Introduction to Quad-SPI interface for STM32 MCUs and MPUs

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 place extra demands on the often limited micocontroller (MCU) and microprocessor (MPU) on-chip memory.

STM32 MCUs and MPUs are referred to as "STM32 devices" in this document. The devices that are concerned are listed in the table below.

Table 1. **Applicable products**

Type	Products, lines and series
Microcontrollers	STM32F7 series, STM32L4 series
	STM32F412 line, STM32F413/423 line, STM32F446 line, STM32F469/479 line STM32H743/753 line, STM32H750 value line STM32L4R5/S5 line, STM32L4R7/S7 line, STM32L4R9/S9 line
	STM32G473CB, STM32G473CC, STM32G473CE, STM32G473MB, STM32G473MC, STM32G473ME, STM32G473PB, STM32G473PC, STM32G473PE, STM32G473QB, STM32G473QC, STM32G473QE, STM32G473RB, STM32G473RC, STM32G473VB, STM32G473VC, STM32G473VE STM32G483CE, STM32G483ME, STM32G483PE, STM32G483QE, STM32G483RE, STM32G483VE STM32G474CB, STM32G474CC, STM32G474CE, STM32G474MB, STM32G474MC, STM32G474ME, STM32G474PB, STM32G474PC, STM32G474PE, STM32G474QB, STM32G474QC, STM32G474QE, STM32G474RB, STM32G474RC, STM32G474RE, STM32G474VB, STM32G474VC, STM32G474VE STM32G484CE, STM32G484ME, STM32G484PB, STM32G484PE, STM32G484QE, STM32G484RE, STM32G484VE STM32G491CC, STM32G491CE, STM32G491KC, STM32G491KE, STM32G491MC, STM32G491ME, STM32G491NE, STM32G491RC, STM32G491RE, STM32G491VC, STM32G491VE STM32G4A1CE, STM32G4A1KE, STM32G4A1MC, STM32G4A1ME, STM32G4A1RE, STM32G4A1VE STM32WB55CC, STM32WB55CE, STM32WB55CG, STM32WB55RC, STM32WB55RE, STM32WB55RG, STM32WB55VC, STM32WB55VE, STM32WB55VG, STM32WB35CC, STM32WB35CE STM32WBA23CE, STM32WBA23KE, STM32WBA25CE, STM32WBA25HE
Microprocessors	STM32MP151, STM32MP153, STM32MP157 lines

External parallel memories are used to extend the STM32 devices on-chip memory and solve the memory size limitation. Usually this action represents an increase in the pin count and implies a more complex design.

To face these requirements, STM32 devices embed an external memory interface named Quad-SPI (see more details on Table 2). This interface allows the connection of external compact-footprint Quad-SPI high-speed memories. This Quad-SPI interface is used for data storage such as images, icons, or for code execution.

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

For more detailed information about the products listed in the table below, refer to the corresponding datasheets and reference manuals available from the STMicroelectronics web site http://www.st.com.

1 General information

This document applies to Arm®-based microcontrollers and microprocessors.

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

2 Overview

The Quad-SPI is a serial interface that allows the communication on four data lines between a host (STM32) and an external Quad-SPI memory. The QUADSPI supports the traditional SPI (serial peripheral interface) as well as the Dual-SPI mode which allows to communicate on two lines. QUADSPI 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 the STM32 devices to fit memory-hungry applications, to simplify PCB (printed circuit board) designs and to reduce costs.

2.1 QUADSPI availability and features across STM32 families

All STM32 devices shown in the table below have mainly the same QUADSPI features.

Table 2. **QUADSPI availability and features across STM32 families**

Products	Maximum speed (MHz)(1)		Dual-Flash	FIFO size (bytes)	Maximum addressable space(2)	
	SDR	DDR			Memory mapped	Indirect
STM32F412 line	100	80	Yes	32	256 Mbytes	4 Gbytes
STM32F413/423 line(3)						
STM32F446 line(4)	90	60				
STM32F469/479 line		80				
STM32F730xx devices	108					
STM32F7x2 line(4)						
STM32F750xx						
STM32F7x3						
STM32F7x5						
STM32F7x6						
STM32F7x7						
STM32F7x8						
STM32F7x9						
STM32G473	110	70		16		
STM32G474						
STM32G483						
STM32G484						
STM32G491						
STM32G4A1(5)						
STM32H743/753	133	100				
STM32H750 Value line						
STM32L471xx	60	48	No			
STM32L412xx						
STM32L422xx						
STM32L432xx						
STM32L442xx						
STM32L475xx						
STM32L476xx						
STM32L486xx						
STM32L431xx			Yes			
STM32L451xx						

Products	Maximum speed (MHz)[1]		Dual-Flash	FIFO size (bytes)	Maximum addressable space[2]	
	SDR	DDR			Memory mapped	Indirect
STM32L452xx STM32L462xx STM32L4x3[6]	60	48	Yes	16	256 Mbytes	4 Gbytes
STM32L496xx STM32L4A6xx	60	48	Yes	16	256 Mbytes	4 Gbytes
STM32WB35xx STM32WB55xx	60	50	No	32	256 Mbytes	4 Gbytes
STM32L4R5/S5 STM32L4R7/S7 STM32L4R9/S9[7]	86	60	Yes	32	256 Mbytes	4 Gbytes
STM32MP1	166	90	Yes	32	256 Mbytes	4 Gbytes
STM32WBA2xxx	64	32	No	32	256 Mbytes	4 Gbytes

1. *Maximum QUADSPI speed from datasheet. For more details on the QUADSPI maximum speed refer to the relevant device datasheet.*
2. *32-bits address mode should be used to reach 256 Mbytes in Memory-mapped mode and 4 Gbytes in Indirect mode.*
3. *UFQFPN48 does not support Quad-SPI.*
4. *LQFP64 supports only Bank1 and Single-SPI/Dual-SPI only.*
5. *UFQFPN48 and LQFP48 do not support Dual-Flash mode.*
6. *For this set of products, Dual-Flash mode is supported only with LQFP100 and UFBGA100 packages.*
7. *This set of products contains two Octo-SPI interfaces, each one of them can connect one or two Quad-SPI memories with Single-Flash or Dual-Flash modes.*

2.2 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 permits the connection of a lower cost external Flash memory to small packages, reducing the PCB area, simplifying the PCB design and reducing the 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, amongst which eight lines are for data.

2.2.1 Main benefits of STM32 embedded Quad-SPI interface

The table below 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 Quad-SPI memories. Uses six pins in Quad-SPI mode and four pins for Single or Dual-SPI modes. 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 Quad-SPI memories.
Save cost	Easier and faster design permits 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 Quad-SPI memory to be seen as an internal memory. Allows code execution (XIP mode) from Quad-SPI Flash memory. Supports SIOO mode also named Continuous-read mode by some memory manufacturers (see Section 3.4.1: Send instruction only-once (SIOO)) for higher execution performance.
Extended size for data storage	Memory-mapped mode allows Quad-SPI memory to be accessed autonomously by any AHB (advanced high-performance) or AXI (advanced extensible interface protocol) master. 32-bits address mode enables the possibility to address up to 4-Gbyte Quad-SPI memory size. Dual-Flash mode enables the use of two Quad-SPI 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 mode doubles throughput. Perfect for graphical applications.
Multiple memory solutions	There are volatile Quad-SPI SRAM (static random-access memory) available from Microchip, ON Semiconductor and others. Available non-volatile Quad-SPI Flash memories. NOR, NAND.
Supports any Quad-SPI memories available in the market	Its fully configurable and flexible frame format permits to support almost all Quad-SPI devices available on the market.
Growing amount of manufacturers	Spansion, Winbond, Micron, Macronix, ONSemiconductors, Cypress, APmemory and ISSI among others. Huge investment on higher densities Quad-SPI Flash memories such as NAND.

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

2.3 QUADSPI in a smart architecture

The Quad-SPI interface 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 QUADSPI is integrated in a smart architecture which allows the following features:

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

2.3.1 System architecture: STM32L4 Series

The STM32L4 Series system architecture consists mainly of a 32-bit multilayer AHB bus matrix that interconnects multiple masters to multiple slaves.

The QUADSPI can be accessed by relevant masters like the Arm® Cortex®-M4 either through S-Bus or through I-bus and D-bus when remap is enabled. QUADSPI is also accessible by DMA1 and DMA2.

Enabling physical remap over I-bus and D-bus boosts execution performances for the Cortex®-M4.

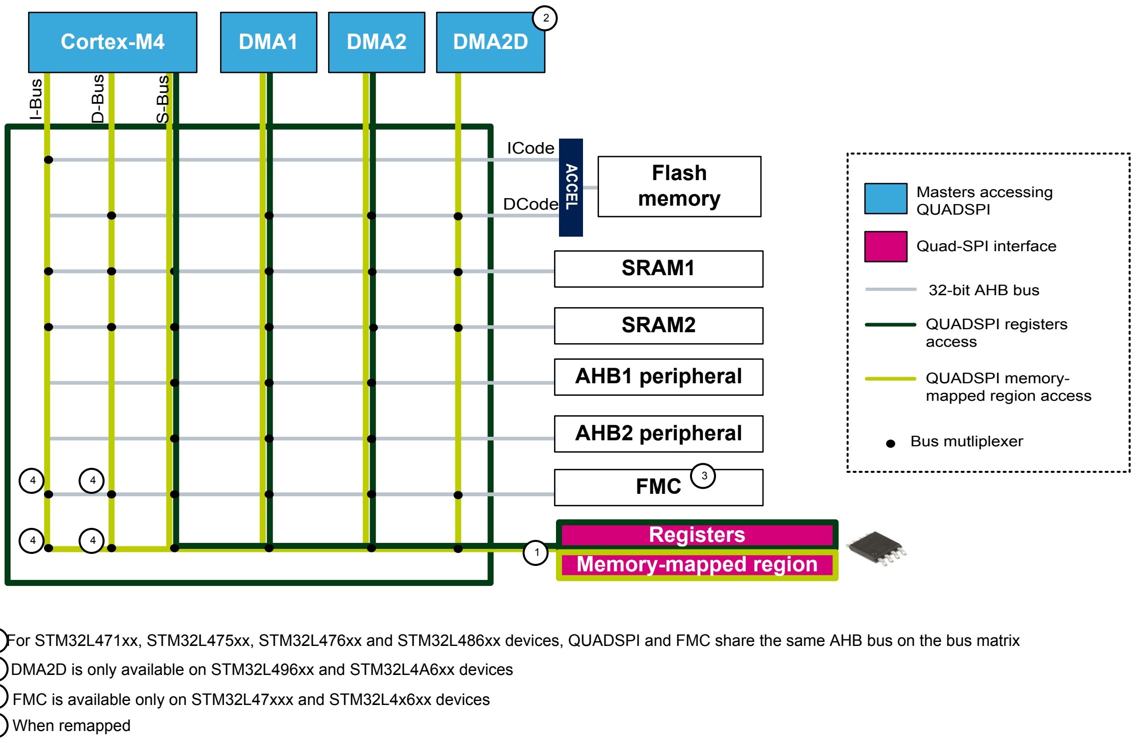
The access to the QUADSPI can be either a registers access or a memory-mapped region access:

- The registers access can be done by the Cortex®-M4 for registers configuration or data transfer. The register access can be done also by the DMA1 and DMA2 for data transfer.
- The memory-mapped region access can be done by the Cortex®-M4 for code and data fetch. The memory-mapped region can also be accessed by the DMA1, DMA2 and DMA2D for data transfer.

The figure below shows a QUADSPI interconnection in the STM32L4 Series system.

Note: DMA2D is available only in STM32L496xx and STM32L4A6xx devices.

Figure 1. System architecture: STM32L4 Series


Cortex-M4
DMA1
DMA2
DMA2D
2
I-Bus
D-Bus
S-Bus
ICode
DCode
ACCEL
Flash memory
SRAM1
SRAM2
AHB1 peripheral
AHB2 peripheral
FMC
3
4
4
4
4
1
Registers
Memory-mapped region
Masters accessing QUADSPI
Quad-SPI interface
32-bit AHB bus
QUADSPI registers access
QUADSPI memory-mapped region access
Bus mutliplexer
1 For STM32L471xx, STM32L475xx, STM32L476xx and STM32L486xx devices, QUADSPI and FMC share the same AHB bus on the bus matrix
2 DMA2D is only available on STM32L496xx and STM32L4A6xx devices
3 FMC is available only on STM32L47xxx and STM32L4x6xx devices
4 When remapped

2.3.2 System architecture: STM32F4 Series

The STM32F4 Series system architecture consists mainly of a 32-bit multilayer AHB bus matrix that interconnects multiple masters to multiple slaves (refer to cover page for detail on applicable products).

The external Quad-SPI memory can be accessed by the Cortex®-M4 through the S-bus. The QUADSPI is also accessible by all the masters on the AHB bus matrix such as DMA1, DMA2, USB OTG HS, MAC Ethernet, LTDC and DMA2D. This accessibility enables an efficient data transfer (like images for graphical applications).

