCR), if no address is required (ADMODE = 00) and no data needs to be provided by the firmware (DMODE = 00)
- A write is performed to ADDRESS[31:0] (QUADSPI_AR), if an address is necessary (ADMODE != 00) and if no data needs to be provided by the firmware (DMODE = 00)
- A write is performed to DATA[31:0] (QUADSPI_DR), if an address is necessary (when ADMODE != 00) and if data needs to be provided by the firmware (DMODE != 00)

Indirect-read mode (FMODE = 01)

Communication starts immediately if:

- A write is performed to INSTRUCTION [7:0] (QUADSPI_CCR), and if no address is required (ADMODE=00).
- A write is performed to ADDRESS [31:0] (QUADSPI_AR), and if an address is necessary (ADMODE!=00).

Status-flag polling mode (FMODE = 10)

The accesses to the Flash memory begins in the same way as in the indirect-read mode, communication starts immediately if:

- A write is performed to INSTRUCTION [7:0] (QUADSPI_CCR) and if no address is required (ADMODE=00).
- A write is performed to ADDRESS [31:0] (QUADSPI_AR) and if an address is necessary (ADMODE!=00).

Memory-mapped mode (FMODE = 11)

Once the memory-mapped mode is configured, the communication starts as soon as there is an access request from any AHB master.

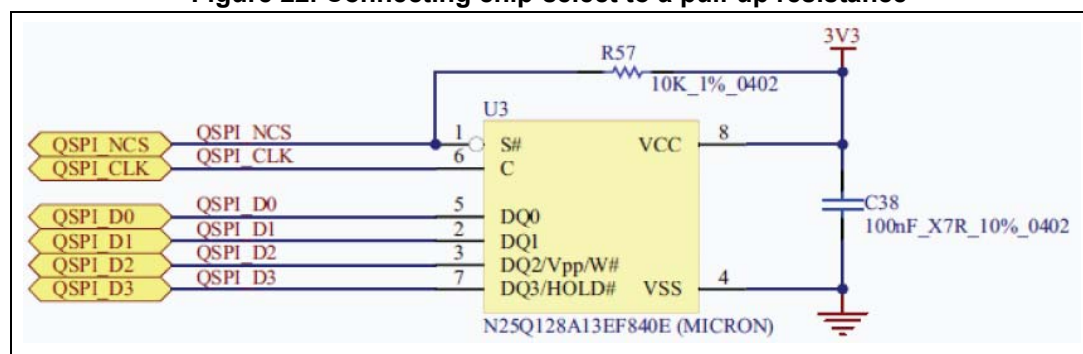
3.3 Hardware considerations

3.3.1 Pull-up resistance

Most Quad-SPI manufacturers recommend to connect a pull-up resistance to the VCC on the CS pin. This is to ensure that, during power-up, the voltage on the chip-select pin tracks that on the VCC. For more details on electrical recommendations please refer to datasheet of the relevant parts.

[Figure 22](#) shows an example of QSPI memory connection where a pull-up resistance is connected to the chip-select pin.

Figure 22. Connecting chip-select to a pull-up resistance



3.3.2 Good PCB design allows maximum Quad-SPI speed

The speed at which the Quad-SPI interface operates depends on many factors, including the board layout and the pad speeds. With a good layout it should be possible to reach the maximum speeds described in [Table 2: Quad-SPI availability and features across STM32](#)

families on page 8 and committed in the datasheet of the product.

The layout should be as good as possible in order to get the best performances. To get on PCB routing guidelines, please refer to the application note “*Getting started with STM32F74xxx/STM32F75xxx MCU hardware development*” (AN4661) section 8.4.3 Quadrature serial parallel interface (Quad SPI), available on the ST website.

3.3.3 Chip-select high time (CSHT)

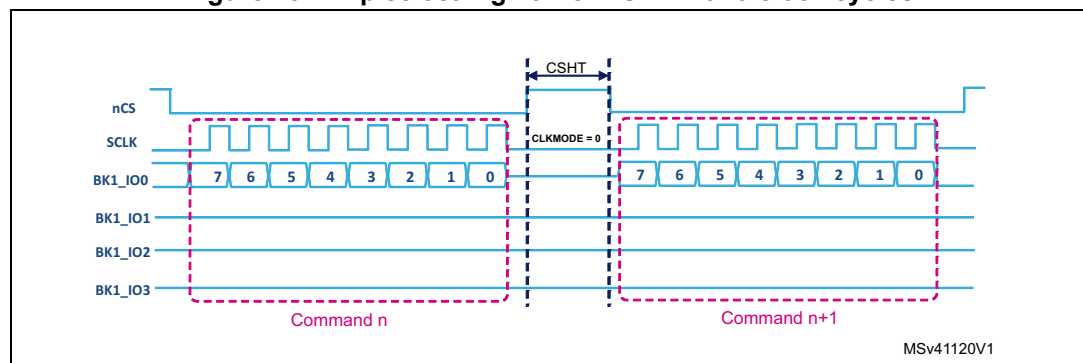
When the Quad-SPI sends two commands, one immediately after the other, it raises the chip-select signal (nCS) high between the two commands for only one CLK cycle by default.

If the Flash memory requires more time between commands, the chip-select high time CSHT[2:0] field in the QUADSPI_DCR register can be used to specify the minimum number of QSPI_CLK cycles (up to eight) that nCS must remain high.

3.3.4 CKMODE

The clock mode indicates the level that CLK takes between commands when nCS is high. Two modes are supported when nCS is high: mode 0 where CLK stays low and mode 3 where CLK stays high.

Figure 23. Chip select high time: CSHT = two clock cycles



3.3.5 Some considerations when using Quad-SPI in classical SPI mode

When using the Quad-SPI in classical SPI mode, the user should consider the following equivalences:

- **Mode 0** for Quad-SPI is equivalent to CPOL = 0 and CPHA = 0 for classical SPI
- **Mode 3** for Quad-SPI is equivalent to CPOL = 1, and CPHA = 1 for classical SPI

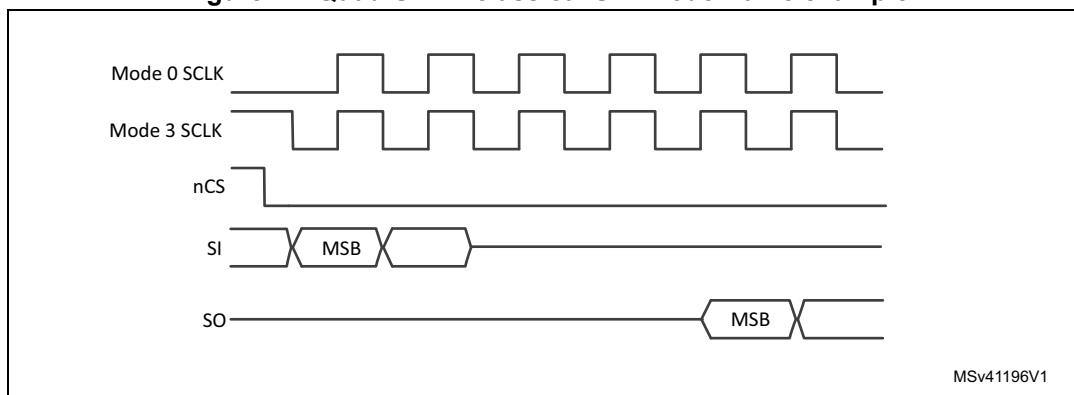
The main difference between the two modes is the clock polarity when the bus master is in standby mode and not transferring any data.

- SCLK will stay at logic low state with CPOL = 0, CPHA = 0
- SCLK will stay at logic high state with CPOL = 1, CPHA = 1

Note: Full duplex is not supported when using the Quad-SPI in classical SPI mode, so only half duplex is supported.

Figure 24 shows a classic SPI frame example highlighting the Quad-SPI clock modes equivalence with classical SPI.

Figure 24. Quad-SPI in classical SPI-mode frame example



4 Programming QSPI Flash memory

This section describes how to program a QSPI Flash memory in the following use cases:

- **For an end application development:** in this case the QSPI memory is programmed during the product's development with static data or code to be used in the final product. A dedicated Flash memory loader is needed allowing to place the data or the code to be used in the application. The Flash loaders provided by ST can be used for programming if the user is using one of the ST EVAL or Discovery boards, otherwise the user should develop its own Flash memory loader.
- **On the fly when application is running:** in this case the QSPI Flash memory is used in a final product as an external mass-storage device, allowing the application to store data any time that it is needed.

Note: For both cases the programming principle is the same, the only difference is that in the first case, the programming operation is performed with a tool and a Flash memory loader during the application's development, while in the second case the programming operation is performed during a running application in a final product. Only the indirect mode has to be used for programming either if it is a writing or an erasing operation.

Depending on the used Flash memory brand, different programming commands are available, so it is up to the user to configure the desired command supported by the device.

The instruction, address and data phases can be sent in one, two or four lines for command phase depending on the device brand.

The 4-Byte address mode can be used allowing to program the QSPI flashes with sizes up to 4Gbytes.

The automatic-polling mode is a very interesting mode that can be used for waiting while the programming operation is ongoing; when the operation is completed an interrupt can be generated.

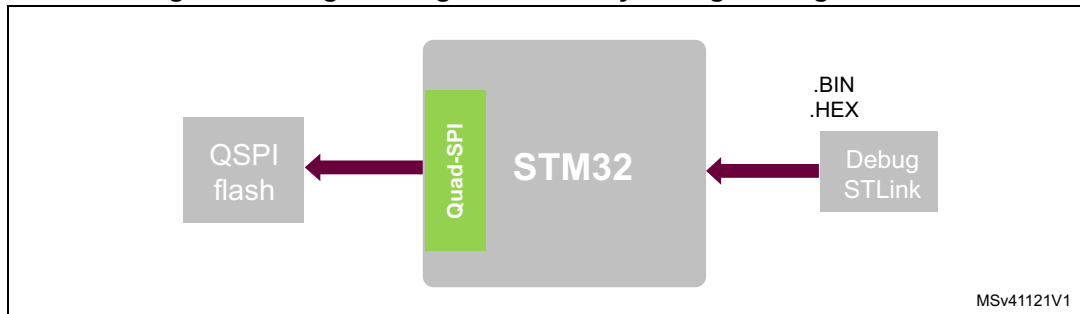
4.1 Programming code or data for an end application

This section describes how to program either static code or data to be used in the final application.

- Programming code for end application: the code for the application is placed in the QSPI memory to be executed by the CPU and then to be extended to the on-chip internal memory. An example of storing code in QSPI Flash memory is described in [Section 5.2 on page 61](#).
- Programming data for end application: this is useful in graphical applications, for example to store graphic content such as icons or images. An example of storing data in QSPI Flash memory for an end application is described in [Section 5.1 on page 56](#).

To program the QSPI Flash for an end application, either the STM32 ST-LINK utility or the integrated development environment can be used. This operation is done using the debug interface (SW, JTAG) through the STM32.

Figure 25. Programming QSPI memory through debug interface



External Flash loader: A dedicated algorithm is used to perform the programming operation, it is loaded to the STM32 through the debug interface then executed to perform the programming operation, and its input are the binary file to be programmed.

Only the QSPI Flash loaders for the memories mounted on the STM32 Evaluation and Discovery boards are provided. For other hardware, the user has to develop its own custom loader.

4.1.1 Programming QSPI Flash memory using the STM32 ST-LINK utility

When the used IDE is not supporting the QSPI programming capability such as System Workbench for STM32, the user can simply use the STM32 ST-LINK utility tool.

How to create a new QSPI Flash memory loader and add it to the ST-LINK utility

For each hardware configuration and for each QSPI Flash memory brand, a dedicated Flash memory loader should be developed. The user has to develop its own dedicated Flash memory loader (.stldr file) if the used hardware is different from ST boards.

A project is provided in the ST-LINK utility install directory "STMicroelectronics\STM32 ST-LINK Utility\ST-LINK Utility\ExternalLoader\N25Q256A_STM32L476G-EVAL_Cube" allowing to develop an external loader for a N25Q256A Flash memory on the STM32L476G-EVAL board. This project can be easily tailored to the user's dedicated hardware and then generate the external loader.

For more details on how to develop an external QSPI Flash memory loader for the STM32 ST-LINK utility, refer to the user manual *STM32 ST-LINK Utility software description* (UM0892), section 3.9 "Developing custom loaders for external memory" available at www.st.com.

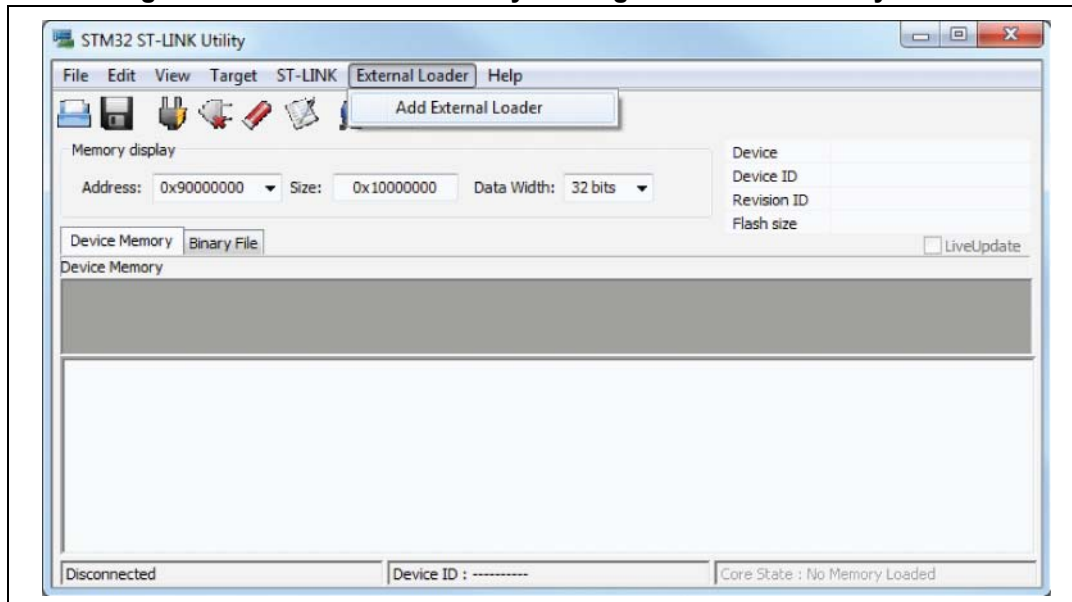
Caution: The used tool chain/compiler for generating the HEX/BIN file for programming the QSPI memory must be exactly the same as the one used for the application development.

To program a QSPI Flash memory with the ST-LINK utility, the dedicated Flash memory loader has to be added.