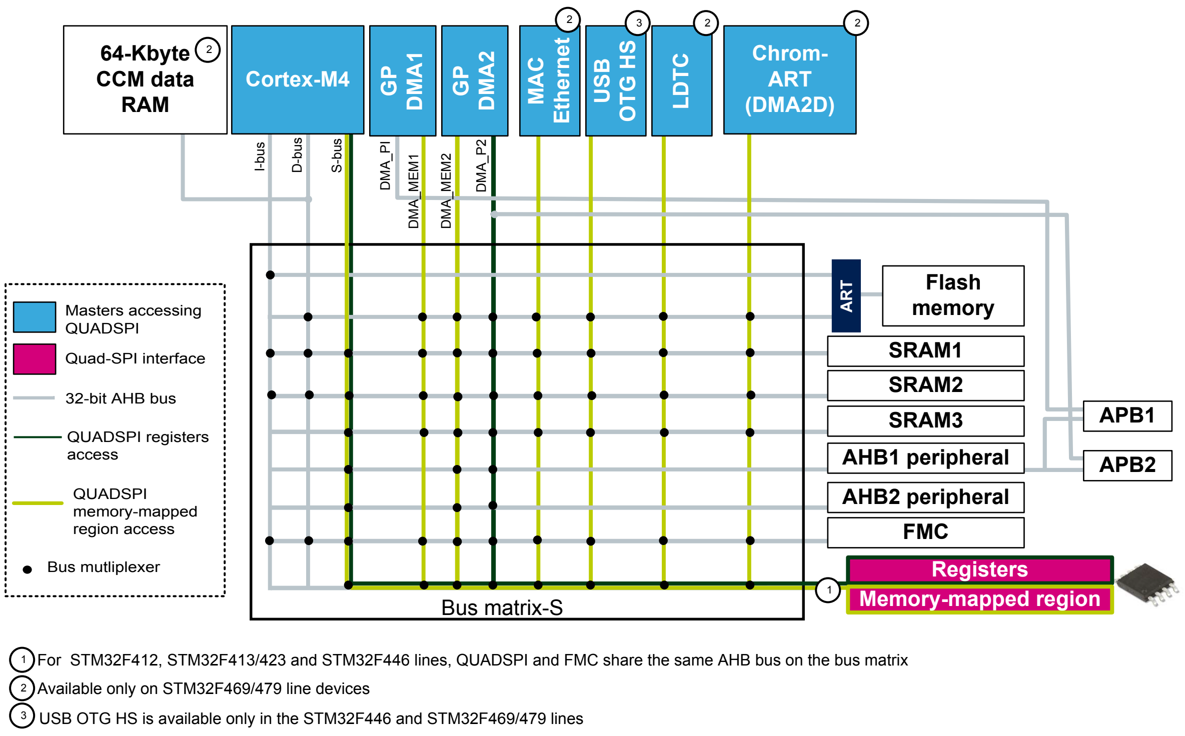
The access to the QUADSPI can be either a registers access or a memory-mapped region access:

- The registers access can be done by the Cortex®-M4 through S-Bus for registers configuration and data transfer. The register access can be done also GP DMA2 for data transfer.
- The memory-mapped region access can be done by the Cortex®-M4 through S-Bus for code and data fetch. The memory-mapped region access can be done also by the GP DMA1, GP DMA2, MAC Ethernet, USB OTG HS, LTDC and DMA2D for data transfer.

The figure below shows a QUADSPI interconnection in the STM32F4 Series system.

Note: For MAC Ethernet, USB OTG HS, LTDC and DMA2D refer to the applicable product.

Figure 2. System architecture: STM32F4 Series


64-Kbyte CCM data RAM (2)
Cortex-M4
GP DMA1
GP DMA2
MAC Ethernet (2)
USB OTG HS (3)
LDTC (2)
Chrom-ART (DMA2D) (2)
I-bus
D-bus
S-bus
DMA_PI
DMA_MEM1
DMA_MEM2
DMA_P2
Masters accessing QUADSPI
Quad-SPI interface
32-bit AHB bus
QUADSPI registers access
QUADSPI memory-mapped region access
Bus mutliplexer
ART
Flash memory
SRAM1
SRAM2
SRAM3
AHB1 peripheral
AHB2 peripheral
FMC
APB1
APB2
Registers
Memory-mapped region
Bus matrix-S
(1)
(1) For STM32F412, STM32F413/423 and STM32F446 lines, QUADSPI and FMC share the same AHB bus on the bus matrix
(2) Available only on STM32F469/479 line devices
(3) USB OTG HS is available only in the STM32F446 and STM32F469/479 lines

2.3.3 System architecture: STM32F7 Series

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 multiple masters and multiple slaves. There are four AXI bus accesses; the QUADSPI is accessible through the second access. This access allows the Cortex®-M7 to perform a memory-mapped region access in order to fetch code or data. This access also allows the Cortex®-M7 to perform a register access for QUADSPI registers configuration or for data transfer.

The QUADSPI is mapped on a dedicated layer on the AHB bus matrix allowing the

Cortex®-M7 to benefit from L1-Cache when accessing the cached data with 0-wait states.

QUADSPI is also accessible by all masters on AHB bus matrix. Registers accesses can be performed by GP DMA2 for data transfer. Memory-mapped region access can be performed by GP DMA1, MAC Ethernet, USB OTG HS, LTDC and DMA2D. This accessibility enables an efficient data transfer (like images for graphical applications).

The following figure shows the QUADSPI interconnection in the STM32F7 Series system.

Note: For MAC Ethernet, USB OTG HS, LTDC and DMA2D refer to the applicable product.

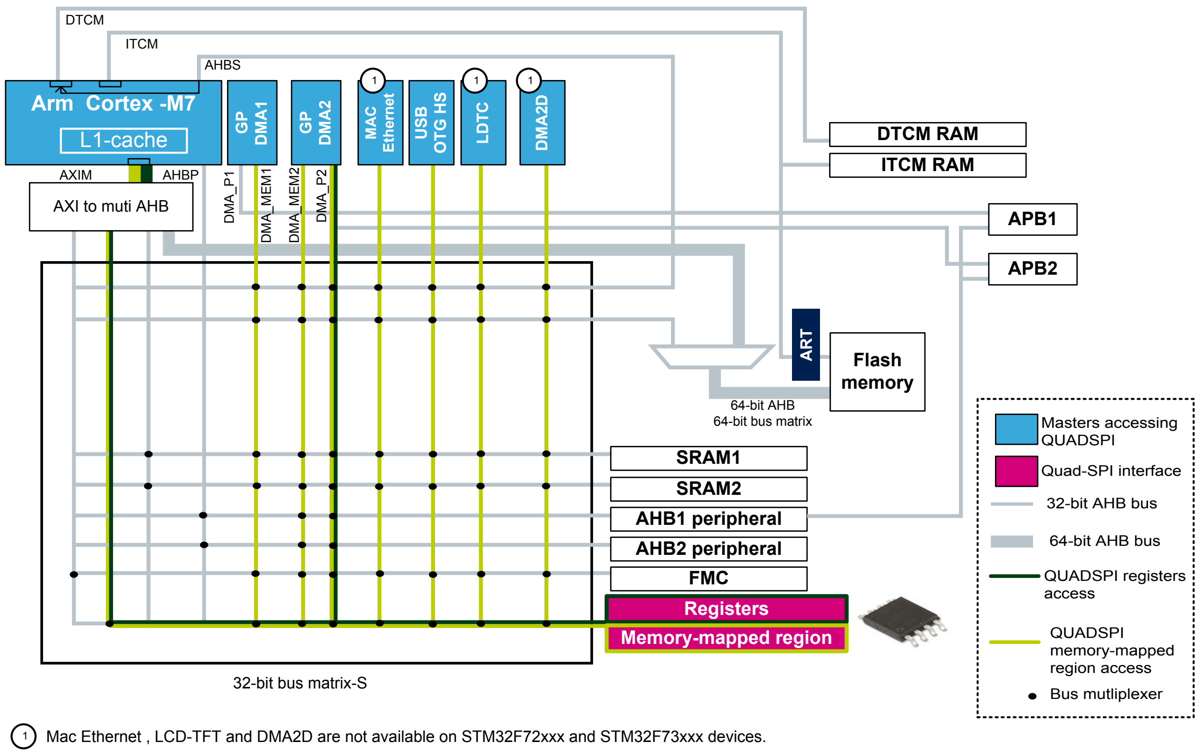


Figure 3. **System architecture: STM32F7 Series**

2.3.4 System architecture: STM32H7 Series

The main system architecture is based on three domains: D1,D2 and D3. Each domain contains a bus matrix that allows a connection between multiple masters and multiple slaves. A 64-bit AXI bus matrix for domain D1 and a 32-bit AHB bus matrix for each of the domains D2 and D3.

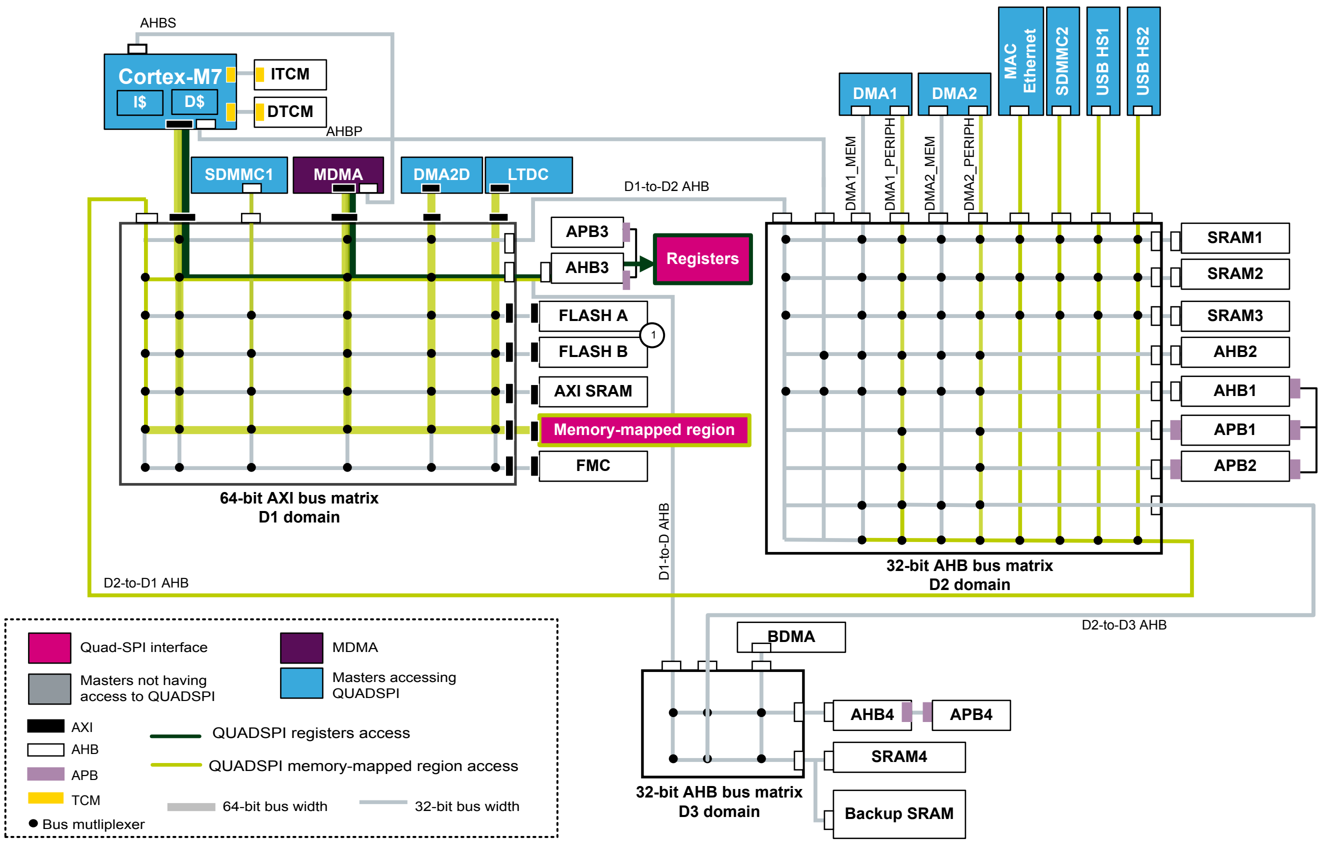
The three domains are connected to each other with the interdomains AHB buses which allow masters from a certain domain to access slaves from an other domain.

The QUADSPI is connected to the D1 domain and can be accessed through:

- A 64-bit AXI bus connected directly to the AXI bus matrix. It allows multiple masters to perform a memory-mapped region access for code and data fetch from D1 domain like the Cortex®-M7. It allows also data transfer from D1 domain (SDMMC1, MDMA, DMA2D and LTDC) and from the D2 domain (DMA1 and DMA2).
- A 32-bit AHB bus accessible through AHB3. It allows the Cortex®-M7 and the MDMA to perform a register access for data transfer or registers configuration.

The following figure shows the QUADSPI interconnection in the STM32H7 Series system.

Figure 4. System architecture: STM32H7 Series



(1) Mac Ethernet , LCD-TFT and DMA2D are not available on STM32F72x and STM32F73x devices

2.3.5 System architecture: STM32WB35xx and STM32WB55xx devices

The STM32WB35xx, STM32WB55xx devices system architecture consists mainly of a 32-bit multilayer AHB bus matrix that interconnects multiple masters and slaves. The QUADSPI is mapped on a dedicated layer on the AHB bus matrix.

The QUADSPI is accessed by relevant masters like the Cortex®-M4 either through S-bus or through I-bus and D-bus when remap is enabled. The QUADSPI is also accessible by DMA1 and DMA2.

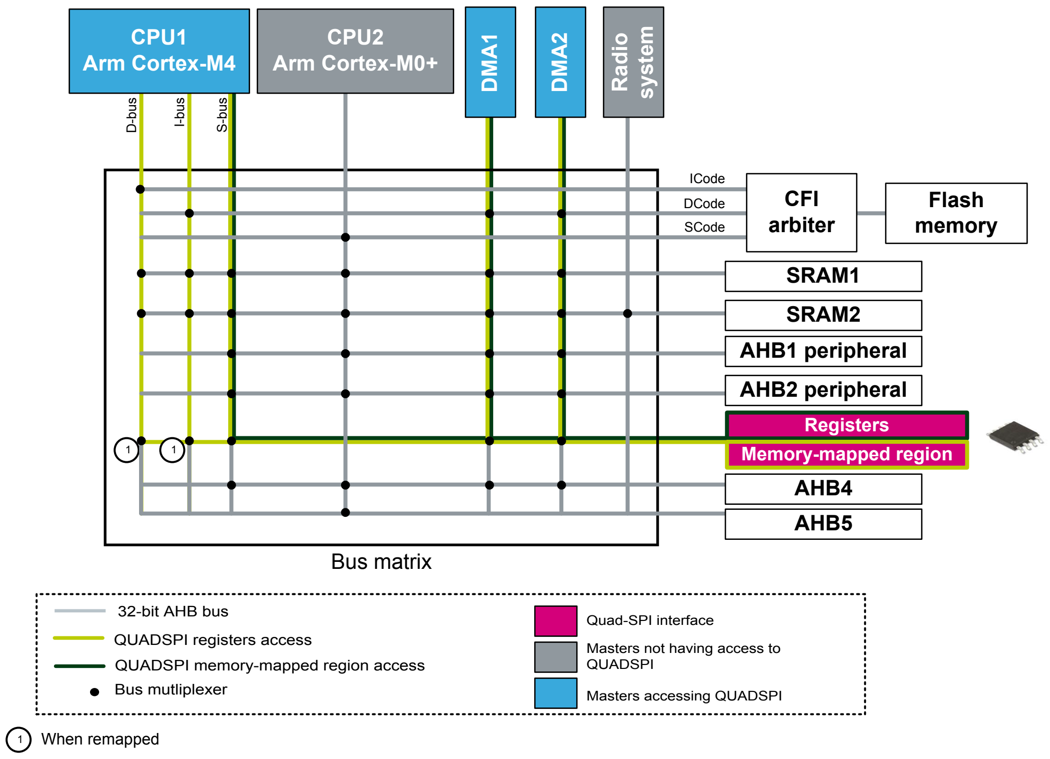
Enabling physical remap over I-bus and D-bus boosts execution performances for the Cortex®-M4.