How to add a QSPI Flash memory loader to the ST-LINK utility

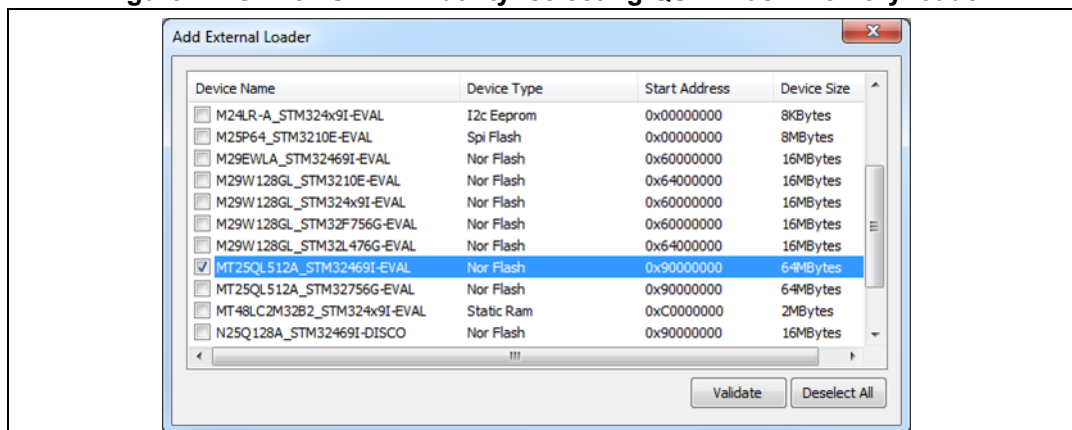
To add an external Flash memory loader, go to the External Loader then click on the Add External Loader button.

Figure 26. STM32 ST-LINK utility: adding QSPI Flash memory loader

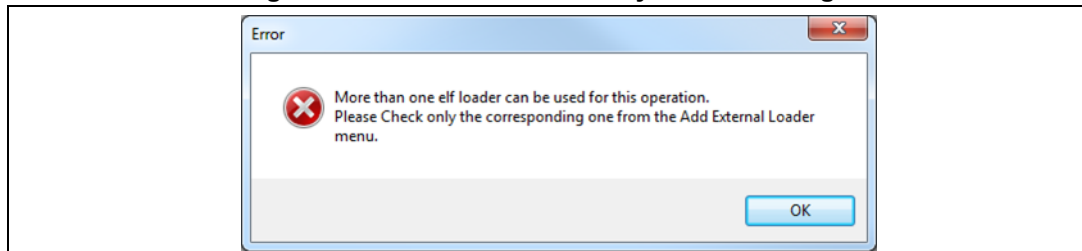


In the following window check your corresponding device and click on the Validate button.

Figure 27. STM32 ST-LINK utility: selecting QSPI Flash memory loader

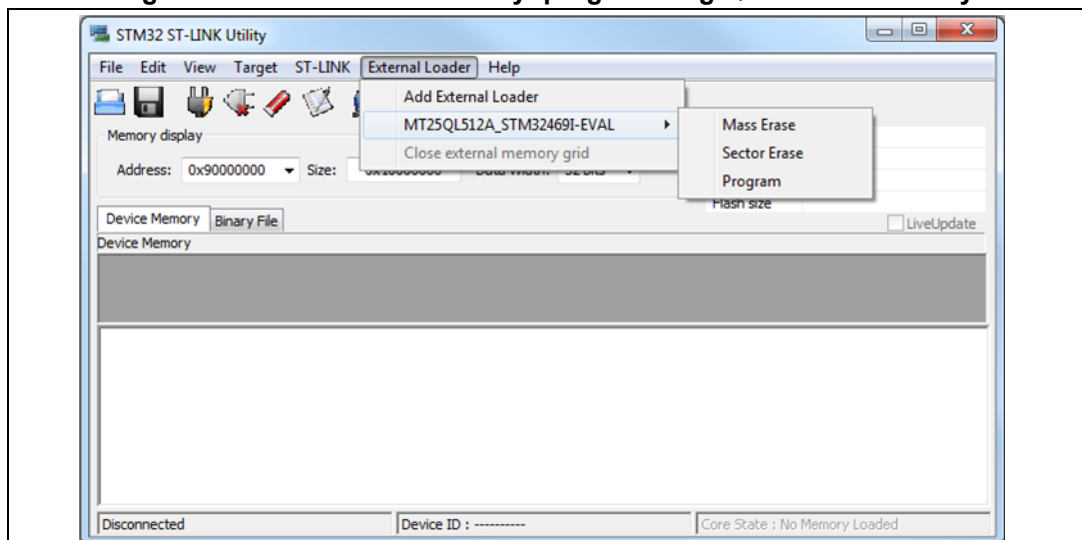


Note: No more than one external loader has to be added, otherwise the following error message appears. The user can remove one and add another Flash memory loader.

Figure 28. STM32 ST-LINK utility: error message**How to program QSPI Flash memory using the ST-LINK utility**

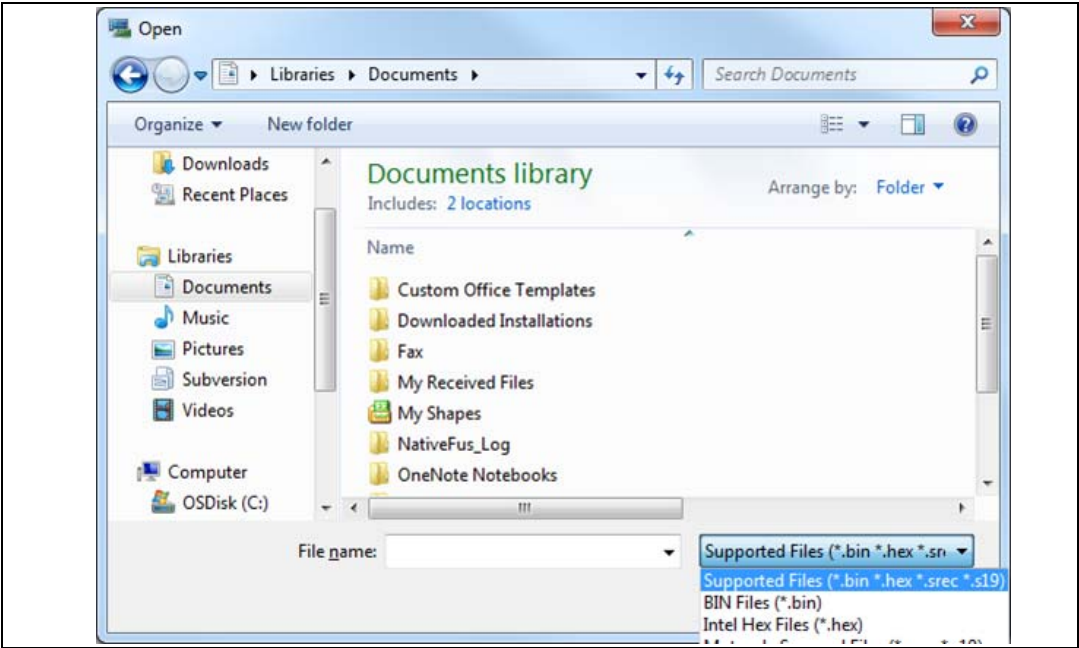
To program the QSPI Flash memory follow the steps below:

- Connect the board using the USB cable through the ST-LINK debug.
- In the External Loader go to the added external Flash memory loader then select Program.

Figure 29. STM32 ST-LINK utility: programming QSPI Flash memory

The following window appears allowing the user to add data to be programmed, which can be a binary file, an HEX file or Motorola S-record files (.srec or .s19).

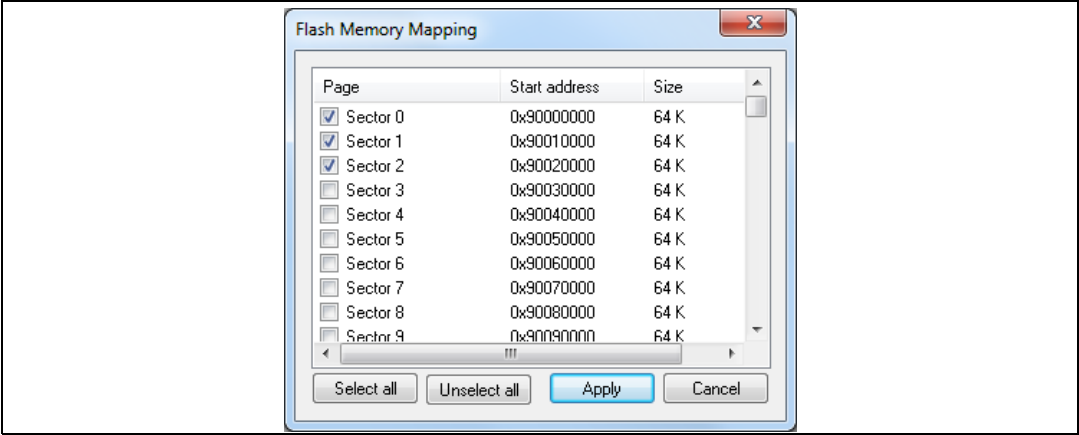
Figure 30. STM32 ST-LINK utility: selecting HEX file for programming



Erasing QSPI memory

To perform a mass-erase operation select Mass Erase, to erase sectors click on Sector Erase then check the sectors to be erased as shown in [Figure 31](#).

Figure 31. STM32 ST-LINK utility: erasing sectors



4.1.2 Programming QSPI Flash memory using IDE

Programming QSPI Flash memory using Keil

In the user project either the code, the data region or both to be programmed in the QSPI memory have to be specified to the linker before programming.

The following example shows how to add a QSPI dedicated load region in a Keil scatter file where also an execute region is created, allowing to execute the code from the QSPI memory.

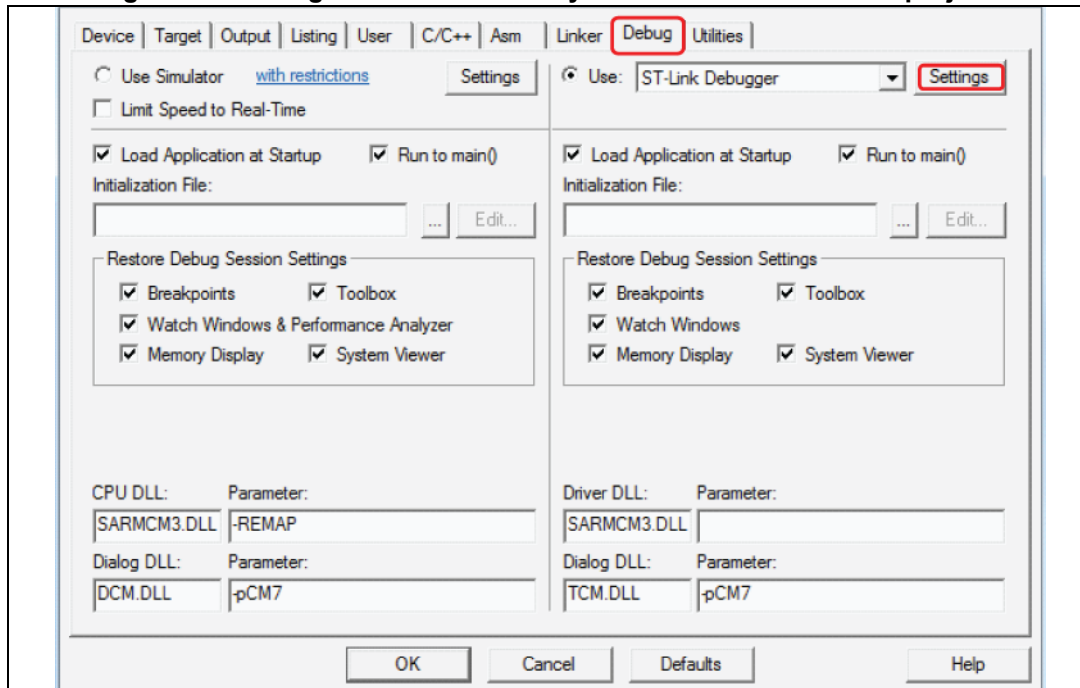
```
; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_IROM1 0x08000000 0x00100000 {      ; load region size_region
    ER_IROM1 0x08000000 0x00100000 { ; load address = execution address
        *.o (RESET, +First)
        *(InRoot$$Sections)
        .ANY (+RO)
    }
    RW_IRAM1 0x20000000 0x00050000 { ; RW data
        .ANY (+RW +ZI)
    }
}
LR_QSPI 0x90000000 0xFFFFFFFF {
ER_QSPI 0x90000000 0xFFFFFFFF {
*.o (.textqspi)
}
}
```

How to add a QSPI Flash memory loader to Keil MDK-ARM

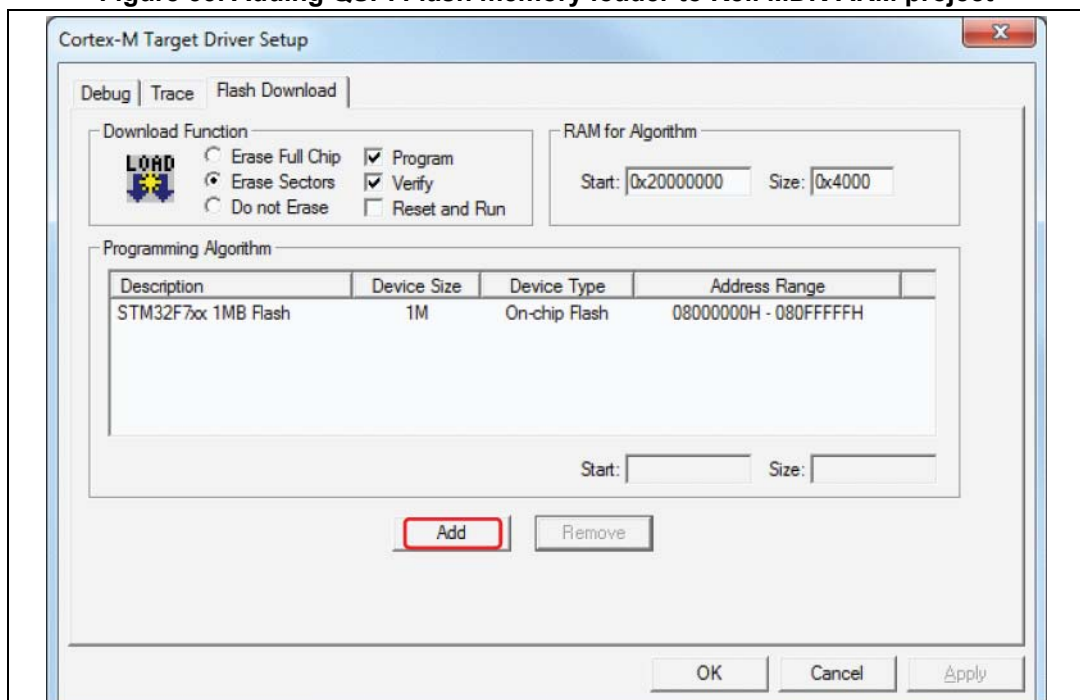
To add the QSPI Flash memory loader, go to Options for Target then in the Options Target window select the Debug tab then click on the Settings button as shown in [Figure 32](#).