The access to the QUADSPI can be either a registers access or a memory-mapped region access:

- The registers access can be done by the Cortex®-M4 for registers configuration or data transfer. The register access can also by done by DMA1 and DMA2 for data transfer.
- The memory-mapped region access can be done by the Cortex®-M4 for code and data fetch and also by the DMA1 and DMA2 for data transfer.

The following figure shows the QUADSPI interconnection in the STM32WB35xx and STM32WB55xx devices system.

Figure 5. System architecture:STM32WB35xx and STM32WB55xx



2.3.6 System architecture: STM32G4 Series devices

The STM32G4 series system architecture consists mainly of a 32-bit multilayer AHB bus matrix that interconnects multiple masters to multiple slaves.

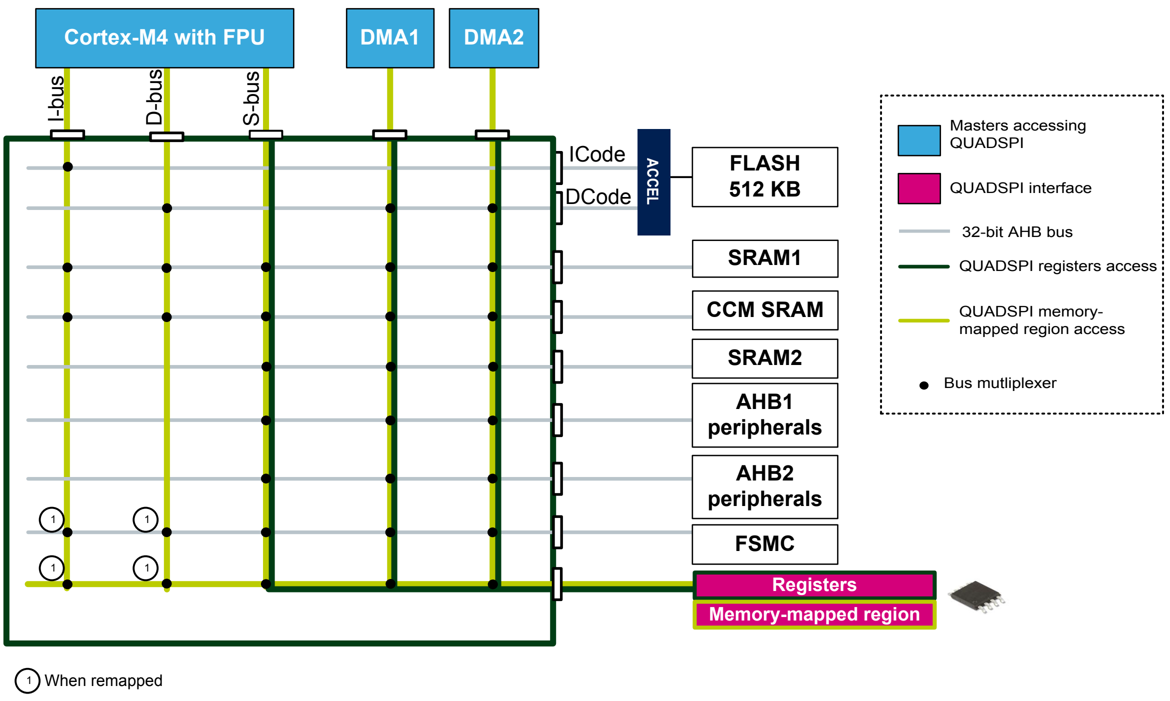
The QUADSPI can be accessed by relevant masters like the Arm® Cortex®-M4 with FPU either through S-Bus or through I-bus and D-bus when remap is enabled. QUADSPI is also accessible by DMA1 and DMA2.

Enabling physical remap over I-bus and D-bus boosts execution performances for the Cortex®-M4 .

The access to the QUADSPI can be either a registers access or a memory-mapped region access:

- The registers access can be done by the Cortex®-M4 for registers configuration or data transfer. The register access can be done also by the DMA1 and DMA2 for data transfer.
- The memory mapped region access can be done by the Cortex®-M4 for code and data fetch or by DMA1and DMA2 for data transfer.

Figure 6. System architecture: STM32G4 Series



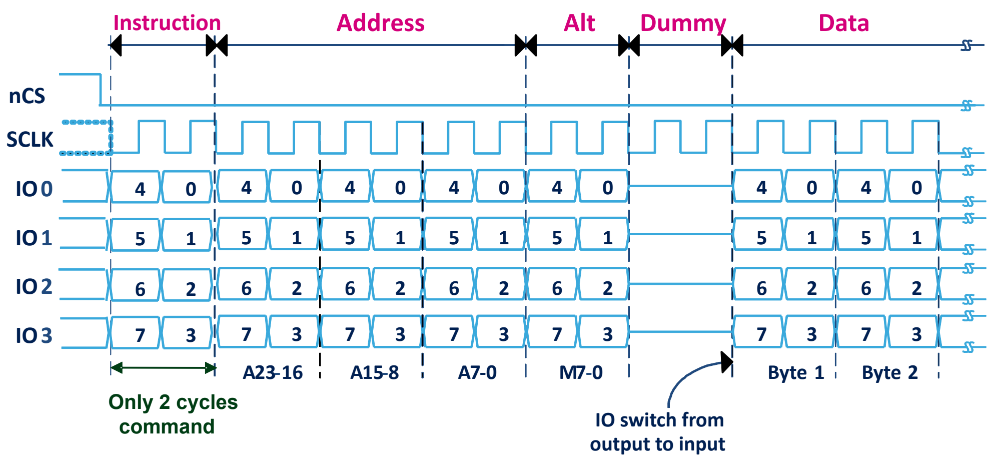
3 Quad-SPI interface description

3.1 Flexible frame format

The Quad-SPI 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.

The frame format can be configured only in Indirect mode or Memory-mapped mode but not in Status-flag polling mode. The figure below shows a reading of the sequence in Quad I/O SDR mode.

Figure 7. Reading sequence in Quad I/O SDR



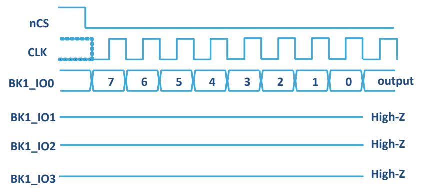
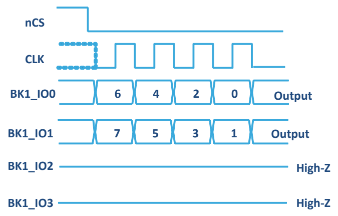
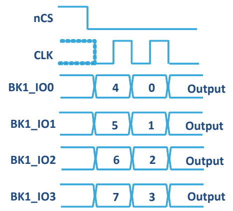
3.1.1 Instruction phase

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 over one, two or four lines. In some use cases where only the address is sent, the instruction phase can be skipped. The following table summarizes the different configurations for instruction phase.

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 one line: Single-SPI mode	IMODE[1:0] = 01	 nCS CLK BK1_IO0 7 6 5 4 3 2 1 0 output BK1_IO1 High-Z BK1_IO2 High-Z BK1_IO3 High-Z
	Instruction on two lines: Dual-SPI mode	IMODE[1:0] = 10	 nCS CLK BK1_IO0 6 4 2 0 Output BK1_IO1 7 5 3 1 Output BK1_IO2 High-Z BK1_IO3 High-Z
	Instruction on four lines: Quad-SPI mode	IMODE[1:0] = 11	 nCS CLK BK1_IO0 4 0 Output BK1_IO1 5 1 Output BK1_IO2 6 2 Output BK1_IO3 7 3 Output

3.1.2 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 over 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

The table below 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 one line: Single-SPI mode	ADMODE[1:0]=01		
	Address on two lines: Dual-SPI mode	ADMODE[1:0]=10		
	Address on four lines: Quad-SPI mode	ADMODE[1:0] =11		

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

3.1.3 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, Micron and Macronix where an alternate byte is sent continuously to keep the memory in Execute-in-place mode.

The alternate-byte phase is fully configurable and permits 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 over one, two or four lines. If not needed, the alternate-byte phase can be skipped.

The table below 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		

Register configuration		Indirect mode	Automatic-polling mode	Memory-mapped mode
Alternate-byte size QUADSPI_CCR[17:16]	4-byte	ABSIZE[1:0] =11		
Alternate-byte phase QUADSPI_CCR[15:14]	No alternate-byte: skipped	ABMODE[1:0] = 00		
	Alternate-byte on one line: Single-SPI mode	ABMODE[1:0] = 01		
	Alternate-byte on two lines: Dual-SPI mode	ABMODE[1:0] = 10		
	Alternate-byte on four lines: Quad-SPI mode	ABMODE[1:0] = 11		

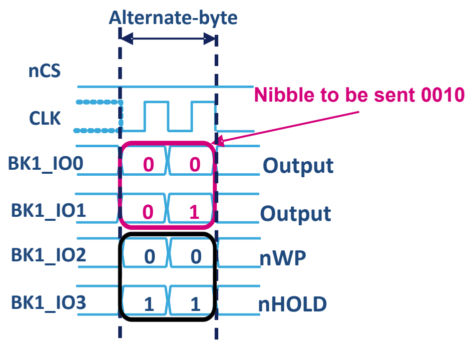
Note: In Dual-Flash mode when DFM = 1, the alternate-byte to be sent to Flash1 is exactly the same as the ones to be sent to Flash2.

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 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 I/O 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 8. **Alternate-byte phase: sending a nibble in Dual-SPI mode**



3.1.4 Dummy-cycle phase

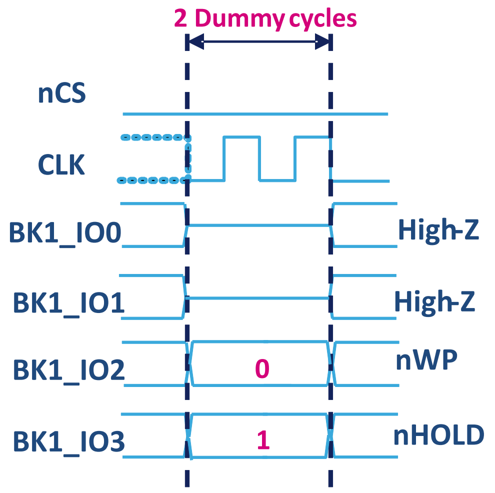
The dummy-cycle phase is needed in some cases when operating at high clock frequencies. This phase allows to ensure enough turnaround time for changing the data signals from output mode to input mode.

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] filed 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 QUADSPI clock cycle.

During this phase, if the QUADSPI 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 Quad-SPI memory. This is fully managed by hardware (QUADSPI peripheral) so nothing needs to be configured by the user.

Figure 9. Dummy-cycle: IO2 maintained low and IO3 maintained high by hardware



3.1.5 Data phase

In this phase; the data is sent or received from or to the Quad-SPI memory. The data phase is fully configurable and permits to send, receive or both any number of bytes to or from the Quad-SPI memory device.

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

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

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

Depending on the software and the hardware configurations, the data transfer can be done 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.

The following table 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®, DMALTDC, DMA2D).
	Writedata	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 one line: Single-SPI mode	DMODE[1:0] = 01		
	Data on two lines: Dual-SPI mode	DMODE[1:0] = 10		

Register configuration		Indirect mode	Automatic-polling mode	Memory-mapped mode
Data phase QUADSPI_CCR[15:14]	Data on four 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.*

3.2 Multiple hardware-configurations

STM32 devices offer a very flexible Quad-SPI interface that permits the connection of external Quad-SPI memories in different hardware configurations. The user can then choose its own configuration.

Depending on the used hardware configuration, the number of used GPIOs can be up to 11. The table below summarizes different use cases.

Table 8. **Hardware configurations versus used GPIO number**

-	-	Single-Flash mode		Dual-Flash mode
Single-SPI/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 3.2.4: Dual-Flash mode.*

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 QUADSPI is active.

3.2.1 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.

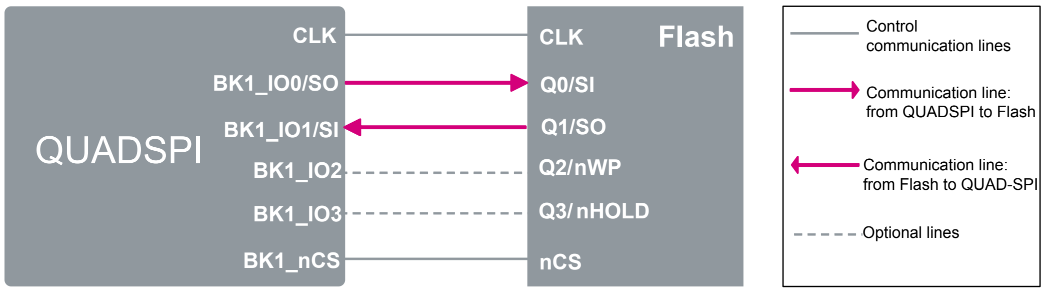
Note: Full duplex transfer is not supported.