Figure 32. Adding QSPI Flash memory loader to Keil MDK-ARM project



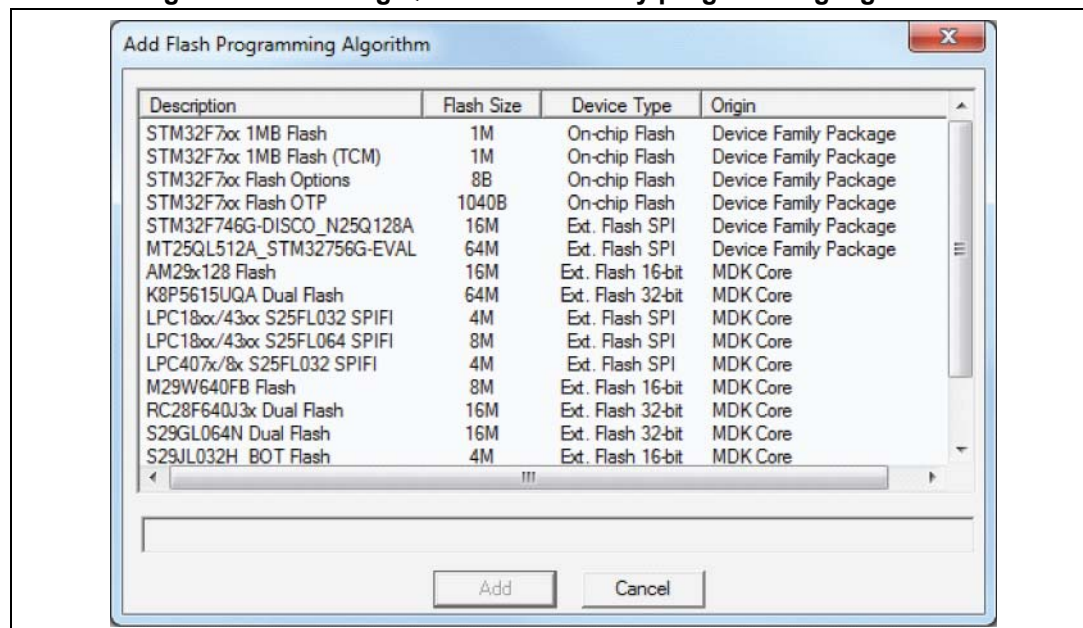
In the following window, to add the QSPI Flash memory loader click on the Add button.

Figure 33. Adding QSPI Flash memory loader to Keil MDK-ARM project



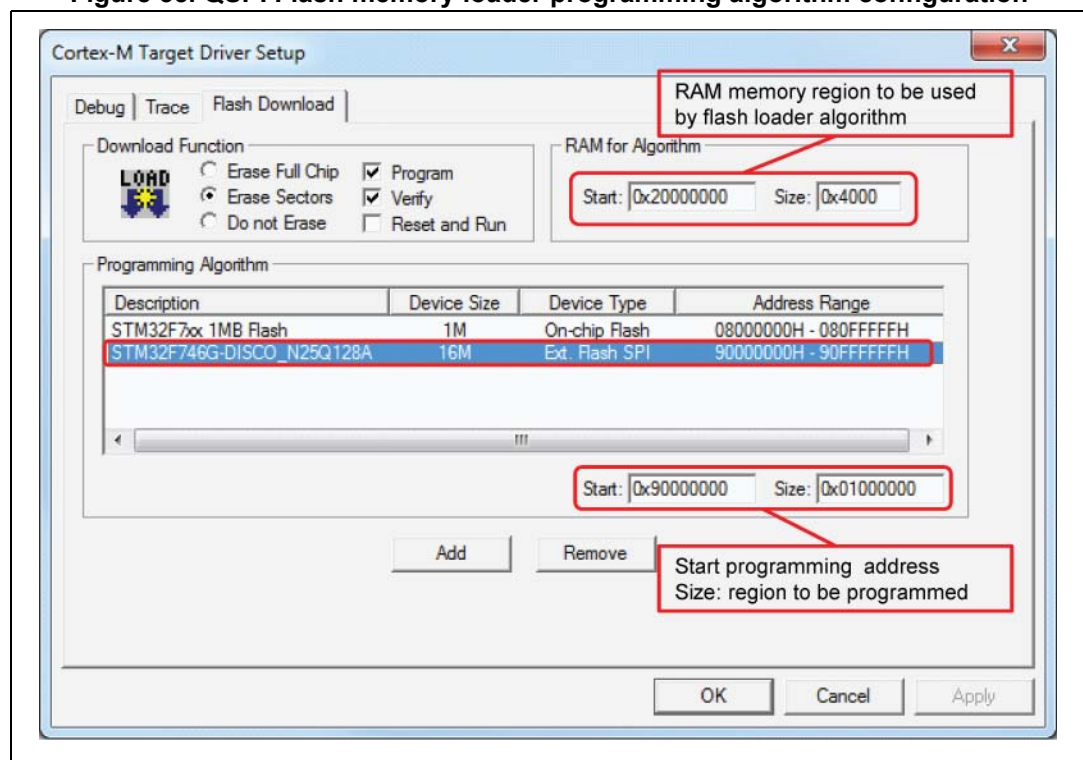
In the following window, select from the list the corresponding Flash memory loader:.

Figure 34. Selecting QSPI Flash memory programming algorithm



Once the corresponding Flash memory loader is added it will appear in the programming algorithm list as shown in [Figure 35](#).

Figure 35. QSPI Flash memory loader programming algorithm configuration



Once the QSPI Flash memory loader is added, programming can be done by clicking on the Load button or pressing F8 key on the keyboard.

Note: The region to be programmed is defined by default in the external loader and can be changed by changing the start address and the size fields shown in the already described figure.

How to proceed if we need to program QSPI Flash memory only once

During an application development, the user needs to debug its project and needs to load the code to the Flash memory many times. In each load-Flash operation, the QSPI Flash memory is loaded as well, which makes the loading operation too long. In this case, if the user already loaded data to the QSPI Flash and does not need to repeat the operation, the user can simply generate the Quad-SPI data related symbol definition file and add it to the project. The symbol definition file is a *.txt file for Keil MDK-ARM and a *.o file for IAR EWARM; it should replace, in the project, the original source code files (*.c or *.h) that were already programmed.

Another easier alternative consists in simply removing the already added QSPI Flash memory loader.

STM32 system workbench

The STM32 system workbench does not support a QSPI external Flash-loader, in that case the user can use the STM32 ST-LINK utility as previously described.

4.2 Storing and erasing data on the fly during running application

4.2.1 Storing data

It is useful in some applications to use the external QSPI Flash memory for data storage. In that case, the Quad-SPI interface has to be configured in indirect-write mode allowing to store data on the fly. Data to be stored can be the result of a processing such as signal processing for audio applications; it can also be storing images captured by a camera through the digital camera interface DCMI or any other data.

Before every programming operation an erasing operation has to be performed. Indirect-write mode has to be used for this operation.

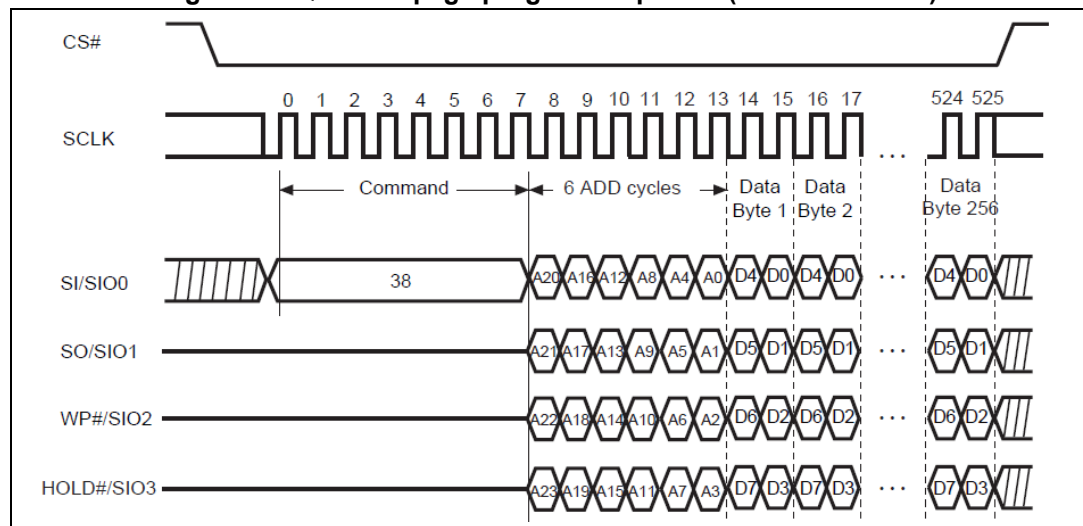
Note: *The writing speed of the external Flash memory is slower than its reading speed. As programming operation takes a considerable period of time, the user can use status-flag polling mode to poll the memory status register and once operation completed an interrupt could be enabled.*

To perform a programming operation the following steps should be done:

- Configure the Quad-SPI interface in the QUADSPI_CR and QUADSPI_DCR registers.
- Configure the Flash memory: enable writing to Flash, set the dummy cycles number depending on the dock speed, and any other needed.
- Configure the frame format in the QUADSPI_CCR register and start the programming sequence.
- Put the Quad-SPI interface in the status polling to poll the end of operation.

Figure 36 shows an example of a page programming sequence where the page size is 256 bytes.

Figure 36. Quad I/O page program sequence (command 0x38)



Note: *Either CPU with interrupts or DMA can be used for programming.*

Programming QSPI memory using indirect-write mode

When using this mode, all programming operations are handled by software by writing directly to the QUADSPI_DR register. An interrupt is generated when a transfer complete is identified or if FIFO threshold is reached.

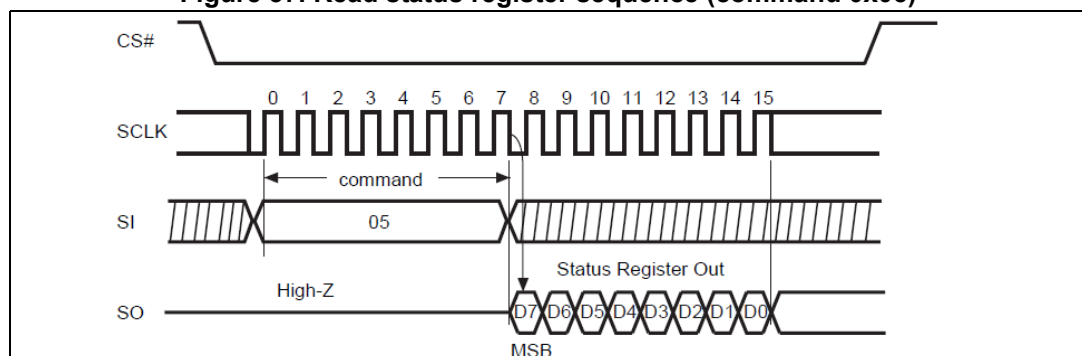
Programming QSPI memory using indirect-write mode with DMA

Depending on the user application, it is generally recommended to use DMA since it offloads the CPU. In some cases where the amount of data to be written to the memory is relatively small, there is no need to use DMA. Once the DMA is configured and the programming operation started, no intervention from the CPU is needed and the operation will end autonomously. For more details on DMA usage, refer to [Section 2.5.2 on page 32](#).

Usage of status-polling mode

The user can use this mode to poll the memory status register. [Figure 37](#) shows an example of status-register reading sequence.

Figure 37. Read status register sequence (command 0x05)



4.2.2 Erasing data

An erasing operation has to be performed before every programming operation. Indirect-write mode have to be used for this operation as well as for programming.

To perform an erasing operation, the required configurations are the same as for a programming operation, except that no data has to be written to the memory. As the data phase is not needed, only the instruction and the address phases are required.

As erasing operation takes a period of time, the user can use status-flag polling mode to poll the memory status register and once operation completed an interrupt could be enabled:

To perform an erasing operation the following steps should be done:

- Configure the Quad-SPI interface in the QUADSPI_CR and QUADSPI_DCR registers.
- Configure the Flash memory: enable writing to Flash.
- Configure the frame format and the indirect mode in the QUADSPI_CCR register
- Send the erasing command and address if needed (address is needed for sector erasing).
- Put the Quad-SPI interface in status-polling to poll the end of the operation.

Most of the Flash memory devices support a sector erasing and a full-chip erasing operations and some of them support an additional erasing operation allowing more flexibility. The user should refer to the manufacturer's datasheet for more details on the supported erasing operations.

Note: *If the used memory size is larger than 16Mbytes, 4-Bytes address mode have to be used, in that case the user should choose the 4-Byte command from the memory's datasheet.*

Sector-erase sequence

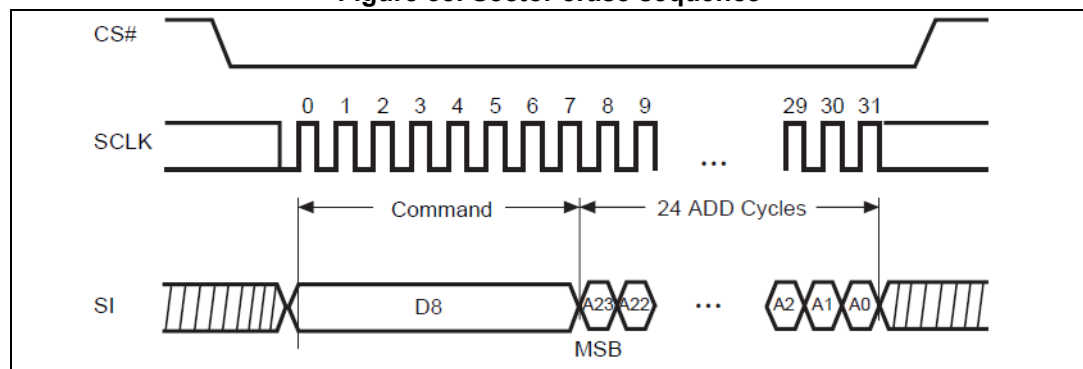
To erase a sector on the memory, a sector-erase command and a starting sector address should be sent.

Example: to perform a sector-erase operation on the MICRON N25Q512A memory, the QUADSPI_CCR register should be configured as below:

```
QUADSPI->CCR = 0x000025D8; /* Instruction= 0xD8; IMODE = 0x01; ADMODE=
0x01; ADSIZE = 0x02 */
QUADSPI->AR = 0x00000000; /* Address 0x00000000 is sent to erase the first
sector */
```

Figure 38 shows an example of a sector-erasing sequence:

Figure 38. Sector erase sequence

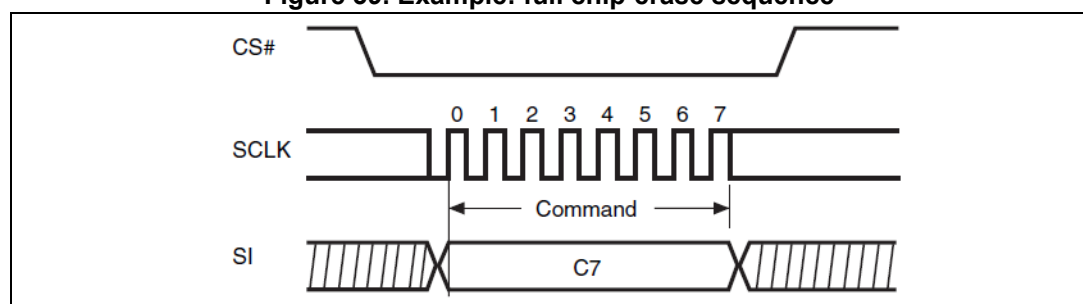


Full-chip erase sequence (bulk erase)

To erase the whole memory, no need to send an address, so only a command is sent.