The IO2 and IO3 lines are optional:

- When used (IO2 and IO3 are connected to the Quad-SPI 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 (QUADSPI 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 10. Hardware configuration: Single-SPI mode

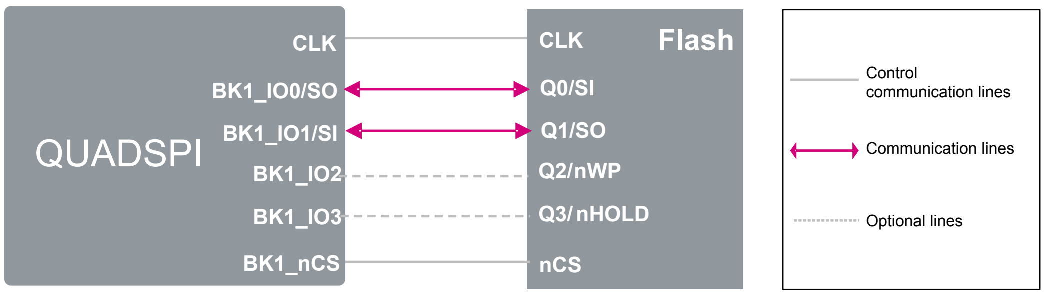

QUADSPI
CLK
BK1_IO0/SO
BK1_IO1/SI
BK1_IO2
BK1_IO3
BK1_nCS
Flash
CLK
Q0/SI
Q1/SO
Q2/nWP
Q3/nHOLD
nCS
Control communication lines
Communication line: from QUADSPI to Flash
Communication line: from Flash to QUAD-SPI
Optional lines

3.2.2 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 11. Hardware configuration: Dual-SPI mode



3.2.3 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-SPI 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 Quad-SPI mode, depending on the Quad-SPI 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.

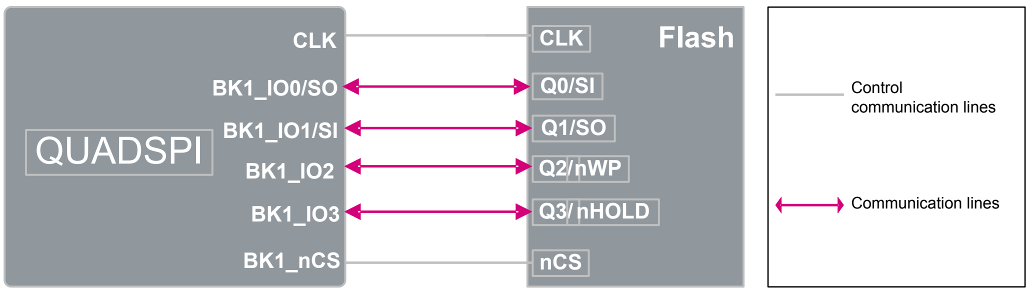


Figure 12. **Hardware configuration: Quad-SPI mode**

3.2.4 Dual-Flash mode

In Dual-Flash 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. A throughput of two bytes per cycle can be attained with Dual-Flash in DDR Quad-SPImode.

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. Section 3.2: Multiple hardware-configurations illustrates all possible hardware configurations.

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

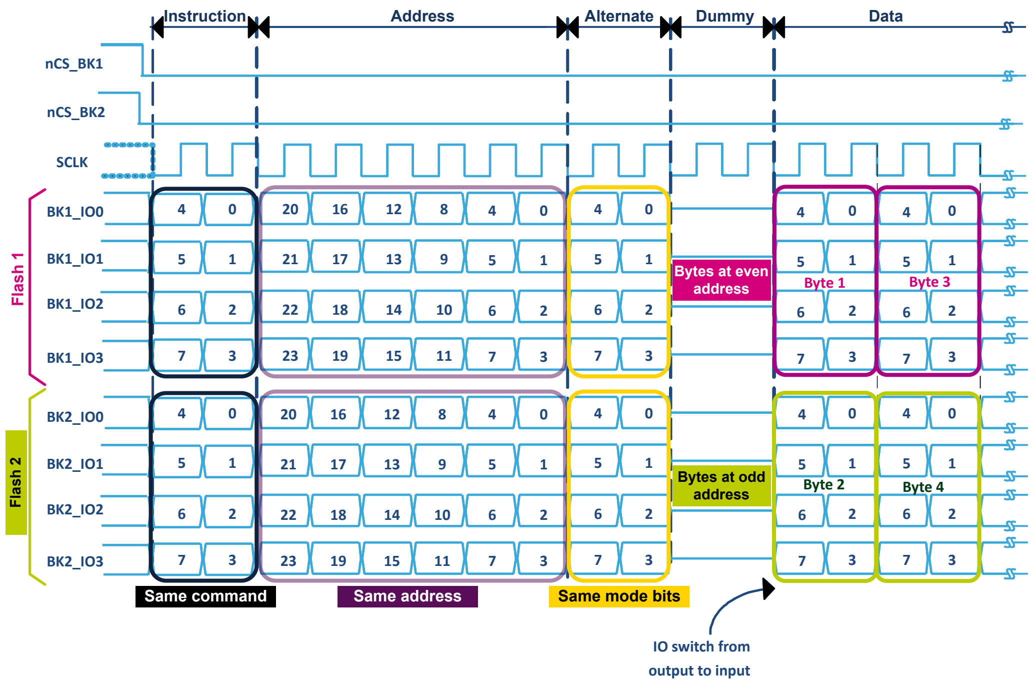
When using two external Quad-SPI memories the size to be configured in FSIZE[4:0] should reflect the total Flash memory capacity, which is double the size of one individual component.

To support dual die packages with two chip-selects and dual Quad-SPI devices, the FIFO size is always 32-bytes either in Single-Flash or Dual-Flash mode.

Note: The addressable space in Memory-mapped mode is up to 256 Mbytes either in Single-Flash mode or Dual-Flash mode.

The following figure shows an example of a read sequence in Dual-Flash Quad I/O SDR mode.

Figure 13. **Read sequence in Dual-Flash Quad I/O SDR mode**



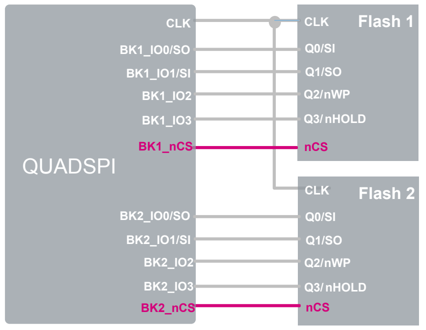
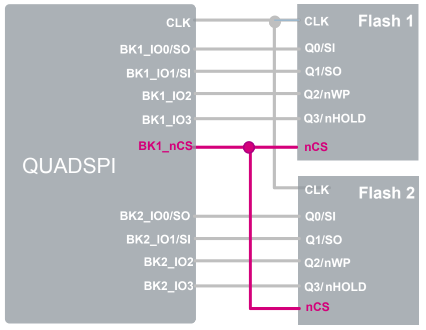
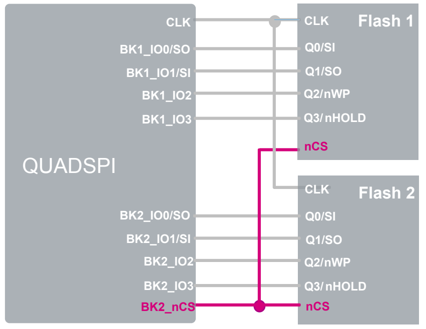
Note that all bytes at even addresses are stored in Flash 1 while all bytes at odd addresses are stored in Flash 2. As described in the figure above, in Dual-Flash mode the same command, address and alternate are sent to both Flash 1 and Flash 2. For example to read the first four bytes in Dual-Flash memory-mapped mode from 0x90000 000 to 0x9000 0003 the following sequence is done by QUADSPI peripheral:

- The address 0x0000 0000 is sent to both Flashes and Byte 1 (at even address 0x9000 0000) is read from Flash 1 while Byte 2 (at odd address 0x9000 0001) is read from Flash 2.
- Then the address 0x0000 0001 is sent to both Flashes and Byte 3 (at even address 0x9000 0002) is read from Flash 1 while Byte 2 (at odd address 0x9000 0003) is read from Flash 2.

Note:

- *In Dual-Flash mode both device models must be identical, because in this mode the same commands and addresses are issued in parallel to both Flash memories; this permits to double the available Quad-SPI external Flash size. 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 the table below, each memory device is configured in Quad-SPI mode. It is possible to connect each device in Single-SPI or Dual-SPI mode. If DFM = 1, both devices must be configured in the same way. This permits to double the available external data size and throughput.*
- *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 hardware configurations**

Used nCS	nCS configuration	Flash memory mode		Hardware configuration[1]
2 nCS enabled	BothnCS_BK1 and nCS_BK2 not connected together	Single-Flash DFM =0[2]	FSEL= 0 Flash 1 enabled	
			FSEL= 1 Flash 2 enabled	
		Dual-Flash DFM =1		
1 nCS enabled	nCS_BK1 connected to both devices	Dual-Flash DFM =1		
	nCS_BK2 connected to both devices	Dual-Flash DFM =1		

1. *Pink lines highlight the used chip select.*
2. *When Single-Flash mode is selected DFM = 0, the user can switch between Flash 1 or Flash 2 using FSEL bit.*

3.2.5 DDR and SDR mode

The SDR mode is activated by default, DDR mode permits to sample at rising and falling edge of each clock cycle and enables the possibility 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 QUADSPI to operate 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 communication start triggering and the configuration procedure are the same as in SDR.
- The command is sent every clock cycle like in SDR mode.
- The alternate, data and address phases are sent on both edges of the clock.
- The dummy cycles are counted every clock cycle like in SDR mode.

3.3 Three operating modes

3.3.1 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 Quad-SPI memory (available in Memory-mapped mode).
- For all the operations to be performed through the QUADSPI data registers using CPU or using DMA.
- To configure the Quad-SPI Flash memory.

In Indirect mode, all operations are performed through the QUADSPI register 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 QUADSPI 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 4 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-flag polling mode and then poll the status register inside the Flash memory to know when the programming or the erase operation is completed.

3.3.2 Status-flag polling mode

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

- To read Quad-SPI Flash memory status register.
- To poll autonomously for the end of an operation: QUADSPI 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 Quad-SPI 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.

3.3.3 Memory-mapped mode

The Memory-mapped mode is used in below cases:

- For reading operations.
- To use external Quad-SPI Flash memory like an internal memory, so any AHB master can read data autonomously.
- For code execution from external Quad-SPI 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 the Quad-SPI memory as an internal memory. The CPU can execute code from the Quad-SPI memory as well.

When Memory-mapped mode is used, a prefetching mechanism fully managed by the hardware permits the optimization of the read and the execution performances from the external Quad-SPI memory. Given that all the communication phases such as sending opcode or address are managed by the QUADSPI peripheral, a 32-bytes FIFO (16-bytes for STM32L4 and STM32G4 Series) 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 QUADSPI peripheral is configured, the Quad-SPI memory is accessed as soon as there is a read request on the AHB; this is done in the Quad-SPI memory-mapped address range. This action is totally transparent for the user.

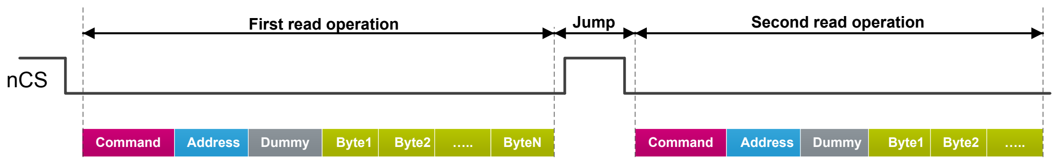
An LTDC master for example can access autonomously to the external Flash memory where all the access operations are fully managed by the Quad-SPI interface. Meanwhile, the Cortex®-M CPU is executing code from the internal Flash memory.

The Quad-SPI interface is able to manage up to 256 Mbytes memory starting from 0x9000 0000 to 0x9FFF FFFF in Memory-mapped mode.

Execute in place (XIP)

The prefetch buffer supports execution in place, therefore the code can be executed directly from the external Quad-SPI memory. The QUADSPI 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 is completed faster since the value is already prefetched.

Figure 14. Executing non-sequential code from Quad-SPI



Booting from Quad-SPI Flash memory

Boot from the Quad-SPI memory is not supported but the user can boot from the internal Flash memory and then configure the QUADSPI in Memory-mapped mode and then the execution starts from the Quad-SPI memory. For more details on how to execute from the external Quad-SPI memory, refer to Section 6.2: Executing from external Quad-SPI memory: extend internal memory size.

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

For all the supported STM32 devices, the Quad-SPI memory is accessible by Cortex®-M through system bus. For the STM32L4, STM32WB and STM32G4 Series, the QUADSPI 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 QUADSPI is remapped at address 0x0000 0000, only 128 Mbytes are remapped. Even when aliased in the boot memory space, the Quad-SPI memory is still accessible at its original memory space.

Note: The data length register QUADSPI_DLR has no meaning in Memory-mapped mode.

3.4 Special features

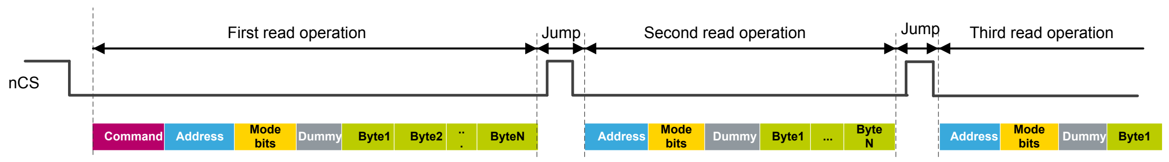
3.4.1 Send instruction only-once (SIOO)

The SIOO feature is named also by some memory manufacturers as *continuous-read mode*, *burst mode* or even as *performance-enhanced 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 to 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; this action permits the reduction of up to eight cycles (in Single I/O mode) for the command. This is a very interesting feature to reduce access overhead to the Quad-SPI memory.