Figure 39 shows an example of a chip-erase sequence:

Figure 39. Example: full chip-erase sequence



5 QSPI application examples

This section provides some typical Quad-SPI use case examples showing how to use the interface in indirect-mode, status-flag polling mode and memory-mapped mode.

Some of these examples are provided in the STM32Cube firmware package while others are based on other application notes also available on the ST website. Some hardware implementation examples are provided as well at the end of this section.

The following examples are described in this section:

- Memory-mapped mode: reading data in a graphical application
- Memory-mapped mode: executing code from the QSPI Flash memory
- Indirect mode: storing data on the fly during a running application
- Indirect mode: erasing data
- Hardware implementation example

5.1 Reading data from QSPI memory: graphical application

As the graphical applications require a large amount of static data (font libraries, HMI style, icons, etc...), the external QSPI Flash memory can be fully dedicated to the static data storage. This section provides two graphic use cases where the Quad-SPI is used for data storage:

- Frame buffer content generation from the QSPI memory
- Displaying images directly from the QSPI memory

5.1.1 Frame buffer content generation from QSPI memory

This section provides an example where the DMA2D peripheral is reading images stored in the external QSPI Flash memory to write them on the external SDRAM. It also prepares the frame buffer content to be read by LTDC and then displayed on a TFT-LCD display.

The example is based on a software demonstration from the STM32F7Cube firmware package available on the ST website. This example includes one project that has been developed for the STM32F746G-DISCO board.

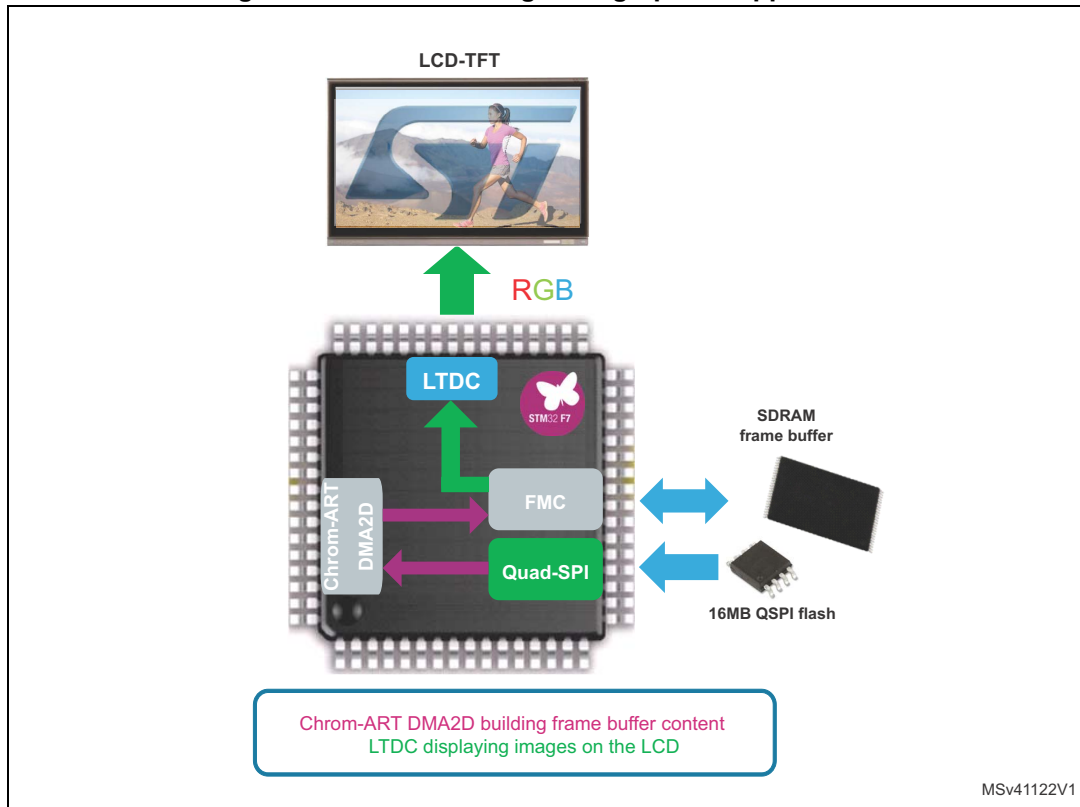
Use case description

This example describes how to use pictures stored on the QSPI Flash memory to build frame buffer content.

The DMA2D is used to render several animated layers on the LCD-TFT. All pictures and icons are stored in the QSPI Flash memory. The DMA2D is used to transfer pictures from the QSPI Flash memory to the SDRAM memory. During this transfer, the transfer time is measured, then the transfer speed is calculated and displayed on the TFT-LCD.

At the same speed, the DMA2D is blending images and loading them to the SDRAM (frame buffer) while the LTDC is updating the LCD-TFT at 60Hz. [Figure 40](#) describes this use case.

Figure 40. Quad-SPI usage in a graphical application



Storing images and icons to be used for the application

All images and icons should be stored on the external QSPI memory to be used in the application. Six images and three icons are available in the project, all of them are included in the “main images.c” file, where each one is defined as a constant in a dedicated header file.

In this project configuration, only two images (img2 and img6 defined respectively in files “img2.h” and “img6.h”) and three icons (icon_S, icon_T and icon_M defined respectively in the files “icon_S.h”, “icon_T.h” and “icon_M.h”) are used and need to be stored to the QSPI Flash memory.

The icons and the images are defined as constants. In order to store these icons and images to the QSPI memory, they are placed in a dedicated section “.textqspi” as described below:

- Img2 and img6 definition

```
_attribute__((section(".textqspi"))) const unsigned char img2[261120] = {}
_attribute__((section(".textqspi"))) const unsigned char img6[261120] = {}
```

- Icon_S, icon_T and icon_M definition:

```
_attribute__((section(".textqspi"))) const unsigned char icon_S[30800] = {}
_attribute__((section(".textqspi"))) const unsigned char icon_T[30800] = {}
_attribute__((section(".textqspi"))) const unsigned char icon_M[42000] = {}
```

As explained in [Section 4.1.2 on page 49](#), to program the QSPI memory using Keil MDK-ARM or IAR EWARM, a dedicated section for all the data to be programmed should be created.

Following the Quad-SPI section “.textqspi” in the Keil MDK-ARM scatter file:

```
LR_IROM2 0x90000000 0xFFFFFFFF { ; load region size_region
    ER_IROM2 0x90000000 0xFFFFFFFF { ; load address = execution address
        *.o (.textqspi)
    }
}
```

To program data to QSPI Flash, the user should first check if the QSPI Flash loader is added to the project (Keil MDK-ARM or IAR EWARM). If it is already added, the user can simply build the project and program the memory. For more details on project configuration, the user can refer to the readme file in the project's directory.

Quad-SPI interface and memory configuration

The Quad-SPI is configured in the memory-mapped mode to allow the DMA2D to read images and icons from the Quad-SPI and to be written into the SDRAM through the FMC interface.

The Cortex®-M7 is running at 200MHz while the Quad-SPI speed is 100MHz. At this clock speed, the maximal reachable throughput is 50Mbytes per second. The Quad-SPI is configured to operate in 1-4-4 mode. Since the embedded QSPI memory does not support the DDR mode, the SDR mode is used.

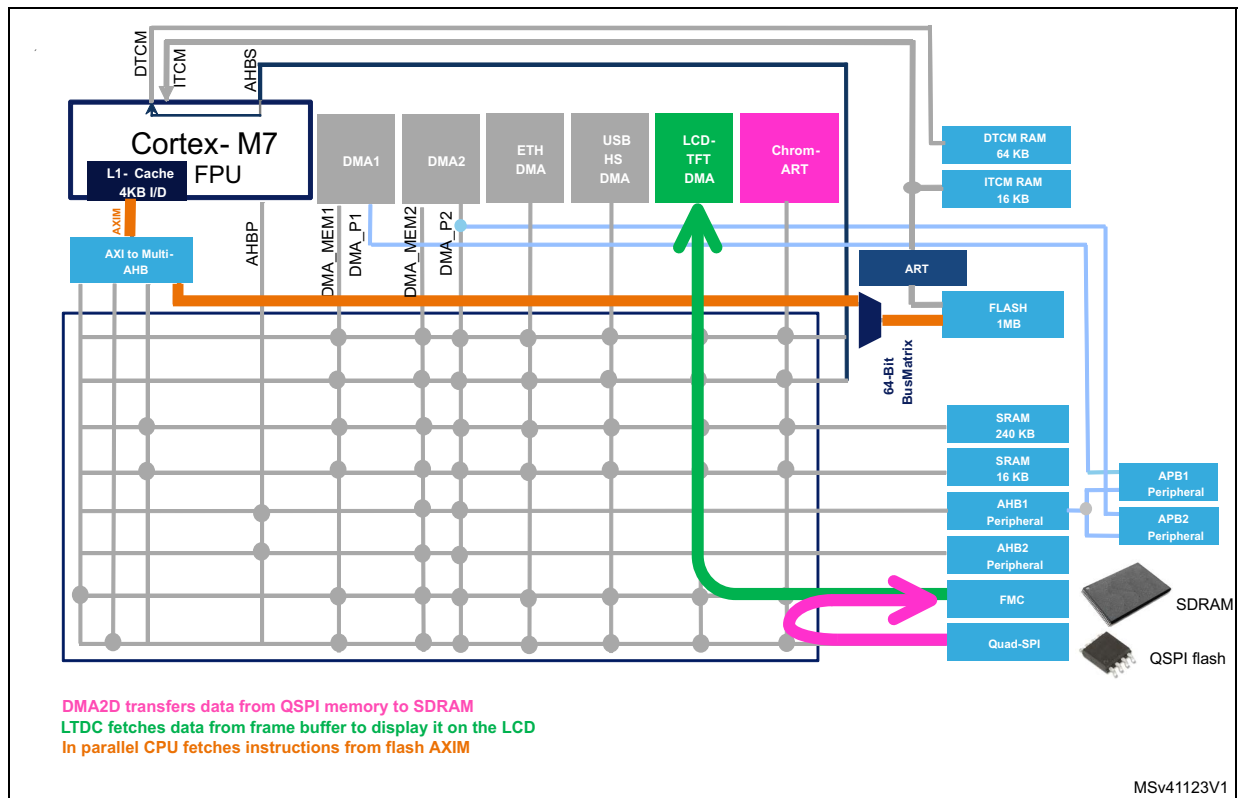
The used read command is the QUAD INPUT/OUTPUT FAST READ (0xEB) allowing to send the address in four lines and read data in four lines as well while the command is sent in one line.

The STM32F7's smart architecture allows to offload the CPU, since the DMA2D acting as an AHB master can perform all transfers from the Quad-SPI to the frame buffer (SDRAM) instead of the CPU. At the same time, when the LTDC is displaying graphics, the Cortex®-M7 can execute code from the internal Flash memory.

This demonstration shows how to interface a 16Mbytes external Flash memory with the STM32F7x6.

The QSPI Flash memory is used as a non-volatile support containing all graphics (icons, images) needed in the application. [Figure 41](#) describes this use case.

Figure 41. DMA2D reading images from Quad-SPI to build frame buffer content



5.1.2 Displaying images directly from the QSPI memory

As previously mentioned, all AHB masters can access the QSPI memory in memory-mapped mode. This is a very interesting feature when the application requires to display QSPI stored graphics directly on the LCD without any CPU intervention.

This section provides an example where the LTDC peripheral is reading an image stored in the external QSPI Flash memory. The example is based on a software demonstration from the application note “*Managing low-power consumption on STM32F7 Series microcontrollers*” (AN4749).

The AN4749 is provided with the X-CUBE-LPDEMO-F7 software package and is available on the ST website. It includes one project that has been developed for the STM32F746G-DISCO board.

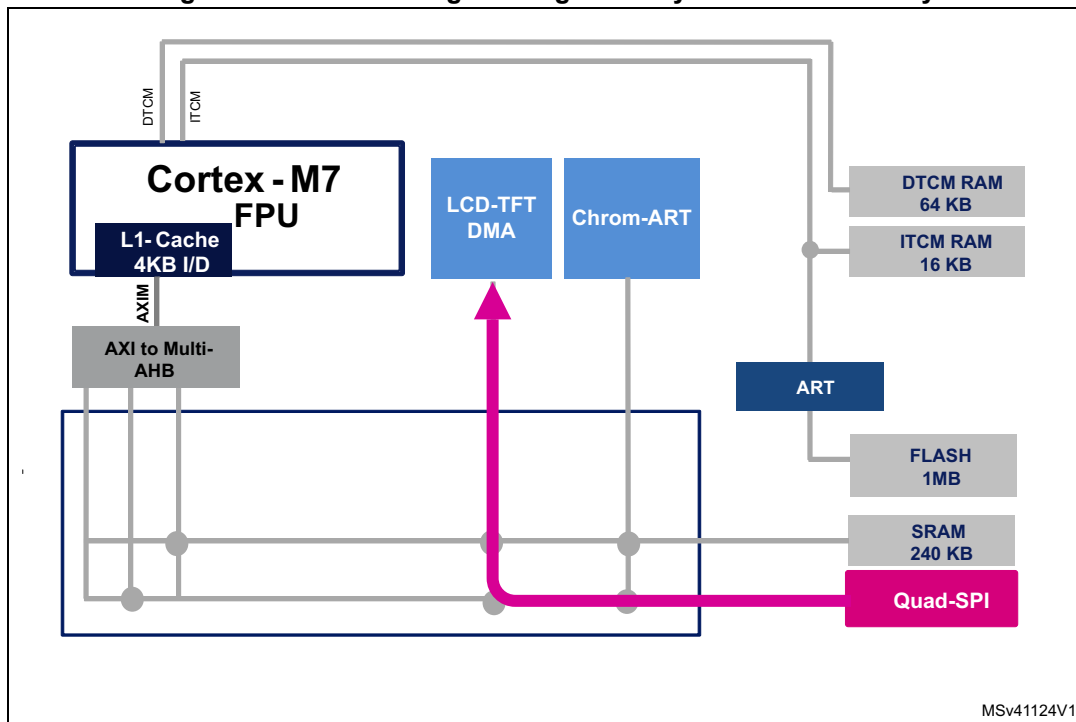
Note: For this example, we will only focus on describing the Quad-SPI usage.

Software demonstration description

An image “screensaver” is located in three different regions: the internal Flash memory, the external QSPI Flash memory at the address 0x90000000 and the SDRAM at the address 0xC0000000. The user should choose from which region the LTDC will display this image.

At first run, the time is displayed on the LCD-TFT for five seconds, then a menu appears showing a box allowing the user to choose the memory location from which the image will be read. If the QSPI Flash memory is selected, then the LTDC reads the image directly from the external Flash memory. [Figure 42](#) describes this use case.

Figure 42. LTDC reading an image directly from QSPI memory



Storing the image to be used for the application

For this application the image should be stored only once to be used in the end application (final product).

The image to be programmed is defined in the RGB565_480x272.h file as a constant data and is located in the RGB565_480x272_qspi dedicated section. The process to place the image in this section with Keil MDK-ARM is described below:

```
const uint16_t RGB565_480x272_QSPI[]
__attribute__((section(".RGB565_480x272_qspi"))); /* Keil MDK-ARM: placing
the image in .RGB565_480x272_qspi section */
const uint16_t RGB565_480x272_QSPI[]={};
```

As explained in [Section 4.1.2 on page 49](#), to program the QSPI memory using Keil MDK-ARM or IAR EWARM, a dedicated section for all the data to be programmed should be created.

Following the Quad-SPI section ".RGB565_480x272_qspi" in the Keil MDK-ARM scatter file:

```
LR_IROM2 0x90000000 0xFFFFFFFF { ; load region size_region
ER_IROM2 0x90000000 0xFFFFFFFF { ; load address = execution address
*.o (.RGB565_480x272_qspi);
}
}
```

To program data to the QSPI Flash, the user should first check if the QSPI Flash loader is added to the project (Keil MDK-ARM or IAR EWARM). If it is already added, the user can simply build the project and program the memory. For more details on the project configuration, the user can refer to the readme file in the project's directory.

QSPI interface and memory configuration

The Quad-SPI is configured in memory-mapped mode to allow the LTDC to read the image directly from the QSPI memory.

Once the Quad-SPI is configured, to display the image stored in the QSPI memory, the following API was called in the LCDConf.c file: `HAL_LTDC_ConfigLayer(&hltdc_F, &pLayerCfg, 0)`.

`pLayerCfg` is the pointer to a `LTDC_LayerCfgTypeDef` structure that contains the address of the image in the QSPI memory.

5.2 Executing from external QSPI memory: extend internal memory size

Using the external QSPI memory allows to extend the total application's available memory space. The STM32F746 embedded on the Discovery or the Evaluation boards integrates one Mbyte Flash memory, so connecting an external QSPI Flash memory extends the available memory space to 64 MBytes for the Evaluation board.

This section describes how to use the external QSPI memory to extend the internal Flash memory allowing code execution from this external memory. The software demonstration from application note *STM32F7 Series system architecture and performance* (AN4667) is selected as a reference to show how to:

- Configure the QSPI in memory-mapped mode during the system's initialization before jumping to the QSPI memory code
- Place application's code in the external QSPI memory

Software demonstration description

The AN4667 is provided with the X-CUBE-32F7PERF embedded software package available on the ST website which includes two projects: `STM32F7_performances` and `STM32F7_performances_DMAs`. Both projects are provided with Keil MDK-ARM tool chain. In this section we will only focus on `STM32F7_performances` project.

The `STM32F7_performances` project shows STM32F746's performance when executing code from the internal and external memories. It includes seven configurations where each one allows to select the data and the code's locations. Since in this section we will only focus on describing the code execution from the QSPI memory, only `6_1-Quad-SPI_rwRAM-DTCM` and `6_2-Quad-SPI_rwRAM-DTCM` configurations will be described in this section.

The number of cycles consumed by the FFT process is calculated based on the system-tick timer. The example was run on the STM32756G-EVAL and the results are shown on the LCD-TFT or on the hyperterminal through the UART or on the IDE printf viewer.

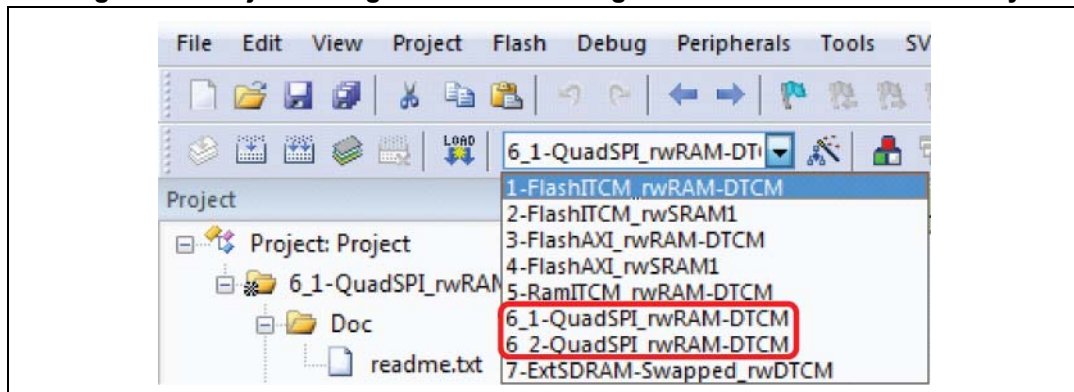
The configuration is also displayed showing the current project configuration, the system frequency, the different configurations of caches, ART, ART-Prefetch (ON/OFF) and the memory configuration in case of an external memory (SDRAM or Quad-SPI).

The software demonstration is developed for STM32756G-EVAL board and can be easily tailored to the STM32F746-DISCO board. For more details on this application note please refer to the AN4667 document available at www.st.com.

The STM32f7_performances project is located in the following path: x-cube-32F7perf\STM32CubeExpansion_AN4667_F7_V1.0.0\Projects\STM32756G_EVAL\stm32f7_performances.

Figure 43 shows the project configurations in Keil MDK-ARM that are described in the following section:

Figure 43. Project configurations: executing code from QSPI Flash memory



Projects configuration

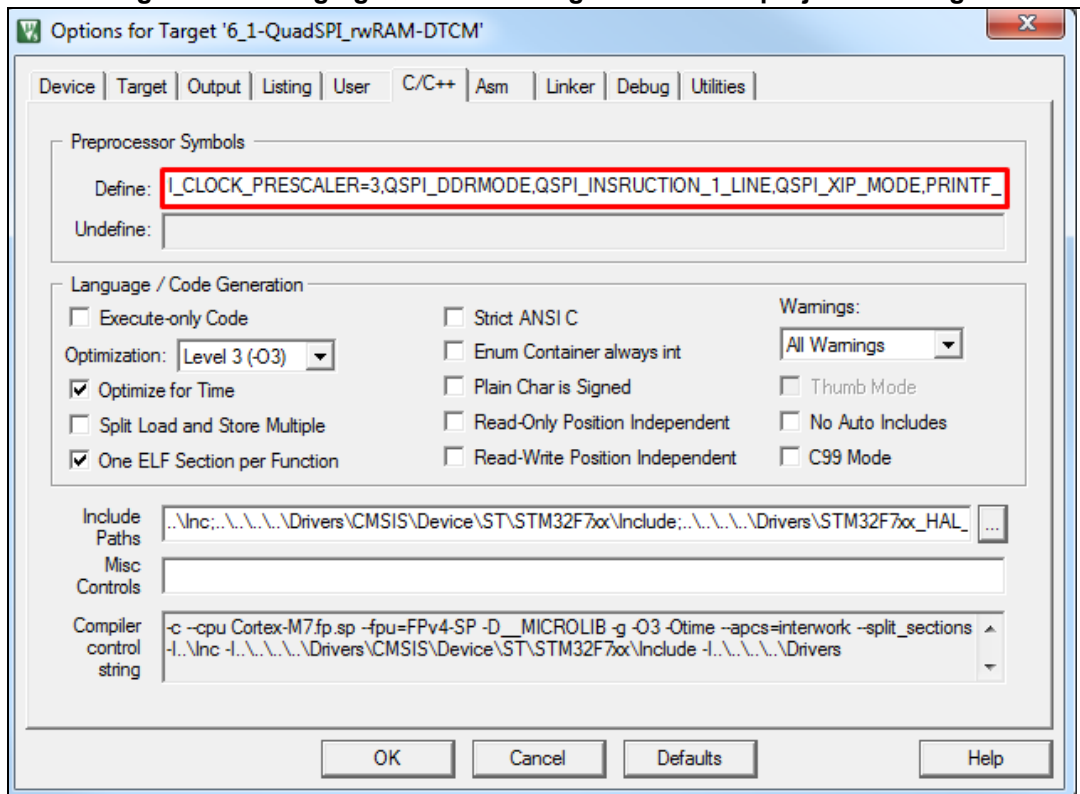
For both project configurations, 6_1-QuadSPI_rwRAM-DTCM and 6_2-QuadSPI_rwRAM-DTCM, the user can change the desired Quad-SPI settings in the Options for Target box as shown in Figure 44.

Note that the operating system clock during system initialization is 16 MHz, so at this moment the $QSPI_CLK = f_{AHB}/1 = 16\text{ MHz}$ (by default the prescaler = 0). Once the system initialization is done, the CPU jumps to the main function (in arm_fft_bin_example_f32.c file) where the system clock configuration is performed. The system clock is configured to run at 216 MHz.

Both project configurations have the following Quad-SPI settings:

- QSPI_CLOCK_PRESCALER = 3 and System clock is 216 MHz => QSPI_CLK = 54 MHz
- QSPI_DDRMODE => DDR mode enabled
- QSPI_INSTRUCTION_1_LINE => instruction is issued in one line
- QSPI_XIP_MODE => execute in place with SIOO enabled.

Figure 44. Changing Quad-SPI configuration in the project's settings



5.2.1 Configuring QSPI in memory-mapped mode during system initialization

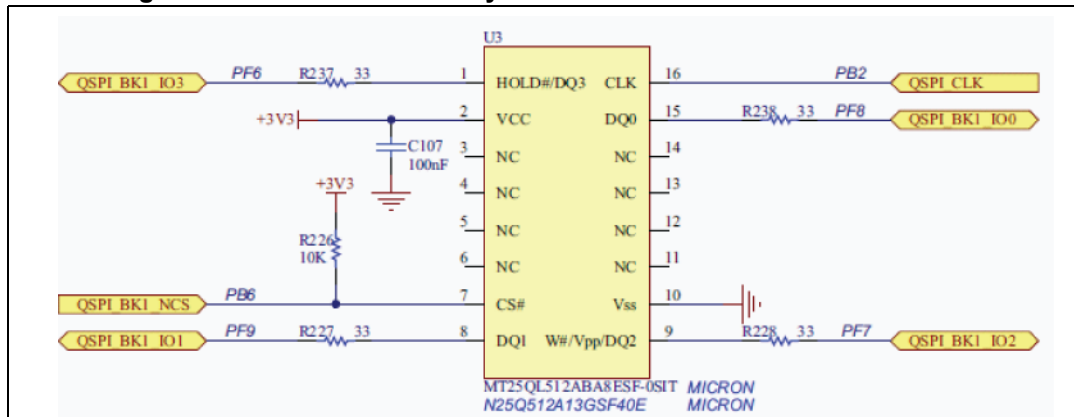
Boot from QSPI Flash memory is not supported, so the user can boot from internal Flash memory, configure the Quad-SPI peripheral in memory-mapped mode and then jump to execution from the external QSPI memory.

In this example the Quad-SPI's configuration is done during the system initialization in the `SystemInit_ExtMemCtl()` function in `system_stm32f7xx.c` file. All required Quad-SPI peripheral and memory configurations are done in `system-stm32f7xx.c` file and are described below.

GPIOs configuration

As shown in [Figure 45](#), the QSPI Flash memory is connected in Quad I/O mode, so six GPIOs have to be configured for the Quad-SPI interface.

Figure 45. QSPI Flash memory connection in STM32756-EVAL board



Following is the Quad-SPI interface GPIOs configuration in the `SystemInit_ExtmemCtl()` function

```
RCC->AHB1ENR |= 0x00000022; /* Enable GPIOB and GPIOF interface clock */
/* Connect PB2 and PB6 pins to Quad-SPI Alternate function */
GPIOB->AFR[0] = 0x0A000900;
GPIOB->AFR[1] = 0x00000000;
/* Configure PBx pins in Alternate function mode */
GPIOB->MODER |= 0x00002020;
/* Configure PBx pins speed to 100 MHz */
GPIOB->OSPEEDR |= 0x00003030;
/* Configure PBx pins Output type to push-pull */
GPIOB->OTYPER = 0x00000000;
/* No pull-up, pull-down for PBx pins */
GPIOB->PUPDR |= 0x00000000;

/* Connect PF6, PF7, PF8 and PF9 pins to Quad-SPI Alternate function */
GPIOF->AFR[0] |= 0x99000000;
GPIOF->AFR[1] |= 0x000000AA;
/* Configure PFx pins in Alternate function mode */
GPIOF->MODER |= 0x000AA000;
/* Configure PFx pins speed to 100 MHz */
GPIOF->OSPEEDR |= 0x000FF000;
/* Configure PFx pins Output type to push-pull */
GPIOF->OTYPER = 0x00000000;
/* No pull-up, no pull-down for PFx pins */
GPIOF->PUPDR = 0x00000000;
```


Enabling Quad-SPI peripheral

In order to configure the QSPI memory, the Quad-SPI peripheral should be enabled to communicate with the external memory.

```
/* Enable the QSPI interface clock */
RCC->AHB3ENR |= 0x00000002;
/* Reset QSPI peripheral */
RCC->AHB3RSTR |= (RCC_AHB3RSTR_QSPIRST); /* Reset */
RCC->AHB3RSTR &= ~(RCC_AHB3RSTR_QSPIRST); /* Release reset */
/* Enable Quad-SPI peripheral */
QUADSPI->CR = 0x00000001;
```

QSPI memory configuration

Once the Quad-SPI peripheral is enabled, it is possible to communicate with the QSPI memory in order to configure it in the desired operating mode. Referring to the MICRON N25Q512A datasheet, the SDR single SPI mode can be used to communicate with the memory, meaning that the command, address and data are sent in one line in order to configure the memory.

Note that the indirect Quad-SPI mode have to be used for external memory configuration.

Resetting the QSPI memory:

Before resetting the QSPI memory registers, the RESET ENABLE command 0x66 should be sent in 1-0-0 mode, so here only the command is sent in indirect mode while the address and the data phases are skipped.

```
/* Send RESET ENABLE command (0x66) to allow memory registers reset*/
QUADSPI->CCR = 0x00000166;
```

To reset the QSPI memory registers, the RESET command 0x99 should be issued in 1-0-0 mode, so here only the command is sent in indirect mode while the address and the data phases are skipped.

```
/* Send RESET command (0x99) to reset the memory registers*/
QUADSPI->CCR = 0x00000199;
```

Configure the memory to receive commands in four lines:

Note that in both project configurations, the QSPI memory is configured to receive commands in one line (QSPI_INSTRUCTION_1_LINE defined), but here we show how to configure the memory to receive the commands in four lines. If this is the desired mode, the user should use the QSPI_INSTRUCTION_4_LINES define instead of the QSPI_INSTRUCTION_1_LINE.

To enable the Quad-SPI operation, the enhanced volatile configuration register of the N25Q512A external memory should be configured with 0x7F.