Data are prefetched continuously while the FIFO is not full, when a discontinuous access is detected, the QUADSPI rises chip-select and starts a new read operation without sending the command but sending directly the new address.

Figure 15. **Executing non-sequential code from QUADSPI with SIOO enabled**



The SIOO feature is supported by many Quad-SPI memory manufacturers such as Micron, Spansion and Macronix, nevertheless before using it, the user has 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 6.2: Executing from external Quad-SPI memory: extend internal memory size for more details on enabling this feature.
- Configure the QUADSPI peripheral by setting the SIOO bit in QUADSPI_CCR register.

3.4.2 Delayed data sampling

For read operations from the external memories, the delayed data sampling is useful when the signals are delayed due to constraints on the PCB layout optimizations; hence the optimization compensates 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.

For write operations from the external memories in DDR mode, the output data can be shifted by one quarter of the QUADSPI 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 QUADSPI timing characteristics refer to the relevant products datasheet.

3.4.3 Timeout counter

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

After each access in Memory-mapped mode, the QUADSPI 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 is 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 is low and permits 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 complete read command sequence is started by the Quad-SPI interface.*

3.4.4 Additional status bits

Other than status flags described in Table 13. QUADSPI interrupts summary, the QUADSPI_SR status register includes additional status bits, the table below summarizes the status flags.

Table 10. **Additional status bits**

Name	Size	Description
FLEVEL	6 bits (5 bits for STM32L4 and STM32G4 Series)	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.

3.4.5 Busy bit and abort functionality

Busy bit

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

Note: *Some QUADSPI registers cannot 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.*

The following table summarizes different cases when the BUSY bit is reset in different QUADSPI operating modes:

Table 11. **BUSY bit reset in different Quad-SPI modes**

Quad-SPI mode	BUSY bit reset
Indirect mode	• The QUADSPI has completed the requested command sequence and the FIFO is empty • Due to an abort
Automatic-polling mode	• After the last periodic access is complete, due to a match when APMS =1 • Due to an abort
Memory-mapped mode	• On a timeout event • Due to an abort • QUADSPI peripheral is disabled

ABORT bit

When an application is running, any ongoing QUADSPI operation can be aborted by setting the ABORT bit in the QUADSPI_CR register. Once the abort is completed, the BUSY bit and the ABORT bit are automatically reset and the FIFO is flushed. If an abort occurs on an ongoing AXI/AHB burst operation, the QUADSPI allows the ongoing burst to complete properly before resetting the BUSY bit and the ABORT bit.

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

3.4.6 4-byte address mode

This mode is named also by some brands *extended-address mode*. In this mode a 4-byte address is sent; this action permits to address the Flash memories with sizes up to 4 Gbytes. This mode is supported by many Quad-SPI memory manufacturers such as Micron, Spansion and Macronix.

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 QUADSPI 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.

The following table 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 256 bytes
16-Bits address	01	01111	Up to 64 Kbytes
24-Bits address	10	10111	Upto 16 Mbytes
32-Bits address	11	11111	Up to 4 Gbytes(1)

1. *Only the first 256 Mbytes of the Quad-SPI memory can be read in Memory-mapped mode (from 0x90000000 to 0x9FFF FFFF).*

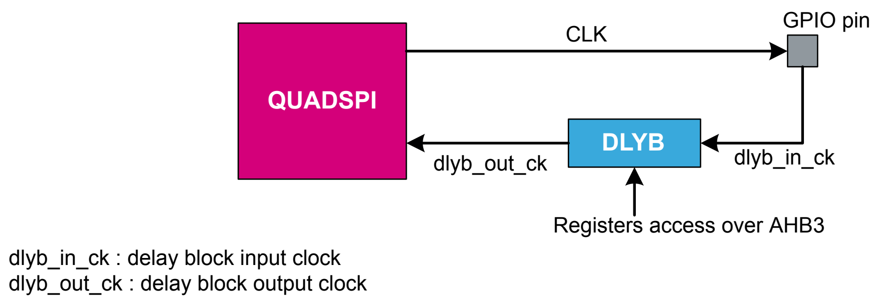
Cautions on 4-byte address mode

- In Indirect mode, if the address plus the data length exceeds the Flash memory size, TEF flag is 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 256 Mbytes 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.

3.4.7 QUADSPI and delay block in STM32H7 Series

In the STM32H7 Series, the QUADSPI has its own delay block accessible through AHB3. The delay block can be used to generate a sampling clock which is phase-shifted from the output clock sent on the chip pin. The delay block aligns the sampling clock on the incoming data.

Figure 16. QUADSPI and delay block


QUADSPI
CLK
GPIO pin
DLYB
dlyb_out_ck
dlyb_in_ck
Registers access over AHB3
dlyb_in_ck : delay block input clock
dlyb_out_ck : delay block output clock

3.5 Interrupts and DMA usage

3.5.1 Interrupts usage

The QUADSPI 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 QUADSPI global interrupt on the NVIC side.

The table below summarizes all the supported interrupts.

Table 13. QUADSPI 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. *Al lclear bits are available in the QUADSPI_FCR register.*

3.5.2 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 Quad-SPI memory is allowed.

DMA usage with QUADSPI in Indirect mode

In DMA mode, the DMA is the flow controller. When QUADSPI FIFO threshold is reached while DMAEN bit is set, DMA requests are generated from QUADSPI to the DMA.

The transfer is started when

- The DMAEN bit is set and the QUADSPI 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 when configuring the DMA for data transfer from/to the QUADSPI, the QUADSPI should be considered as a peripheral:

- Memory to peripheral mode in case of writing data to the QUADSPI from the internal memory.
- Peripheral to memory mode in case of reading data from the QUADSPI to be transferred into the internal memory.

Also, the address of the QUADSPI should be written into the peripheral address register (DMA channel/stream x peripheral address).

The table below summarizes the different DMA requests and transfer directions versus the STM32 series.

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

Product[1]	DMA1	DMA2	MDMA
STM32L4 Series	Request 5 Channel 5	Request 3 Channel 7	NA
STM32F4 Series	NA	Stream 7 Channel 3	NA
STM32F7 Series	NA	Stream 7 Channel 3	NA
STM32H7 Series	NA	NA	quadspi_ft_trg channel X[0..15]/Stream22
	NA	NA	quadspi_tc_trg channel X[0..15]/Stream23

1. *For applicable devices of each series embedding a QUADSPI.*

DMAusage with QUADSPI in Memory-mapped mode

In Memory-mapped mode the QUADSPI allows the access to the external memory for read operation through the memory mapped address region (from 0x9000 0000 to 0x9FFF FFFF) and allows the external memory to be seen just like an internal memory.

In that case when configuring the DMA to transfer data from the QUADSPI memory-mapped region into an other internal memory, the QUADSPI memory-mapped region should be considered as a memory when configuring the DMA registers:

Memory to memory mode in case of reading data from the QUADSPI memory mapped region to be transferred into the internal memory.

Also the address of the QUADSPI should be written into the peripheral address register (DMA channel/stream x memory address).

In Memory-mapped mode the DMA is the flow controller since the QUADSPI does not generate DMA requests.

In Memory-mapped mode either DMA1 or DMA2 can be used to transfer data from the external Quad-SPI memory to any other memory or peripheral. To perform a transfer using DMA from external Quad-SPI memory to any other memory or a peripheral, the user should configure the DMA by setting the source address (from 0x9000 0000), 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 QUADSPI address range (from 0x9000 0000 to 0x9FFF FFFF).

Once the DMA configured transfer is started by software, the DMA reads the data from the Quad-SPI memory exactly as an internal memory. The QUADSPI 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 register DMA_SxNDTR (or DMA_CNDTRx register for STM32L4x6xx). There is no need to configure the QUADSPI_DLR register as it has no effect in 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.

QUADSPI and master DMA in STM32H7 Series

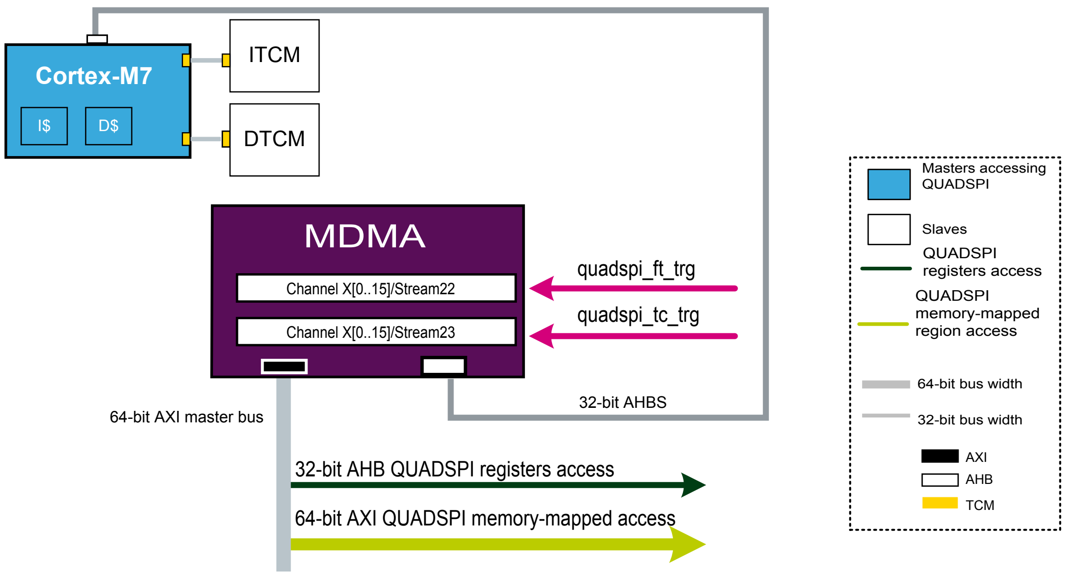
Int he STM32H7 Series the MDMA manages the DMA access to the Quad-SPI interface. The MDMA can access the QUADSPI in the following ways:

- Directly from the AXI bus matrix over a 64-bit AXI bus for memory mapped access.
- From AHB3 over a 32-bit AHB bus for registers access.

The master DMA offers two trigger signals from the Quad-SPI interface to procure more flexibility for the user's application. The two trigger signals are:

- quadspi_ft_trg: QUADSPI FIFO threshold trigger.
- quadspi_tc_trg: QUADSPI transfer complete trigger.

Figure 17. **QUADSPI and master DMA**


Cortex-M7
I$
D$
ITCM
DTCM
MDMA
Channel X[0..15]/Stream22
Channel X[0..15]/Stream23
quadspi_ft_trg
quadspi_tc_trg
32-bit AHBS
64-bit AXI master bus
32-bit AHB QUADSPI registers access
64-bit AXI QUADSPI memory-mapped access
Masters accessing QUADSPI
Slaves
QUADSPI registers access
QUADSPI memory-mapped region access
64-bit bus width
32-bit bus width
AXI
AHB
TCM

3.6 Low-power modes

TheSTM32 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 QUADSPI has to be reconfigured after wakeup from this mode.

Note: The Quad-SPI 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 Quad-SPI memory in low-power mode before entering the STM32 in low-power mode. More information on reducing power consumption is available on Section 7.2: Decreasing power consumption.

It is possible to perform transfers in Sleep mode while the CPU is stopped thanks to the STM32 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 QUADSPI (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 wakeup the STM32.

Refer to the products reference manuals for low-power mode configuration details.

4 QUADSPI configuration

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

4.1 GPIOs configuration

The user should configure the GPIOs to be used for interfacing with the Quad-SPI memory, and this is dependent on the preferred hardware configuration. For more details on the hardware configurations please refer to Section 3.2: Multiple hardware-configurations.

Note: *It is recommended to reset the QUADSPI 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 Quad-SPI memories are connected.

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

4.1.1 GPIOs configuration using STM32CubeMX tool

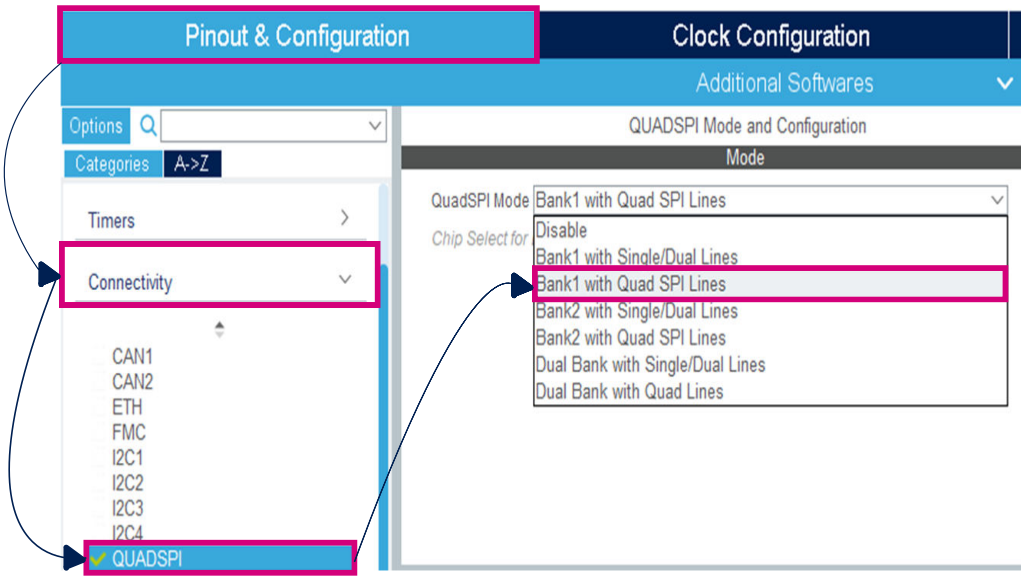
The following example shows how to configure QUADSPI GPIOs in Quad I/O mode using Bank1 GPIOs.

Using the STM32CubeMX tool is a very simple, easy and rapid way to configure the QUADSPI peripheral and its GPIOs as it permits the generation of a project with a preconfigured QUADSPI.

Once the STM32CubeMX project is created, a hardware configuration can be chosen on the Modewindow. This window is found on Pinout and configuration tab under the Connectivity section and by selecting the QUADSPI menu.

The figure below shows how to select the QUADSPI hardware configuration with the STM32CubeMX where Bank1 GPIOs are used.

Figure 18. STM32CubeMX: QUADSPI GPIOs configuration


Pinout & Configuration
Clock Configuration
Additional Softwares
Options
Categories
A->Z
Timers
Connectivity
CAN1
CAN2
ETH
FMC
I2C1
I2C2
I2C3
I2C4
QUADSPI
QUADSPI Mode and Configuration
Mode
QuadSPI Mode
Chip Select for
Disable
Bank1 with Single/Dual Lines
Bank1 with Quad SPI Lines
Bank2 with Single/Dual Lines
Bank2 with Quad SPI Lines
Dual Bank with Single/Dual Lines
Dual Bank with Quad Lines

QuadSPI Mode: Bank1 with Quad SPI Lines

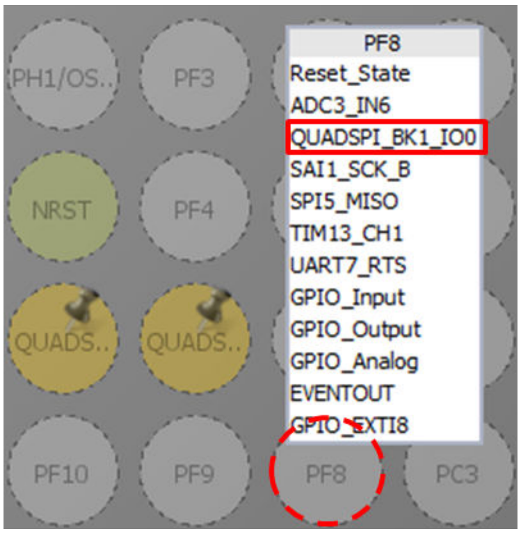
Note: *Dual-bank mode in STM32CubeMX refers to Dual-Flash mode.*

If after selecting one hardware configuration (as shown in the previous figure) 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 QUADSPI alternate functions availability versus GPIOs, refer to the alternate function mapping table in the relevant datasheet.

The following figure shows how to configure manually a PF8 pin to QUADSPI_BK1_IO0 alternate function.

Figure 19. **STM32CubeMX: PF8 pin configuration to QUADSPI_BK1_IO0 alternate function**



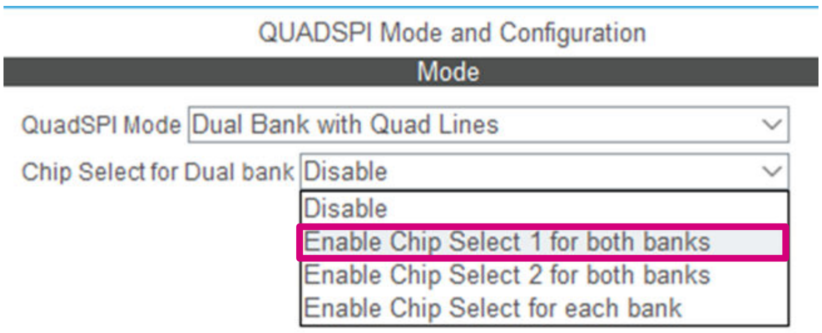
The used pins are highlighted in green once the GPIOs of the Quad-SPI interface are correctly configured.

Dual-Flash case

If Dual-bank is selected, the user should select one of the listed chip-select configurations. For more details on different Dual-Flash chip-select configurations refer to Section 3.2.4: Dual-Flash mode.

The following figure shows how to configure chip-select 1 for both banks with STM32CubeMX.

Figure 20. **STM32CubeMX: Dual-Flash QUADSPI with chip-select 1 configuration**



QUADSPI Mode and Configuration

Mode

QuadSPI Mode: Dual Bank with Quad Lines

Chip Select for Dual bank: Disable

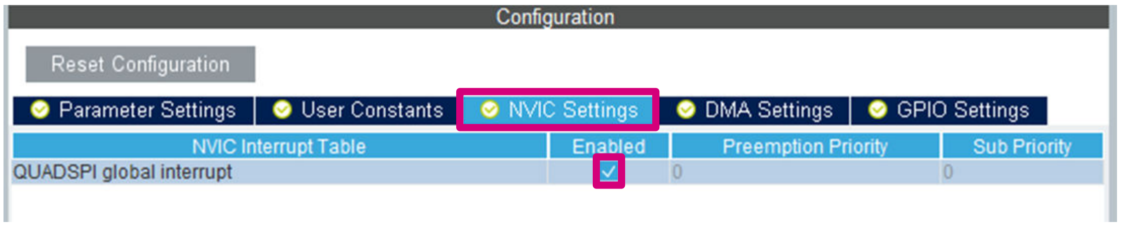
Disable
Enable Chip Select 1 for both banks
Enable Chip Select 2 for both banks
Enable Chip Select for each bank

Enabling QUADSPI interrupts

To be able to use QUADSPI interrupts, the user should enable the QUADSPI 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. QUADSPI interrupts summary).

The figure below shows the configuration window where the QUADSPI global interrupt can be enabled, under the NVIC tab.

Figure 21. STM32CubeMX: enabling QUADSPI global interrupt



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
QUADSPI global interrupt	✓	0	0

4.2 QUADSPI peripheral configuration and clock

4.2.1 QUADSPI peripheral configuration (QUADSPI_CR register)

The QUADSPI 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 (DMAEN bit not available for STM32H7 Series products). In case of interrupt usage, their respective enable bit can also be set during this phase. The FIFO level both for DMA request generation and for interrupt generation is programmed in the FTHRES bits.

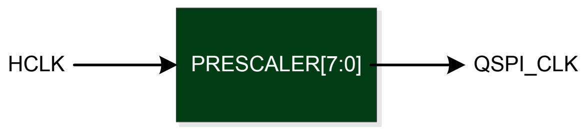
If timeout counter is needed, the TCEN bit can be set and the timeout value can be programmed in the QUADSPI_LPTR register.

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

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

The following figure shows a QUADSPI clock scheme for STM32F4 , STM32L4 , STM32F7 , STM32WB and STM32G4 Series devices where QSPI_CLK = HCLK / (Prescaler + 1).

Figure 22. QUADSPI clock configuration on QUADSPI_CR register



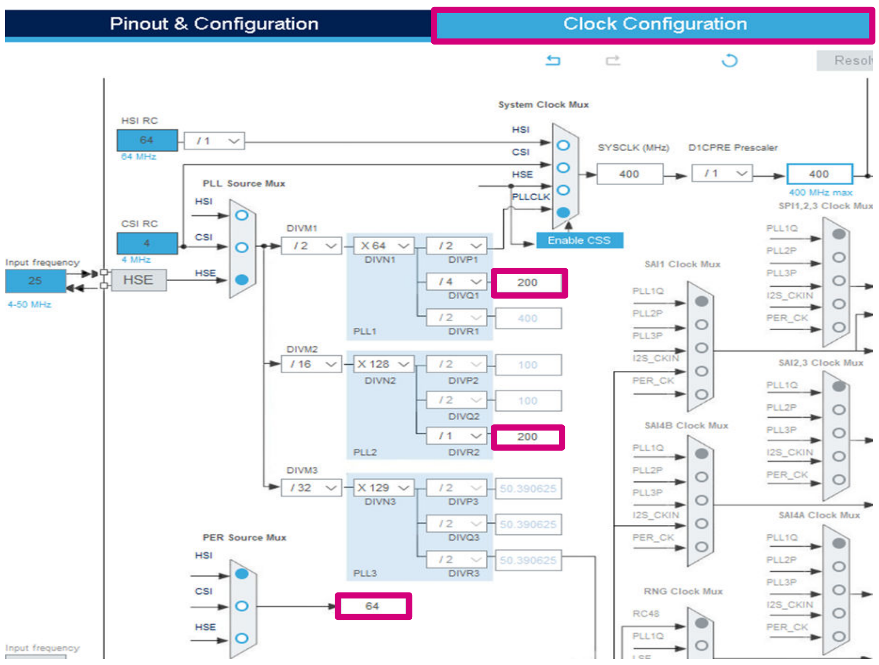
In the STM32H7 series devices the QUADSPI contains two different source clocks:

- quadspi_ker_ck
  It is the source clock to generate QUADSPI CLK using the following relation (QSPI_CLK=quadspi_ker_ck/(Prescaler +1).
- quadspi_hclk (hclk3)
  It is the source clock for the register interface. This clock has no impact on the QUADSPI CLK.

STM32CubeMX permits the configuration of quadspi_ker_ck source clock in the clock configuration section.

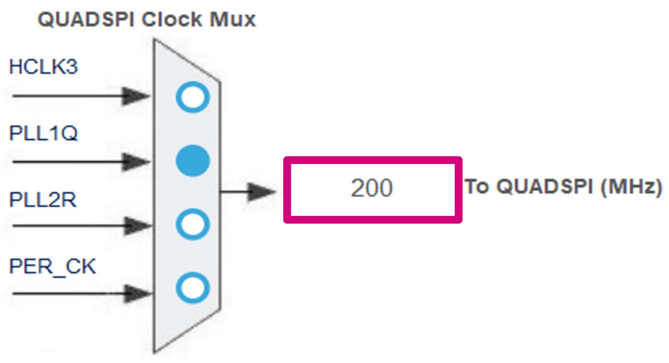
The following figure shows the multiple source clocks for quadspi_ker_ck using STM32CubeMX.

Figure 23. **STM32CubeMX: quadspi_ker_ck source clock configuration in STM32H7 Series**



The source clock for quadspi_ker_ck can be selected by using the QUADSPI clock mux as shown in the following figure.

Figure 24. **STM32CubeMX: quadspi_ker_ck source clock selection in STM32H7 Series**



4.2.2 Quad-SPI flash memory parameter 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 QUADSPI parameters can be changed when the application is running but not during an ongoing transfer, in other words not when the busy bit is set. If a change in the parameters is needed during an ongoing transfer, the abort bit can be used to stop the ongoing operation, then the configuration can be changed.

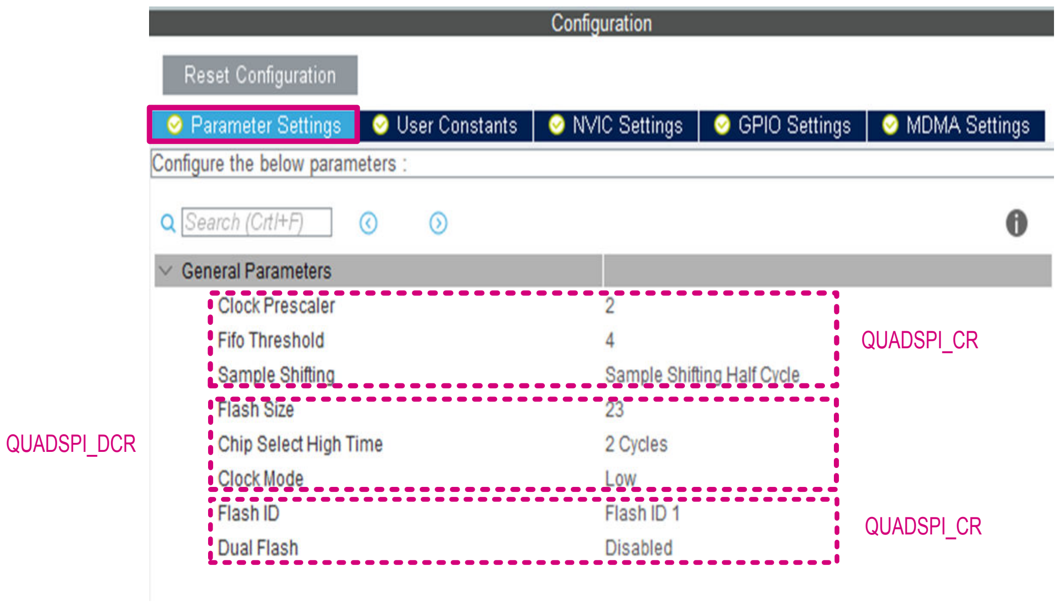
QUADSPI configuration using STM32CubeMX

The STM32CubeMX tool can be used to configure the QUADSPI peripheral. Once the GPIOs have been correctly configured, as already described, the QUADSPI parameters can be configured. However, all configurations related to starting communications 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. The figure below shows an example of QUADSPI configuration according to the following conditions:

- Clock prescaler = 2 => QSPI_CLK = FAHB/3 (*forSTM32H7 Series QSPI_CLK=quadspi_ker_ck/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 in case of write operation or FTF is set if there are five or more valid bytes that can be read from the FIFO
- Sample shifting half cycle enabled =>sample the data read from the memory half-clock cycle later (adjust the sampling time in case of cumulated delays on PCB)
- Flash memory size is 16 Mbytes => 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
- Flash ID = Flash ID 1 => select the Flash memory 1 to be addressed in Single-Flash mode FSEL=0
  Note that FSEL is ignored when DFM = 1
- Dual Flash = disabled ==> Dual-Flash mode disabled DFM = 0

Figure 25. **STM32CubeMX: QUADSPI peripheral configuration**



Clock Prescaler	2
Fifo Threshold	4
Sample Shifting	Sample Shifting Half Cycle
Flash Size	23
Chip Select High Time	2 Cycles
Clock Mode	Low
Flash ID	Flash ID 1
Dual Flash	Disabled

4.2.3 QUADSPI and MPU configuration

The memory protection unit (MPU) can be used to make the user application more robust and more secure by protecting a memory region.

When using an STM32 product based on a Cortex®-M7 (such as STM32F7 series and STM32H7 series), it is highly recommended to configure the QUADSPI memory region accessible in Memory-mapped mode as *strongly ordered memory*.