To write 0x7F to the Enhanced Volatile Configuration Register, the 0x61 command should be sent. In that case, the Quad-SPI should send 0x61 command and 0x7F data while no address needs to be sent, then the used mode is 1-0-1.

```
/* Enable write cmd : 0x06. This to allow to write to enhanced volatile
register to allow instructions to be written in 4 lines*/
while(QUADSPI->SR & 0x20); /* Wait for busy flag to be cleared */
QUADSPI->CCR = 0x0106;
```

```

/* Write to Enhanced Volatile Configuration Register of the external memory
(MT25QL512): Enable quad I/O command input. Write to enhanced volatile
configuration register cmd = 0x61, Configuration: 0x7F*/
while(QUADSPI->SR & 0x20); /* Wait for busy flag to be cleared */
QUADSPI->CCR = 0x01000161;

```

```

while(!(QUADSPI->SR & 0x04)); /* Wait for FTF flag to be set */
QUADSPI->DR = 0x7F;
while(!(QUADSPI->SR & 0x02)); /* Wait for TCF flag to be set */

```

Enabling SIOO mode (named XIP mode in MICRON's datasheet):

```

/* Enable write cmd: 0x06. This is done to allow to write to volatile
configuration register. For more details refer to MT25QL512 datasheet. */
QUADSPI->CCR = ( 0x0106 | QSPI_CCR_IMODE );

```

```

while(QUADSPI->SR & 0x20); /* Wait for busy flag to be cleared */
/* Configure the Quad-SPI in 1-0-1 mode to write to VOLATILE CONFIGURATION
REGISTER*/
QUADSPI->CCR = (0x00000081 | QSPI_CCR_IMODE | QSPI_CCR_DMODE );

```

```

while(!(QUADSPI->SR & 0x04)); /* Wait for FTF flag to be set */
/* Write 0x83 to volatile configuration register: bit 3 = 0 to enable XIP,
and bits [7:4] = 8 to set eight dummy cycles*/
QUADSPI->DR = ((MEM_DUMMY_CYCLE_XIP << 4) | 0x3 );
while(!(QUADSPI->SR & 0x02)); /* Wait for TCF flag to be set */

```

Quad-SPI peripheral configuration

Configure Quad-SPI peripheral (QUADSPI_CCR register):

Once the QSPI memory is configured, the Quad-SPI interface should be configured in the memory-mapped mode; the frame format is set in QUADSPI_CCR register as described below:

- Use DTR QUAD INPUT/OUTPUT FAST READ command: INSTRUCTION [7:0] = 0xED
- Send command in one line: IMODE = 0b01
- Send address in four lines: ADMODE = 0b11
- Configure 3 bytes address: ADSIZE = 0b10
- Send alternate-byte in four lines: ABMODE = 0b11
- Configure 1 alternate-byte: ABSIZE = 0b00
- Receive data in 4 lines: DMODE = 0b11
- Configure 7 dummy cycles: DCYC = 0b00111
- Enable memory-mapped mode: FMODE = 0b11
- Enable SIOO mode: SIOO = 1
- Enable DDR mode: DDRM = 1

As SIOO mode is enabled (called XIP in MICRON datasheet), an alternate-byte have to be sent (QUADSPI_ABR = 0x00) at every new read sequence in order to keep the QSPI memory in SIOO mode.

According to MICRON's datasheet, a latency of eight dummy cycles before receiving data should be set when using 0xED command in Extended SPI mode. It depends also on the QSPI_CLK.

Note that only seven dummy cycles are configured for Quad-SPI peripheral, however it is eight cycles on the QSPI memory side; this is due to the additional alternate-byte cycle that is needed in order to send one byte in DDR mode.

After the address phase, we have in total a latency of eight cycles before the data phase: seven dummy cycles + one alternate-byte cycle.

See below the configuration code:

```
QUADSPI -> CCR = (0x0F002C00 | QSPI_CCR_DDRM | QSPI_CCR_DCYC |
FAST_READ_CMD | QSPI_CCR_IMODE | QSPI_CCR_SIOO | QSPI_CCR_ABMODE);
```

Configure Quad-SPI peripheral (QUADSPI_CR register)

The user can choose to operate either in DDR or in SDR mode, depending on the project configuration that by default is the QSPI_DDRMODE defined in both configurations. Since the DDR mode is enabled, the delayed sample shifting must be disabled.

The following code describes how to enable or disable the DDR mode and how to configure the prescaler and the QSPI memory size:

```
#ifdef QSPI_DDRMODE
QUADSPI->CR |= QSPI_CLK_PRESCALER; /* SSHIFT = 0 delayed sample shifting
disabled in DDR mode */
#else
QUADSPI->CR |= QSPI_CLK_PRESCALER | 0x10 ; /* 0x10: SSHIFT = 1 */
#endif
QUADSPI->DCR = 0x00190000; /* Memory size: 512 Mb (64MB): 2^(26-1) ->
2^(25) -> 2^(0x19) */
#endif
```

5.2.2 Placing application's code in external QSPI memory

The code to be loaded in the QSPI memory consists of calculation algorithms used to get the maximum energy bin in the frequency domain of an input signal using complex FFT, complex magnitude, and maximum functions.

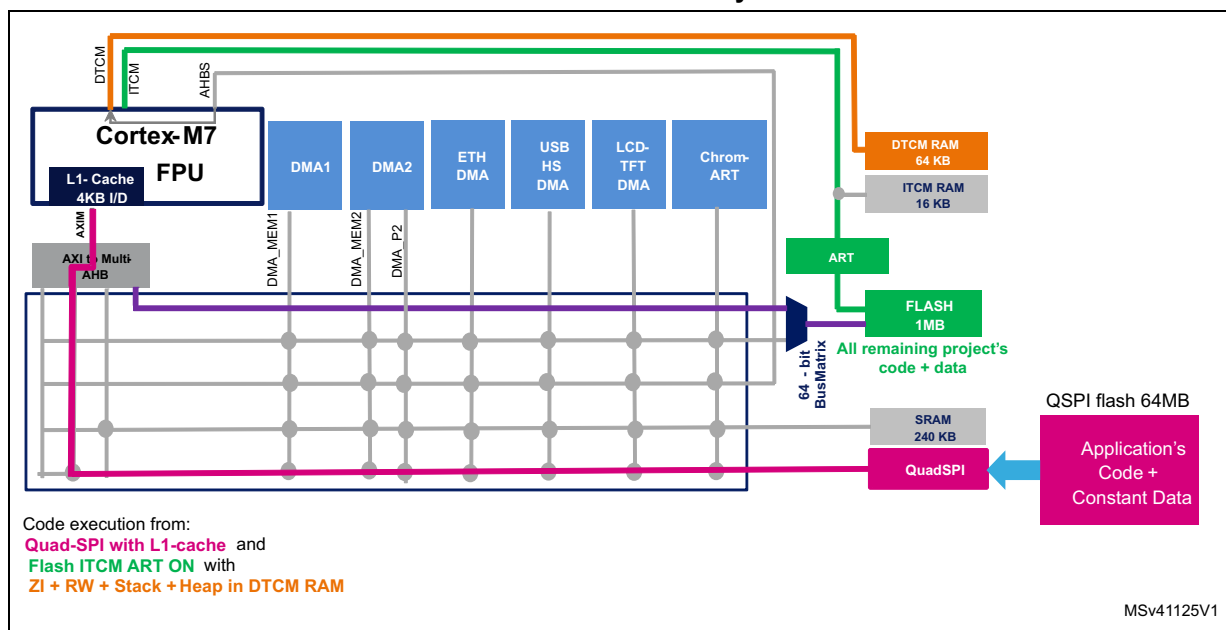
To place this code in the external QSPI memory a dedicated load region should be created in the linker file of the project. Two project configurations are available in the project: one where the code of the application and the constant data are both placed in the QSPI memory, and the other where the code of the application is placed in the QSPI memory where the constant data is placed in the ITCM Flash. For both project configurations the L1-DCache is enabled.

Code and constant data are all placed in QSPI memory: 6_1-Quad-SPI_rwRAM-DTCM

In this project configuration, the application code and its related constant data are both placed in the QSPI memory; so the Cortex®-M7 have to fetch them from the external memory.

All remaining project codes as the peripheral drivers and the vector tables are placed in the Flash memory ITCM. [Figure 46](#) describes the 6_1-Quad-SPI_rwRAM-DTCM project configuration.

Figure 46. 6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in QSPI memory



To place code and constant data in the QSPI memory a dedicated load region has to be created as shown in the following Keil MDK-ARM scatter file:

```
; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_IROM1 0x00200000 0x00100000 { ; load region size_region
    ER_IROM1 0x00200000 0x00100000 { ; load address = execution address
        *.o (RESET, +First)
        *(InRoot$$Sections)
        ; Place all remaining code and const data in Flash TCM.
        .ANY (+RO)
    }
}

LR_IROM2 0x90000000 0x00100000 { ; load region size_region
    ER_IROM2 0x90000000 0x00100000 { ;load address = execution address
        arm_fft_bin_example_f32.o (+RO-CODE)
        arm_bitreversal2.o (+RO-CODE)
```

```

    arm_cfft_f32.o (+RO-CODE)
    arm_cfft_radix8_f32.o (+RO-CODE)
    arm_cmplx_mag_f32.o (+RO-CODE)
    arm_max_f32.o (+RO-CODE)
    arm_fft_bin_example_f32.o (+RO-DATA)
    arm_common_tables.o (+RO-DATA)
    arm_const_structs.o (+RO-DATA)
}
RAM_RW_ZI 0x20000000 0x4000 {
    .ANY (+RW +ZI)
}
RAM_STACK 0x20004000 0x4000 {
    .ANY (STACK)
}
RAM_HEAP 0x20008000 0x8000 {
    .ANY (HEAP)
}
}

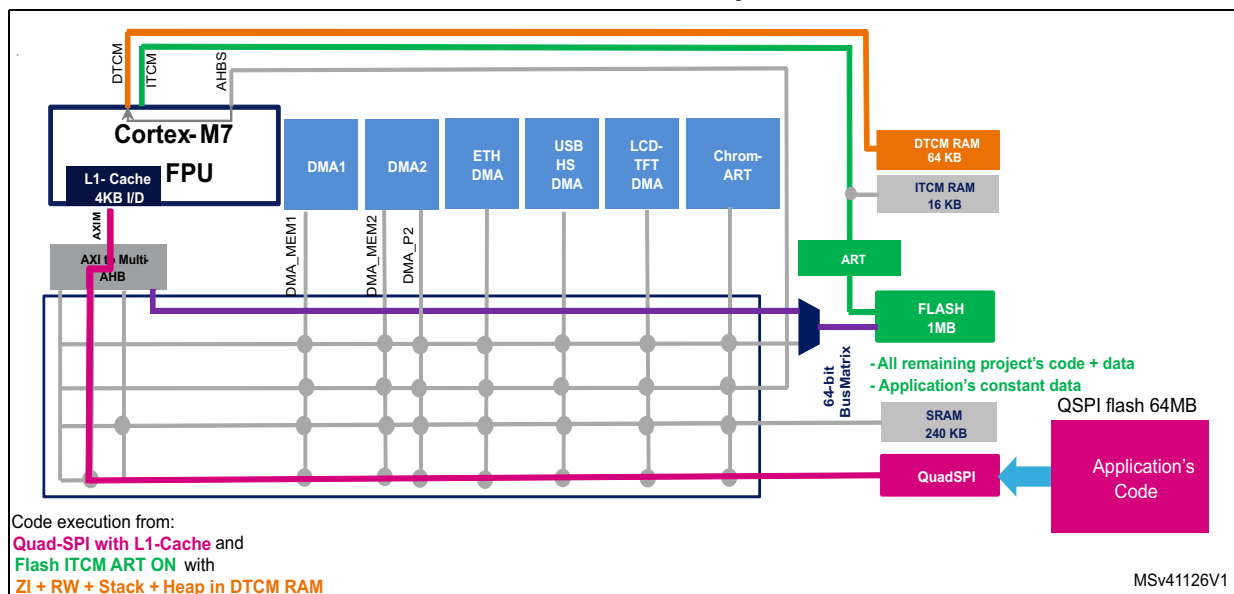
```

Code placed in QSPI memory while constant data in Flash memory ITCM: 6_2-Quad-SPI_rwRAM-DTCM

In this project configuration, the application code is placed in the QSPI memory while its related constant data is placed in the Flash ITCM. The Cortex®-M7 have to fetch code from the QSPI memory and data from the Flash ITCM.

All remaining project codes as the peripheral drivers and the vector tables are placed in the Flash memory ITCM. [Figure 47](#) describes the 6_2-Quad-SPI_rwRAM-DTCM project configuration.

Figure 47. 6_2-Quad-SPI_rwRAM-DTCM project configuration: only code in QSPI memory



To place code and constant data in the QSPI memory a dedicated load region has to be created as shown in the following Keil MDK-ARM scatter file:

```
; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_IROM1  0x00200000 0x00100000 {   ;load region size_region
    ER_IROM1  0x00200000 0x00100000 {   ;load address = execution address
*.o (RESET, +First)
    *(InRoot$$Sections)
    ; Place all remained code and const data in Flash TCM.
    .ANY (+RO)
    }
}

LR_IROM2  0x90000000 0x00100000 {   ;load region size_region
ER_IROM2  0x90000000 0x00100000 {   ;load address = execution address
    arm_fft_bin_example_f32.o (+RO-CODE)
    arm_bitreversal2.o (+RO-CODE)
    arm_cfft_f32.o (+RO-CODE)
    arm_cfft_radix8_f32.o (+RO-CODE)
    arm_cmplx_mag_f32.o (+RO-CODE)
    arm_max_f32.o (+RO-CODE)
    }
}

RAM_RW_ZI  0x20000000 0x4000 {
    .ANY (+RW +ZI)
    }

RAM_STACK  0x20004000 0x4000 {
    .ANY (STACK)
    }

RAM_HEAP   0x20008000 0x8000 {
    .ANY (HEAP)
    }
}
```

Performances analysis

The results are obtained with the STM32756G-EVAL, the CPU is running at 216 MHz, VDD=3.3 V and with seven-wait states access to the internal Flash memory. The Quad-SPI is configured in DDR 1-4-4 mode with SIOO enabled and QSPI_CLK = 54 MHz.

[Table 17](#) shows the obtained results for FFT demonstration for MDK-ARM in each configuration.

Table 17. Execution performances versus configuration

Feature configuration	Memory location configuration	CPU cycle number ⁽¹⁾
-	5-RAMITCM_rwRAM-DTCM	112428
I-cache + D-cache ON (Const data in QSPI memory)	6_1-Quad-SPI_rwRAM-DTCM	171056
I-cache + ART + ART-PF ON (Const data in Flash TCM)	6_2-Quad-SPI_rwRAM-DTCM	126900

1. The number of cycles may change from a version to another of the tool chain.

If the results of the case one and the case two of the “6-Quad SPI_rwRAM-DTCM” configuration are compared, we note that there is a significant difference in terms of performance since the demonstration uses a huge constant data.

- For the case one (6_1-Quad SPI_rwRAM-DTCM), since the read-only data and the instructions are both located in the QSPI Flash memory, a latency occurs due to the concurrency access of the instruction fetch and the read-only data on the Quad-SPI interface.
- For the case two (6_2-Quad SPI_rwRAM-DTCM), the read-only data and code are separated. The read-only data is located in Flash-TCM, therefore, the concurrency of the read-only data and the instruction fetch is avoided and the CPU can fetch the instruction from AXI while the data is loaded from TCM at the same time. This is the reason why the performance of the second case is clearly better than the first one.

If we compare the case 6_2-Quad-QPI_rwRAM-DTCM with the 5-RAMITCM_rwRAM-DTCM (which gives the best performances at 112428 CPU cycles as per AN4667 document), we can notice that it is very close in terms of performances.

This is an example of how it is important to benefit from the STM32F7x5/F7x6 smart architecture in order to improve the execution performances from the external QSPI memory. For more details on how to improve the execution performances from QSPI memory, refer to [Section 6.1 on page 79](#).

5.3 Storing (programming) data on the fly during a running application

Based on two examples from STM32Cube firmware, this section describes how to program QSPI Flash memory on the fly while running an application. The programming can be done with a software by writing directly to the QUADSPI_DR register or using DMA.

The STM32Cube firmware package provides two examples of reading and programming the QSPI memory:

- QSPI_ReadWrite_DMA: using DMA
- QSPI_ReadWrite_IT: using interrupts.

These examples are provided for STM32L476G-EVAL, STM32446E-EVAL, STM32469I-EVAL and STM32756G-EVAL boards and can be easily tailored to Discovery boards. In this section we will describe the STM32756G-EVAL programming examples.

Note: Since Flash memories need to be erased before writing; the user should perform an erasing operation before programming the Flash memory. Only the region to be programmed need to be erased; there is no need to perform a mass-chip erase operation if only one sector will be programmed.

5.3.1 Quad-SPI indirect write: programming QSPI memory using DMA

This section describes the STM32756G-EVAL example which is available on the STM32Cube_FW_F7 in the “Projects\STM32756G_EVAL\Examples\QSPI\QSPI_ReadWrite_DMA” directory.

This example shows how to program the QSPI memory with data from the internal SRAM. DMA2 is used to transfer the data from the internal SRAM to the Quad-SPI. The data to be written “aTxBuffer” is a buffer generated on the SRAM. It contains the string ******QSPI communication based on DMA******.

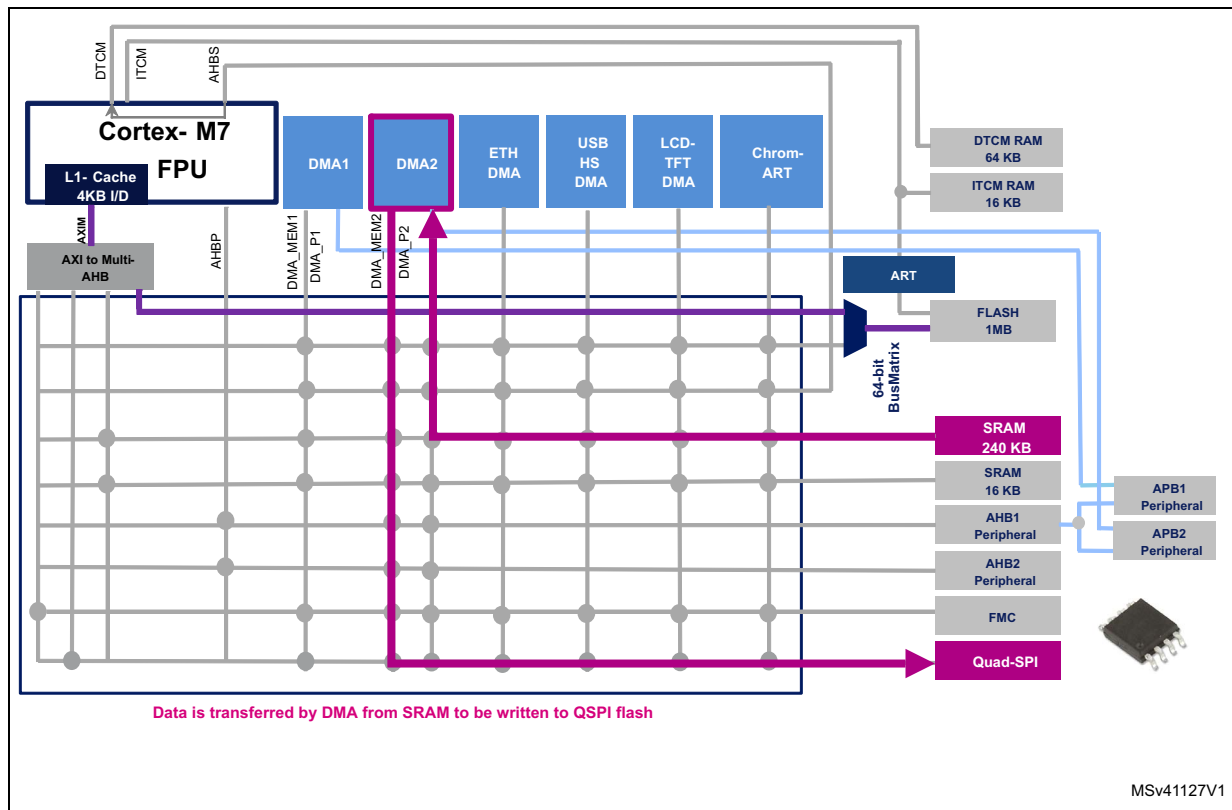
In this example, the following sequence is done in a forever loop:

1. Sector 1 of the QSPI memory is erased
2. Data is written to the memory in DMA mode
3. Data is read in DMA mode to be compared with the source
 - LED1 toggles each time a new comparison is good
 - LED3 is on as soon as a comparison error occurs
 - LED3 toggles as soon as an error is returned by HAL API.

In this section we focus on writing to the QSPI memory. Thanks to the STM32F7x5/Fx6 smart architecture, the DMA can be used to program or to read the QSPI Flash memory and then to offload the CPU. Once the DMA is configured and the transfer is started, no CPU intervention is needed. An interrupt can be generated once the transfer is completed.

As described in [Figure 48](#), the DMA reads the data “aTxBuffer” from the SRAM and writes it to the QSPI memory, in the meanwhile the CPU can execute code from the internal Flash.

Figure 48. Indirect write mode: programming QSPI memory using DMA



Quad-SPI GPIO and DMA configuration

The GPIOs and DMA2 are configured in the `HAL_QSPI_MspInit()` function in the `stm32f7xx_hal_msp.c` file. Note that the `HAL_QSPI_MspInit()` function is called in the `HAL_QSPI_Init()` function which include all the Quad-SPI required configurations.

The Quad-SPI global interrupt is enabled. The GPIOs configuration is done with respect to the QSPI memory connection in the STM32756G-EVAL board.

DMA2 is configured as follows:

- DMA2 Configuration: Stream 7 Channel 3 is enabled
- Memory Increment enabled
- DMA2 interrupt enabled.

Quad-SPI peripheral configuration

The Quad-SPI peripheral is configured in the HAL_QSPI_Init() function as described below:

- Clock Prescaler = 2 => QSPI_CLK = (HCLK / 3) = 72 MHz.
- FIFO threshold FTHRES = 0x03.
- Sample shifting delay is disabled.
- Flash Size is set in QUADSPI_DCR register: the memory size is 64Mbytes = $2^{[FSIZE+1]}$ = $2^{[19+1]}$ so FSIZE = 0x19.
- Chip-select high time (CSHT) is set to one cycle.
- Quad-SPI peripheral is enabled by setting EN bit in QUADSPI_CR register.

QSPI memory configuration

As previously mentioned, the write-enable command should be sent at first. This is done by calling QSPI_WriteEnable() function.

Start programming sequence

To start the programming sequence, the QUADSPI_CCR register is configured in the HAL_QSPI_Command() function as described below:

- Instruction: QUAD_IN_FAST_PROG_CMD (0x32)
- IMODE: one line
- ADMODE: one line
- ADSIZE: 24-Bits
- DMODE: four lines
- FMODE: indirect mode
- The number of bytes to be written is set in the QUADSPI_DLR register in the HAL_QSPI_Command() function.

The programming sequence is started in HAL_QSPI_Transmit_DMA() function, so the following configurations are done in this function:

- DMA Direction configuration: DMA_MEMORY_TO_PERIPH
- The number of bytes to be transferred in S7NDTR register (same as QUADSPI_DLR)
- The DMA transfer is Enabled by setting the DMAEN bit in the QUADSPI_CR register.

Note that the used address is 0x00000000 in the QUADSPI_AR register as writing will be performed in the first sector. In this way the programming sequence is immediately started once DMAEN bit is set.

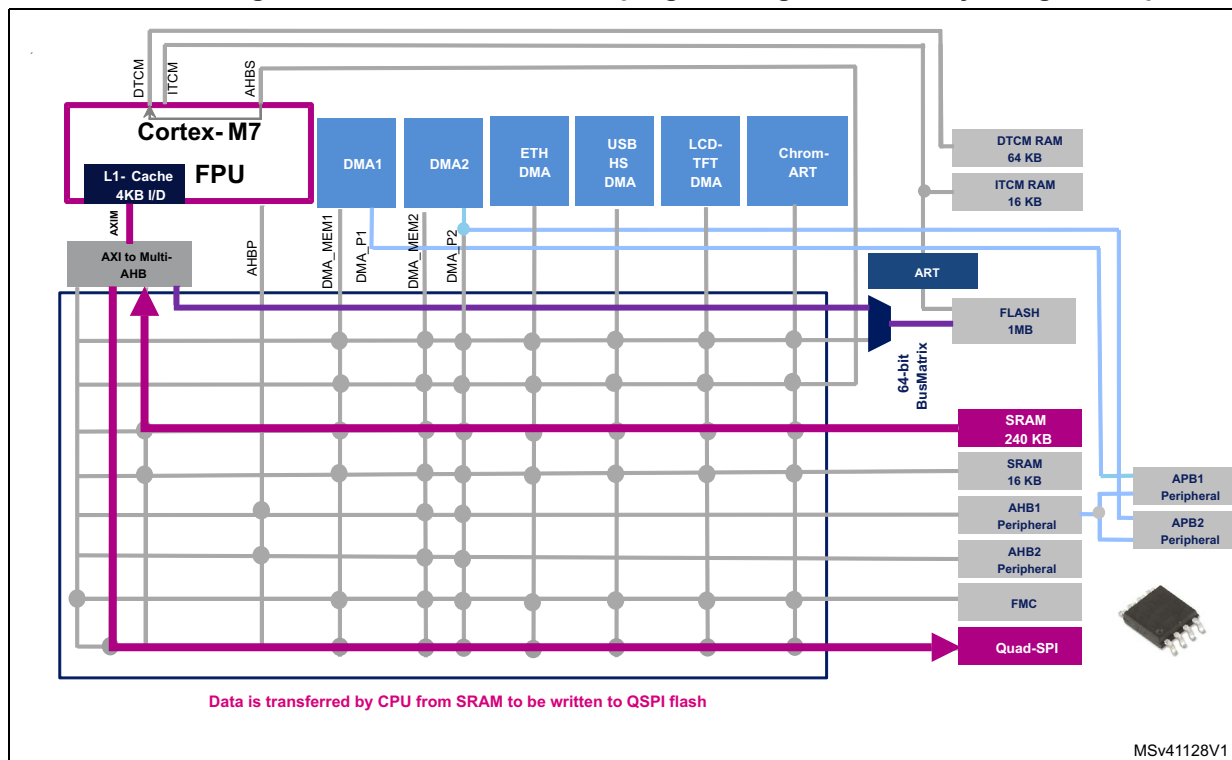
5.3.2 Quad-SPI indirect write: programming QSPI memory using interrupts

This example is available for all the STM32 embedding Quad-SPI interface which are shown in [Table 2 on page 8](#).

This section describes the STM32756G_EVAL example which is available on the STM32Cube_FW_F7 in the directory
Projects\STM32756G_EVAL\Examples\QSPI\QSPI_ReadWrite_IT.

This example shows how to program the QSPI memory with data from the internal SRAM using CPU and interrupts. The data to be written (aTxBuffer) is a buffer generated on the SRAM.

Figure 49. Indirect write mode: programming QSPI memory using interrupt



Quad-SPI GPIO configuration

The GPIOs and DMA2 are configured in the HAL_QSPI_MspInit() function in the stm32f7xx_hal_msp.c file. Note that the HAL_QSPI_MspInit() function is called in the HAL_QSPI_Init() function which include all Quad-SPI required configurations.

The Quad-SPI global interrupt is enabled. The GPIOs configuration is done with respect to the QSPI memory connection in the STM32756G-EVAL board.

Quad-SPI peripheral configuration

The Quad-SPI peripheral is configured in the HAL_QSPI_Init() function as described below:

- Configure Clock Prescaler = 2 => QSPI_CLK = (HCLK / 3) = 72 MHz.
- FIFO threshold FTHRES = 0x03 => Quad-SPI starts writing to QSPI Flash as soon as FTHRES is reached and FTF flag is set.
- Sample shifting delay is disabled.
- Flash Size is set in QUADSPI_DCR register: the memory size is 64Mbytes = $2^{[FSIZE+1]}$ = $2^{[19+1]}$ so FSIZE = 0x19.
- Chip-select high time (CSHT) is set to one cycle.
- Quad-SPI peripheral is enabled by setting EN bit in QUADSPI_CR register.

QSPI memory configuration

As previously mentioned, the write-enable command should be sent at first. This is done by calling QSPI_WriteEnable() function.

Start programming sequence

To start the programming sequence, the QUADSPI_CCR register is configured in the HAL_QSPI_Command() function as described below:

- Instruction: QUAD_IN_FAST_PROG_CMD (0x32)
- IMODE: one line
- ADMODE: one line
- ADSIZE: 24-Bits
- DMODE: four lines
- FMODE: indirect mode
- The number of bytes to be written is set in the QUADSPI_DLR register in the HAL_QSPI_Command() function.

The programming sequence is started in HAL_QSPI_Transmit_IT() function, so the following configurations are done in this function:

- Enable the Quad-SPI transfer error TEIE, FIFO threshold FTIE and transfer complete TCIE Interrupts

Note that the used address is 0x00000000 in the QUADSPI_AR register as writing will be performed in the first sector.

Once that the FTF flag is set, an interrupt is generated and the programming sequence is immediately started by writing to the FIFO in the HAL_QSPI_IRQHandler() function stm32f7xx_hal_qspi.c file.

5.4 Erasing-data example

This section describes how to erase the QSPI Flash memory. As previously mentioned, the indirect mode should be used for QSPI Flash memory erasing.

All the provided QSPI examples in the STM32Cube firmware package include an erasing operation example. For the STM32F7x5/F7x6 products EVAL board, the examples are available in the following STM32Cube directory:

STM32Cube_FW_F7_VX.X.X\Projects\STM32756G_EVAL\Examples\QSPI.

QSPI memory configuration

Before performing an erasing operation, a write-enable command have to be sent to the memory, this is done using the QSPI_WriteEnable() function.

Start erasing sequence

To start erasing sequence, the QUADSPI_CCR register is configured in the HAL_QSPI_Command_IT() function as described below:

- Instruction: SECTOR_ERASE_CMD (0xD8)
- IMODE: one line
- ADMODE: one line
- ADSIZE: 24-Bits
- DMODE: no data
- FMODE: indirect write mode
- QUADSPI_AR = 0x00000000.

After configuring the QUADSPI_CCR register, and since no data need to be sent, the erasing sequence is started immediately once the address of the first sector is provided.

5.5 Hardware implementation example

This section provides some hardware implementation examples of connecting the QSPI Flash memory to the STM32 based on the existing ST Discovery and Evaluation boards. [Table 18](#) summarizes different STM32 boards embedding QSPI Flash memory.