The QUADSPI memory region accessible in Memory-mapped mode goes from 0x9000 0000 to 0x9FFF FFFF. By performing this action, the CPU is prevented to access this region while Memory-mapped mode is not enabled.

Int hat same area (0x9000 0000 to 0x9FFF FFFF), once the external Flash memory size is defined and the Memory-mapped mode is enabled, it is recommended to configure the unused remaining memory area also as *strongly ordered memory*.

For more information about configuring the memory protection unit refer to the application note *Managing memory protection unit (MPU) in STM32 MCUs* (AN4838).

4.2.4 Quad-SPI memory device configuration

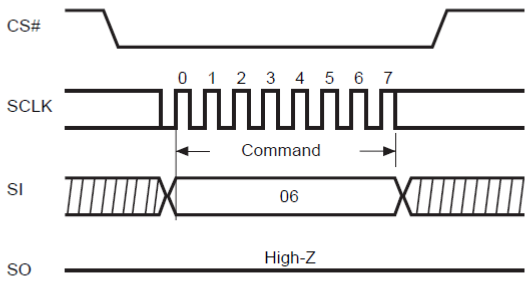
The Quad-SPI memory should be configured after the QUADSPI peripheral has been configured. The Quad-SPI memory configuration depends on the selected configuration for QUADSPI for Indirect mode.

Enable writing to Quad-SPI Flash memory

The write-enable command should be sent to the memory in order to set the write-enable latch (WEL) bit.

The 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 QUADSPI mode is 1-0-0 (instruction on 1 line - no address - no data).

Figure 26. Write enable sequence (command 0x06)



Configuring dummy cycle

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

4.2.5 Starting a communication (QUADSPI_CCR register)

Once the QUADSPI peripheral is configured, the communication can be started in one of these modes:

- Indirect-write or Indirect-read mode
- Status-flag polling mode
- 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 3.1: Flexible frame format 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.

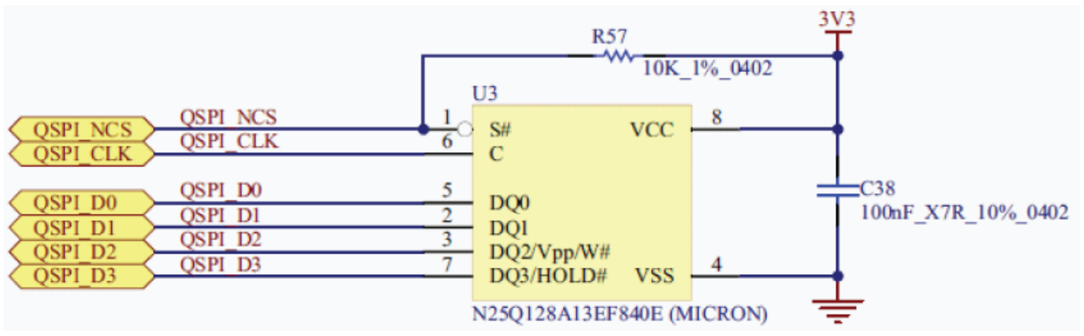
4.3 Hardware considerations

4.3.1 Pull-up resistance

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

The following figure shows an example of a Quad-SPI memory connection where a pull-up resistance is connected to the chip-select pin.

Figure 27. Connecting chip-select to a pull-up resistance



4.3.2 Good PCB design allows maximum QUADSPI speed

The speed at which the QUADSPI 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. QUADSPI availability and features across STM32 families 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, refer to the application note *Getting started with STM32F7 Series MCU hardware development* (AN4661) section “Quadrature serial parallel interface (Quad SPI)”, available on the STMicroelectronics website.

4.3.3 Chip-select high time (CSHT)

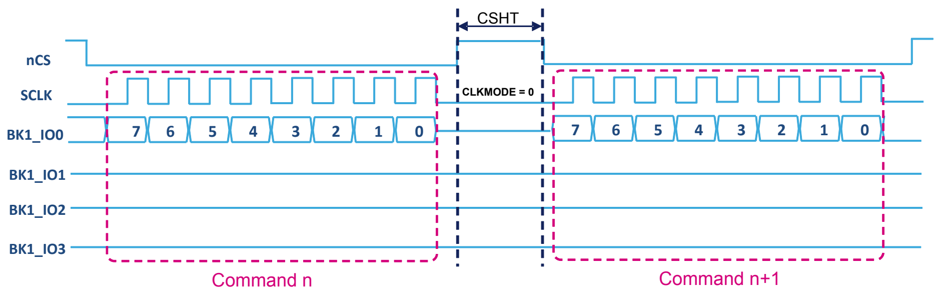
When the QUADSPI sends two commands, one immediately after the other, it raises the chip-select signal (nCS) *high*between the two commands for only one CLK cycle by default.

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.

4.3.4 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 28. Chip select high time: CSHT = two clock cycles



4.3.5 Some considerations when using QUADSPI in classical Single-SPI mode

When using the QUADSPI interface in Single-SPI mode, the user should consider the following equivalences:

- **Mode 0** for QUADSPI is equivalent to CPOL = 0 and CPHA = 0 for Single-SPI mode
- **Mode 3** for QUADSPI is equivalent to CPOL = 1, and CPHA = 1 for Single-SPI

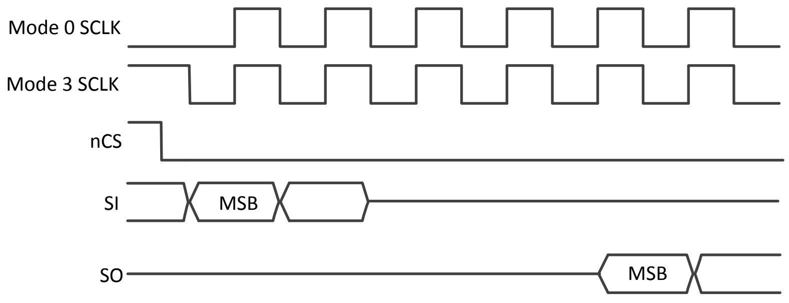
The main difference between the two modes is the clock polarity when the bus master is in Standby mode and not transferring any data.

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

Note: *Full duplex is not supported when using the QUADSPI interface in classical Single-SPI mode, only Half duplex is supported.*

The following figure shows a Single-SPI frame example highlighting the QUADSPI clock modes equivalence with Single-SPI.

Figure 29. QUADSPI in classical Single-SPI frame example



5 Programming Quad-SPI Flash memory

This section describes how to program a Quad-SPI Flash memory in the following use cases:

- **For an end application development**: in this case the Quad-SPI memory is programmed during the development of the product with static data or code to be used in the final product. A dedicated Flash memory loader is needed in order 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 Quad-SPI Flash memory is used in a final product as an external mass-storage device, this permits 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 should be used for programming regardless 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 to program the Quad-SPI Flashes with sizes up to 4Gbytes.

The Automatic-polling mode can be used for waiting while the programming operation is ongoing; when the operation is completed an interrupt can be generated.

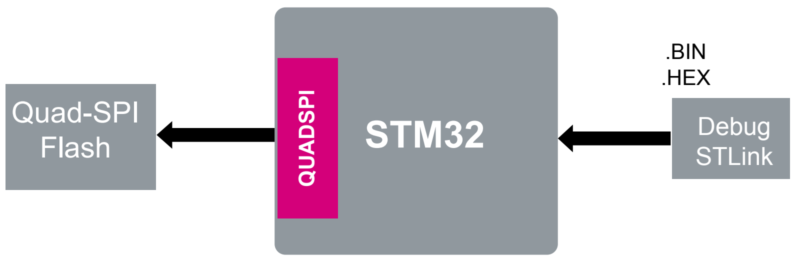
5.1 Programming code or data for an end application

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 Quad-SPI memory to be executed by the CPU and then to be extended to the on-chip internal memory. An example of storing code in Quad-SPI Flash memory is described in Section 6.2: Executing from external Quad-SPI memory: extend internal memory size.
- 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 Quad-SPI Flash memory for an end application is described in Section 6.1: Reading data from Quad-SPI memory: graphical application.

To program the Quad-SPI 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 device.

Figure 30. **Programming Quad-SPI memory through debug interface**



External Flash memory loader

A dedicated algorithm is used to perform the programming operation. The algorithm is loaded to the STM32 device through the debug interface then executed to perform the programming operation. The inputs for the algorithm are the binary file to be programmed.

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

5.1.1 Programming Quad-SPI Flash memory using the STM32 ST-LINK utility

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

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

For each hardware configuration and for each Quad-SPI 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 hardware used is other than 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 the user to develop an external loader for a N25Q256A Flash memory on the STM32L476G-EVAL board. This project can be easily tailored to the user dedicated hardware to generate the external loader.

For more details on how to develop an external Quad-SPI Flash memory loader for the STM32 ST-LINK utility, refer to the user manual *STM32ST-LINK Utility software description* (UM0892), section “Developing custom loaders for external memory” available at *www.st.com.*

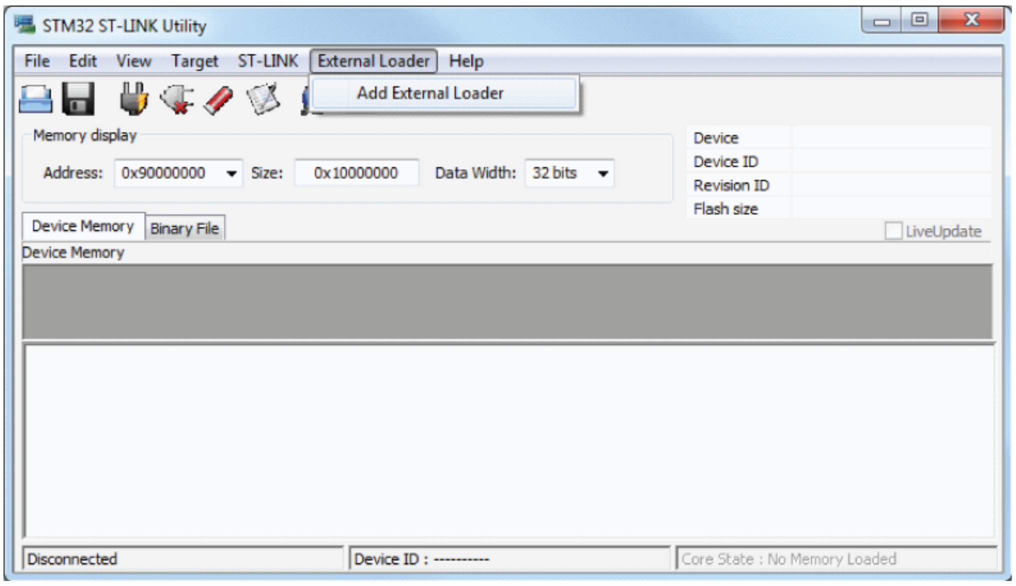
Note: *The tool chain/compiler used to generate the HEX/BIN file to program the Quad-SPI memory must be exactly the same as the one used for the application development.*

The dedicated Flash-memory loader has to be added to the ST-LINK utility in order to be able to program a Quad-SPI Flash memory.

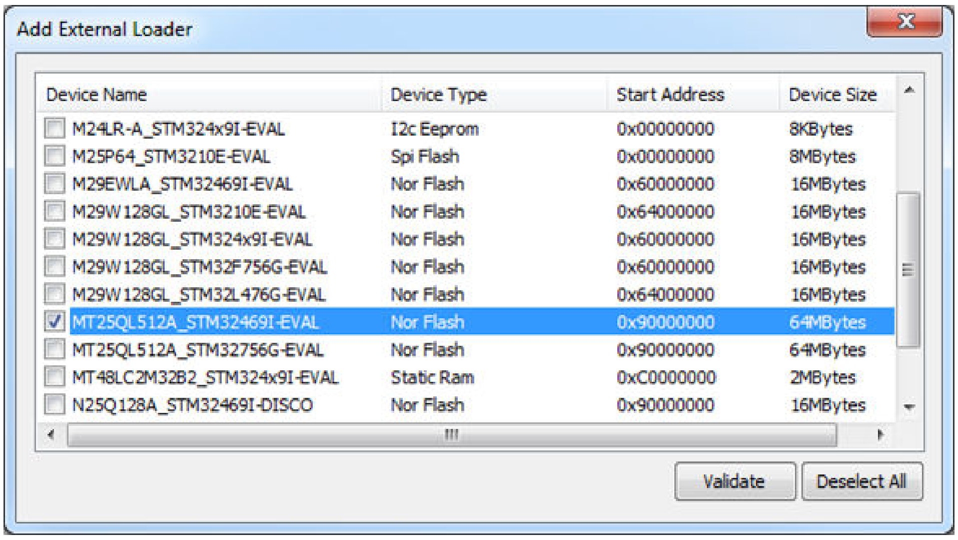
How to add a Quad-SPI Flash memory loader to the ST-LINK utility

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

Figure 31. STM32 ST-LINK utility: adding Quad-SPI Flash memory loader



A window appears where the user should select their device. See an example below:

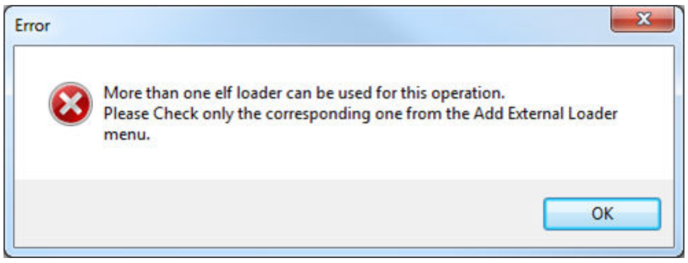


Add External Loader

Device Name	Device Type	Start Address	Device Size
☐ M24LR-A_STM324x9I-EVAL	I2c Eeprom	0x00000000	8KBytes
☐ M25P64_STM3210E-EVAL	Spi Flash	0x00000000	8MBytes
☐ M29EWLA_STM32469I-EVAL	Nor Flash	0x60000000	16MBytes
☐ M29W128GL_STM3210E-EVAL	Nor Flash	0x64000000	16MBytes
☐ M29W128GL_STM324x9I-EVAL	Nor Flash	0x60000000	16MBytes
☐ M29W128GL_STM32F756G-EVAL	Nor Flash	0x60000000	16MBytes
☐ M29W128GL_STM32L476G-EVAL	Nor Flash	0x64000000	16MBytes
✓ MT25QL512A_STM32469I-EVAL	Nor Flash	0x90000000	64MBytes
☐ MT25QL512A_STM32756G-EVAL	Nor Flash	0x90000000	64MBytes
☐ MT48LC2M32B2_STM324x9I-EVAL	Static Ram	0xC0000000	2MBytes
☐ N25Q128A_STM32469I-DISCO	Nor Flash	0x90000000	16MBytes

Figure 32. **STM32 ST-LINK utility: selecting Quad-SPI Flash memory loader**

Note: Only one external loader can be added, otherwise the error message in the figure below appears. The user can remove one external loader and replace it with another one if needed.



Error

More than one elf loader can be used for this operation.
Please Check only the corresponding one from the Add External Loader menu.

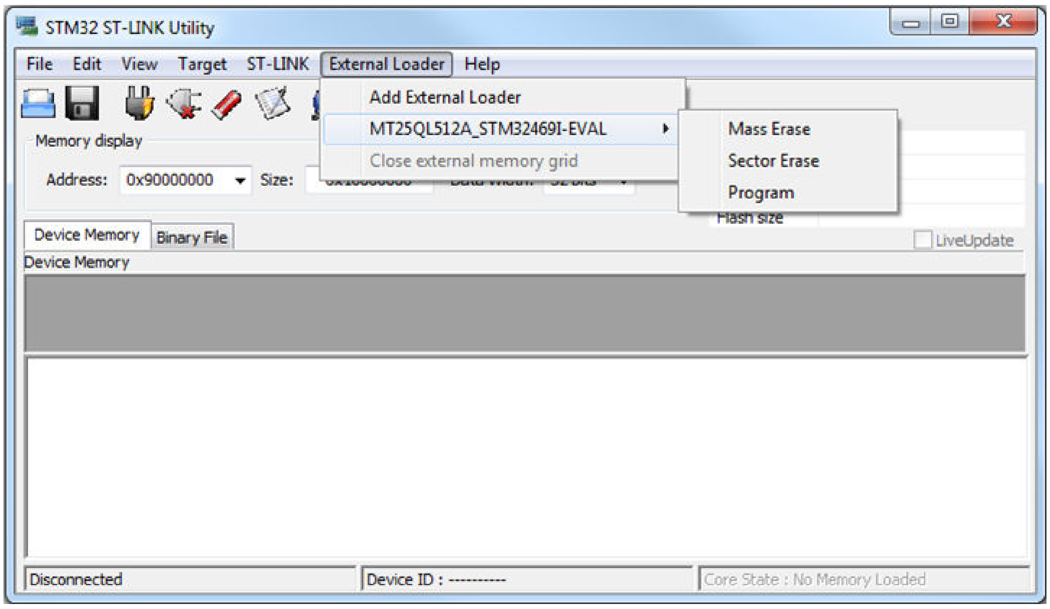
Figure 33. **STM32 ST-LINK utility: error message**

How to program Quad-SPI Flash memory using the ST-LINK utility

To program the Quad-SPI Flash memory follow the steps below:

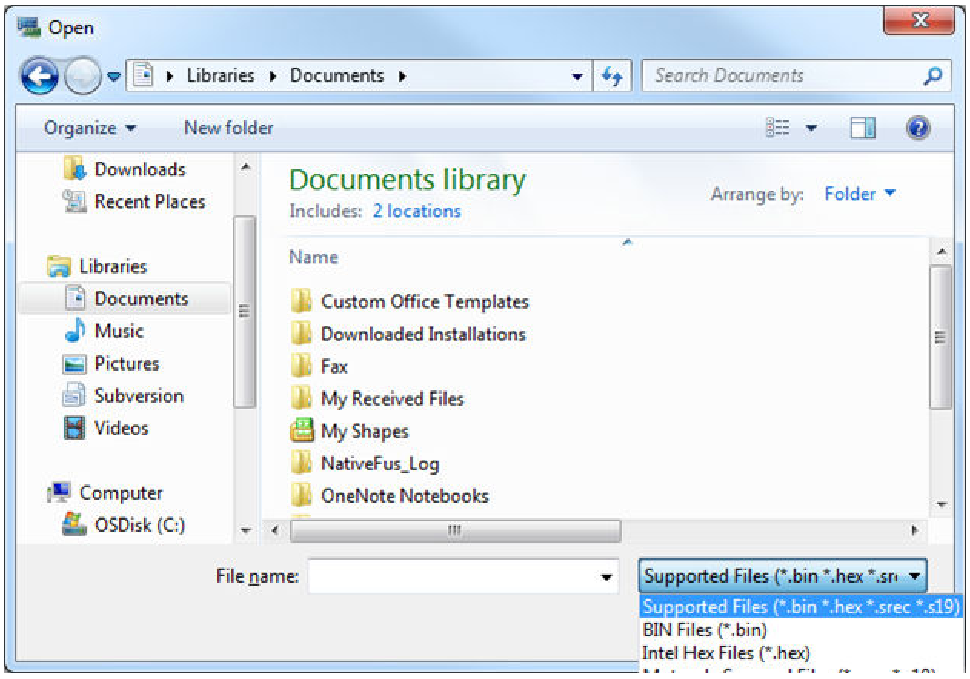
1. Connect the board using the USB cable through the ST-LINK debug.
2. In the *External Loader* go to the added external Flash memory loader then select Program as shown below.

Figure 34. STM32 ST-LINK utility: programming Quad-SPI Flash memory



3. The following window appears allowing the user to browse to the data file to be stored in the Flash memory, which can be a binary file, an HEX file or Motorola S-record files (.srec or .s19).

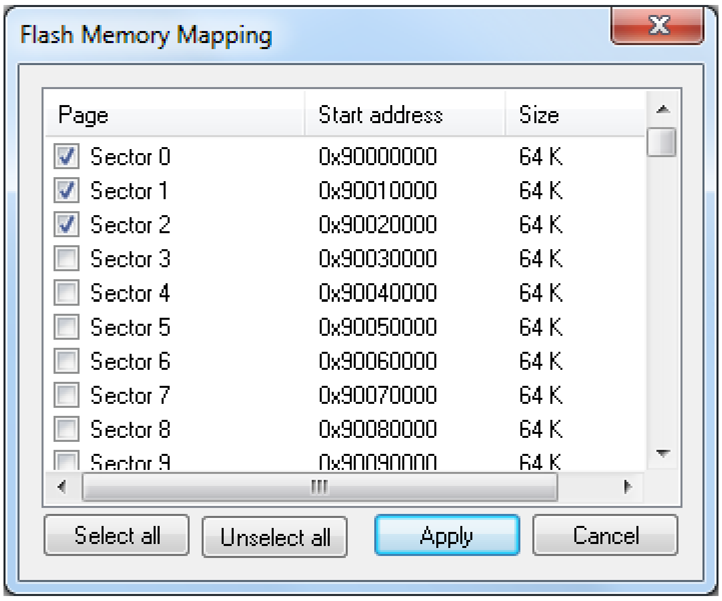
Figure 35. STM32 ST-LINK utility: selecting HEX file for programming



Erasing Quad-SPI memory

To perform a mass-erase operation select MassErase, to erase sectors click on Sector Erase then select the sectors to be erased as shown in the figure below.

Figure 36. **STM32 ST-LINK utility: erasing sectors**


Flash Memory Mapping

Page	Start address	Size
✓ Sector 0	0x90000000	64 K
✓ Sector 1	0x90010000	64 K
✓ Sector 2	0x90020000	64 K
☐ Sector 3	0x90030000	64 K
☐ Sector 4	0x90040000	64 K
☐ Sector 5	0x90050000	64 K
☐ Sector 6	0x90060000	64 K
☐ Sector 7	0x90070000	64 K
☐ Sector 8	0x90080000	64 K
☐ Sector 9	0x90090000	64 K

5.1.2 Programming Quad-SPI Flash memory using IDE

Programming Quad-SPI Flash memory using Keil

In the user project, the code or the data region to be programmed in the Quad-SPI memory have to be specified to the linker before programming.

The following example shows how to add a dedicated Quad-SPI memory load region in a Keil scatter file where also an execute region is created, enabling the execution of the code from the Quad-SPI memory.

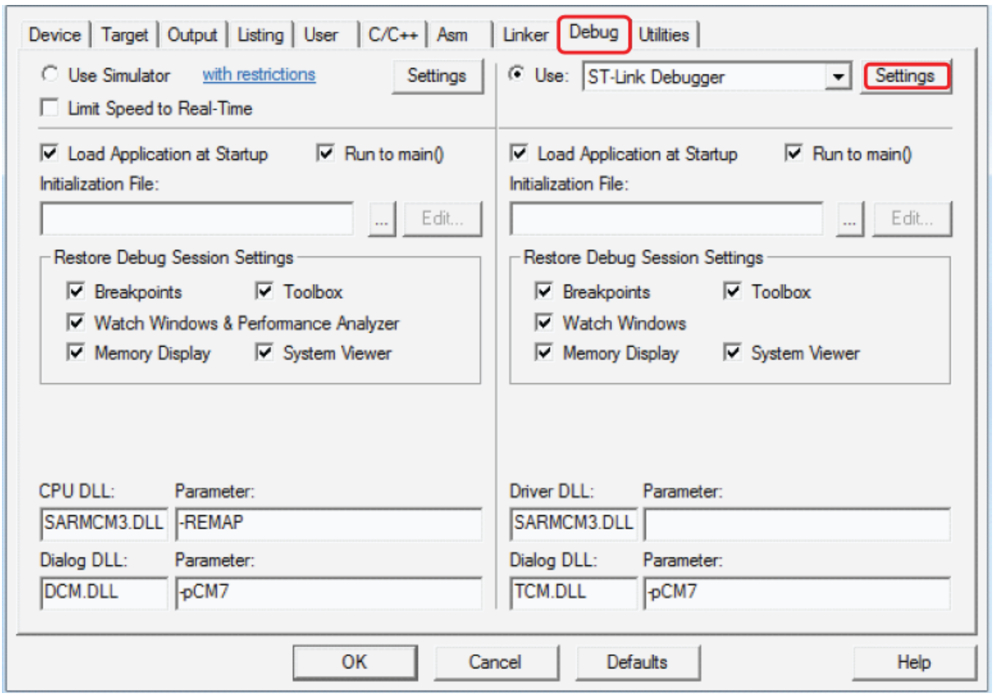
```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

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 0xFFFFFFF {
ER_QSPI 0x90000000 0xFFFFFFF {
*.o (.textqspi)
}
}
```

How to add a Quad-SPI Flash memory loader to Keil® MDK-Arm®

To add the Quad-SPI 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 the figure below.

Figure 37. Adding Quad-SPI Flash memory loader to Keil MDK-Arm project (1)



Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

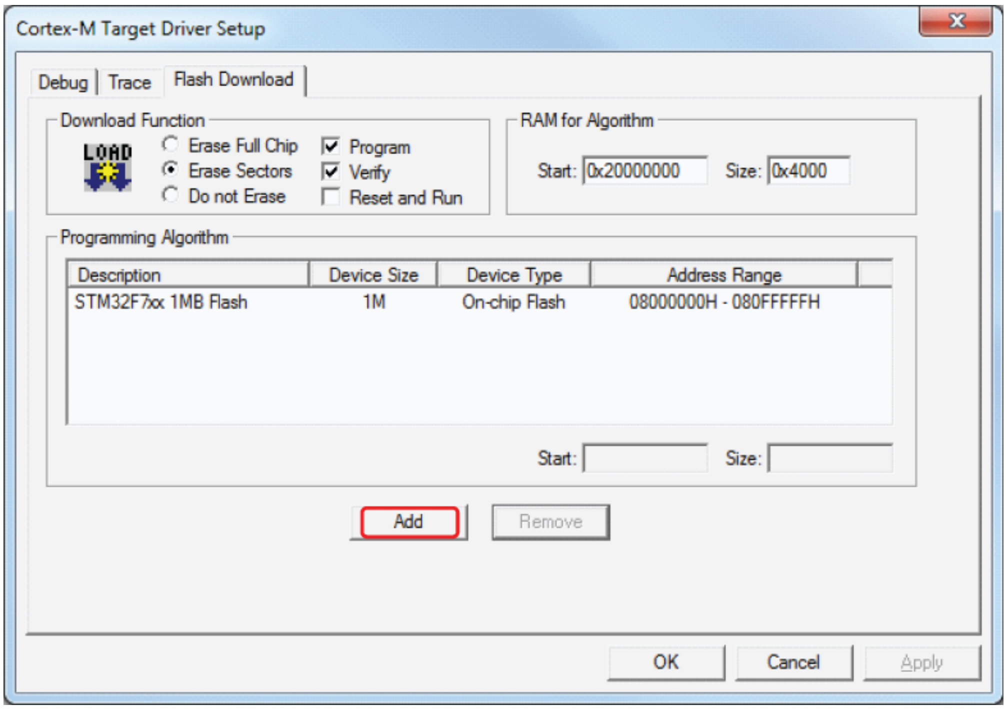
○ Use Simulator with restrictions Settings
☐ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter: -REMAP
Dialog DLL: DCM.DLL Parameter: -pCM7

● Use: ST-Link Debugger Settings
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display ☑ System Viewer
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TCM.DLL Parameter: -pCM7

OK Cancel Defaults Help

The Quad-SPI Flash memory loader is then added in the following window:

Figure 38. Adding Quad-SPI Flash memory loader to Keil MDK-Arm project (2)



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Download Function
○ Erase Full Chip ☑ Program
● Erase Sectors ☑ Verify
○ Do not Erase ☐ Reset and Run

RAM for Algorithm
Start: 0x20000000 Size: 0x4000

Programming Algorithm

Description	Device Size	Device Type	Address Range
STM32F7xx 1MB Flash	1M	On-chip Flash	08000000H - 080FFFFFH