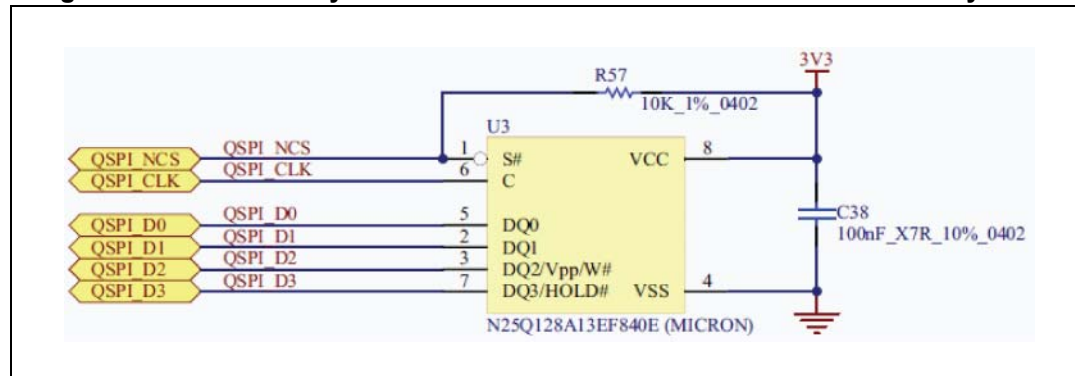
Table 18. Different STM32 boards embedding QSPI Flash memory

Product families	Board	QSPI Flash model	Size (Mbytes)
STM32L4x6	DISCO-L476VG	N25Q128A13EF840E	16
	STM32L476G-EVAL	N25Q256A13EF840E	32
STM32F446	STM32446E-EVAL	N25Q256A13EF840E	32
STM32F469/F479	STM32F469I-DISCO	N25Q128A13EF840E	16
	STM32469I-EVAL	MT25QL512ABA8ESF-0SIT	64
		N25Q512A13GSF40E	
STM32F7x5/F7x6	STM32F746G-DISCO	N25Q128A13EF840E	16
	STM32746G-EVAL	MT25QL512ABA8ESF-0SIT	64
	STM32756G-EVAL	N25Q512A13GSF40E	

STM32F746G-DISCO discovery board

Figure 50 shows an example of a connected MICRON QSPI Flash memory in Quad I/O mode on the STM32F746G-DISCO discovery board.

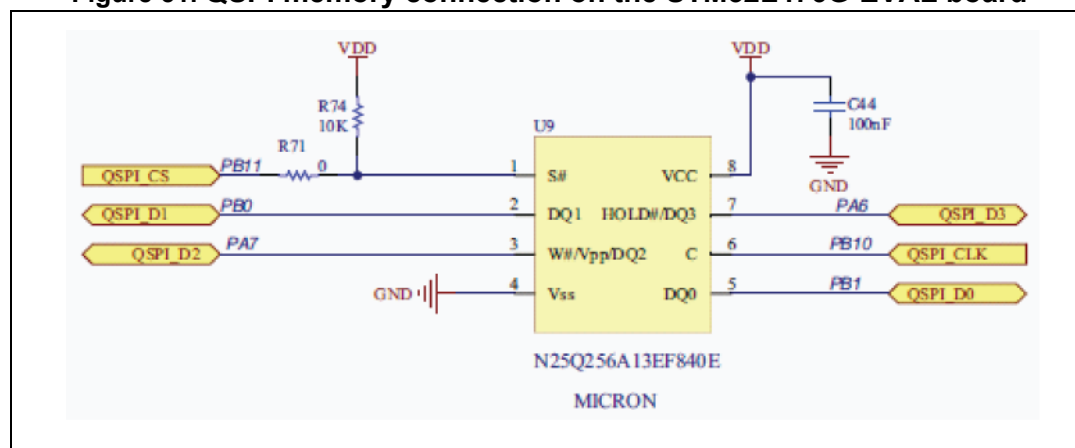
Figure 50. QSPI memory connection on the STM32F746G-DISCO discovery board



STM32L476G-EVAL board

Figure 51 shows an example of how to connect MICRON QSPI Flash memory in Quad I/O mode on the STM32L476G-EVAL discovery board.

Figure 51. QSPI memory connection on the STM32L476G-EVAL board



6 Performance and power

This chapter discusses how to get the best performances and how to decrease the power consumption.

6.1 How to get the best performances

6.1.1 Write performance

Since the writing speed of the QSPI Flash memories is considerably low, there is no considerable benefit when optimizing the transmission's speed. It is still beneficial to use burst (page programming) writing to reduce the command's overhead. In addition, DMA can be used to offload the CPU.

6.1.2 Read performance

This section describes how to get the optimum reading performances using Quad-SPI peripheral features and following the described recommendations.

Configure Quad-SPI at maximum speed

One of the most important parameters allowing to boost the read performances is the Quad-SPI clock that should be as high as possible. As previously mentioned, (see [Section 3.3.2 on page 41](#), the maximum reachable Quad-SPI operating speed depends mainly on the PCB design quality so the user should optimize the PCB design.

For instance, if the user's hardware allows the Quad-SPI to operate at 60MHz in DDR Quad I/O mode, then for a sequential access case, a 4 KB image can be read at almost 60 MByte/s. In that case the total transfer time will be 4107 cycles: eight cycles for command + three cycles for the address (24-Bits in DDR mode) + 4096 cycles for the 4 KB image.

Another important parameter to be considered by the user when selecting the QSPI memory device, is the maximal clock frequency supported by the QSPI memory in SDR and DDR modes.

Use DDR mode (DTR)

Use the DDR mode to double the throughput, for instance in SDR Quad I/O mode, one byte is read every two clock cycles while in DDR mode one byte is read every clock cycle.

Another interesting usage for the DDR mode is that it can be a very good alternative in low-power applications requiring to operate at a low system clock in order to save power where it is not possible that the Quad-SPI operates at its maximum speed. In that case the user can use the DDR mode to double reading speed but at the same operating Quad-SPI clock.

Note: The user should make sure that the used read command does support the DDR mode.

Use Quad I/O mode

Using Quad I/O mode allows to boost reading performances, so the user should use the Quad I/O mode rather than the single or dual I/O mode. It is recommended to use the Quad I/O mode for all the phases: command, address, alternate-byte and data.

Note that sending the command in four lines is supported by some memory devices such as Micron or Spansion.

Use dual-flash mode

Using dual-Flash mode requires adding only four additional GPIOs (IO4 ... IO7) and allows doubling the throughput, for example in SDR Quad I/O dual-Flash mode one byte is read each QSPI_CLK cycle, while in DDR Quad I/O dual-flash mode two bytes are read each QSPI_CLK cycle.

Reduce command overhead

Each access to QSPI memory needs a command and an address to be sent which leads to command overhead. In order to reduce command overhead and boost the read performance, the user should use the following recommendations:

- Use large burst transfers

Since each access to QSPI memory needs to send a command and an address, it will be beneficial to perform large burst transfers rather than small repetitive transfers allowing to reduce command overhead.

- Sequential access

The best read performance is achieved if the stored data is read out sequentially, which avoids command and address overhead and then leads to reach the maximum performances at the operating Quad-SPI clock speed.

In general it is the case where QSPI memory is used for storage applications such as graphics or multimedia contents (see examples in [Section 5.1: Reading data from QSPI memory: graphical application on page 56](#).)

- Consider Timeout counter

The user should consider that enabling timeout counter in memory-mapped mode may increase command overhead. When timeout occurs, the Quad-SPI rises chip-select. After that, to read from the QSPI memory a new read sequence needs to be initiated; it means that the read command should be issued again, which leads to command overhead (see [Section 2.4.3: Timeout counter on page 29](#)).

Note that timeout counter allows decreasing power consumption (see [Section 6.2.1: Use timeout counter on page 82](#)), but if the performance is a concern, the user can increase the timeout period in the QUADSPI_LPTR register or even disable it.

If the power consumption is also a concern, the user can enable the SIOO feature without the need to disable the timeout counter.

- Use SIOO feature (Continuous Read Mode) for random and non-sequential accesses

For random and non-sequential accesses, the command overhead increases. As described in [Figure 12](#), a command and an address are sent to the memory every new read sequence. In that case, the user should enable the SIOO feature in order to reduce the command overhead (see [Section 2.4.1: Send instruction only-once \(SIOO\) on page 28](#)).

Note: Not all the read commands support the Continuous Read Mode (Enhance Performance Mode) so the user should consider this information when selecting the read command.

Execution performance

To improve the execution performance from the QSPI Flash, the user should follow the previously described read performance recommendations.

Executing from the QSPI memory is generally characterized by its random and non-sequential accesses. As already mentioned, an important recommendation to boost execution performance is enabling the SIOO feature.

As seen in [Figure 46: 6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in QSPI memory on page 68](#), placing both the code and the read-only data in the QSPI memory leads to concurrency on the Quad-SPI interface during execution.

In order to avoid this concurrency, the user can separate the read-only data and the code. For example, the read-only data can be located in the QSPI memory while the code can be located in the Flash-TCM. This allows to avoid the concurrency of the read-only data and the instruction fetch; therefore, the CPU can fetch the instructions from Flash-TCM while the data is loaded from the QSPI memory at the same time.

When the application contains huge constants data, the user can separate constants and code, each one in a dedicated section. If the code section size fit the internal Flash memory size, the code can be loaded in the internal Flash memory while the constants are loaded in the QSPI memory.

For the STM32F7x5/F7x6, it is recommended to enable the Cortex®-M7's L1-Cache.

General recommendations

- Use DMA for data transfers in order to offload the CPU.
- Use flag-status polling mode rather than software flag checking.
- Use memory-mapped mode to allow to any AHB master to access the QSPI memory without CPU intervention.

6.2 Decreasing power consumption

One of the most important requirements in wearable and mobile applications is the power consumption. Some recommendations can be followed in order to decrease the power consumption.

To decrease the total application's power-consumption, the user usually puts the STM32 in low-power mode. To reduce even more the current consumption, the connected QSPI memory also can be put in low-power mode (also known as deep power-down mode).

For most QSPI Flash memory devices the default mode after the powering-up sequence is the standby mode. In standby mode, there is no ongoing operation; the nCS is high but current consumption can be reduced even more. The DPD mode allows reducing the power consumption.

6.2.1 Use timeout counter

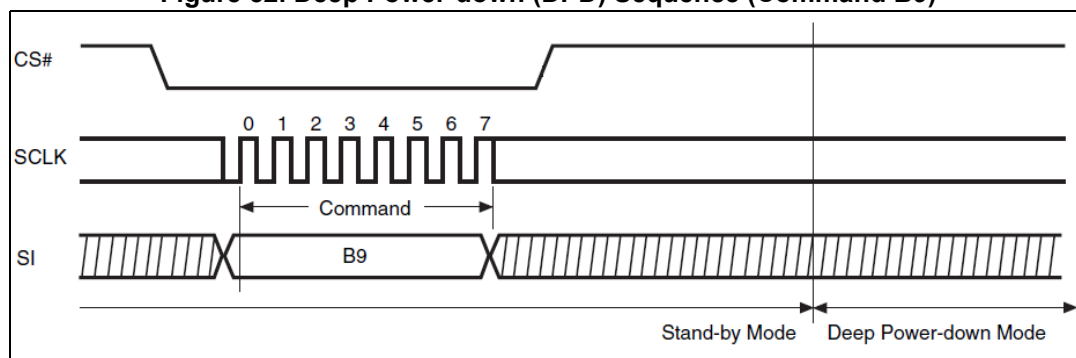
The timeout nCS feature can be used to avoid any extra power-consumption in the external Flash memory. When the clock is stopped for a long time, the timeout counter can release the nCS pin to put the external Flash memory in a lower-consumption state after a period of timeout elapsed without any access (see [Section 2.4.3 on page 29](#)).

6.2.2 Put the QSPI memory in deep power-down mode

The deep power-down mode requires to enter the deep-power instruction. During the deep power-down mode, the device is not active and all of the write, program and erase instructions are ignored. When CS# goes high, it is only in standby mode and not in deep power-down mode. Deep power-down mode is different from standby mode.

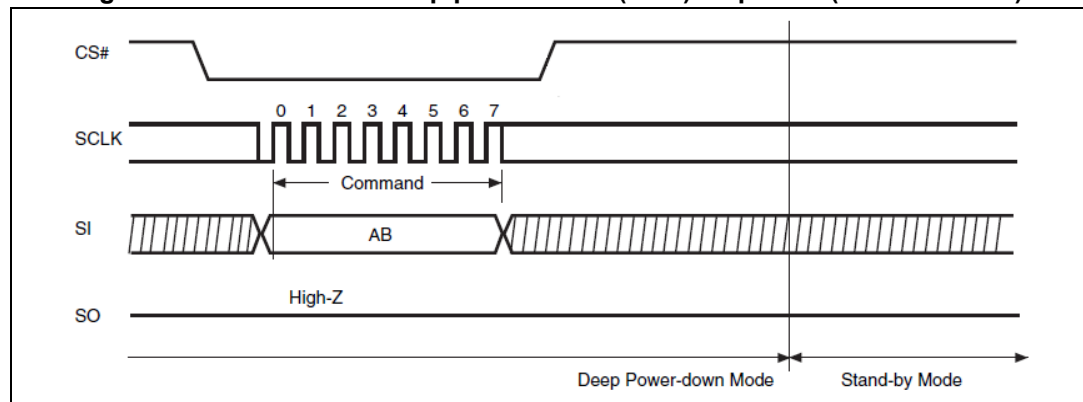
Before entering a low-power mode or when the QSPI memory is not used, it can be put in DPD mode in order to reduce the overall application's power-consumption. For several memory brands the command 0xB9 should be sent to the external serial-Flash memory to enter the deep power-down mode. [Figure 52](#) shows the DPD sequence:

Figure 52. Deep Power-down (DPD) Sequence (Command B9)



To exit DPD mode, the RELEASE FROM DEEP POWERDOWN command (0xAB) should be sent. [Figure 53](#) shows the RPD sequence:

Figure 53. Release from deep power-down (RDP) sequence (Command AB)



Note: *Not all the serial Flash memories support the deep power-down mode. If the selected external serial memory does not support the deep power-down mode, the STM32 may control an external-power switch through a GPIO to remove the power supply of the external QSPI Flash memory and to cancel its current consumption.*

6.2.3 Quad-SPI Flash memories supporting DPD mode

Many QSPI memory brands support the DPD mode, here below an example:

- WINBOND: W25Q64CV
- Spansion: S25FL032P
- MICRON: MT25QL512AB
- Macronix: MX25L12865E.

7 Supported devices

The STM32's Quad-SPI interface has a very flexible frame format that allows to:

- Send up to five phases: instruction – address – alternate byte – dummy – data
- Skip any phase
- Send each phase in one, two or four lines
- Send address in one, two, three or four byte
- Send one, two, three or four alternate-byte
- Send up to 31 dummy clock cycles.

In addition, STM32's Quad-SPI interface allows to send any command, so the user can program the desired command in the QUADSPI_CCR register in the INSTRUCTION[7:0] field.

The STM32's Quad-SPI interface is fully configurable in terms of frame format and hardware allowing to support any QSPI memory in the market.

There are several suppliers of Quad-SPI compatible memories, such as Winbond, Spansion, Macronix, MICRON (Numonyx), Microchip (SST) and others.

8 Conclusion

The STM32 MCUs provide a very flexible and useful Quad-SPI interface, allowing to fit all the memory hungry applications at a lower development cost. It also avoids the complexity of design with external parallel Flash memories by reducing the pin count and offering better performances. This application, note demonstrates the STM32's excellent Quad-SPI interface performances and its flexibility, allowing lower development costs and faster time to market.

9 **Revision history**

Table 19. Document revision history

Date	Revision	Changes
01-Apr-2016	1	Initial release.



IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved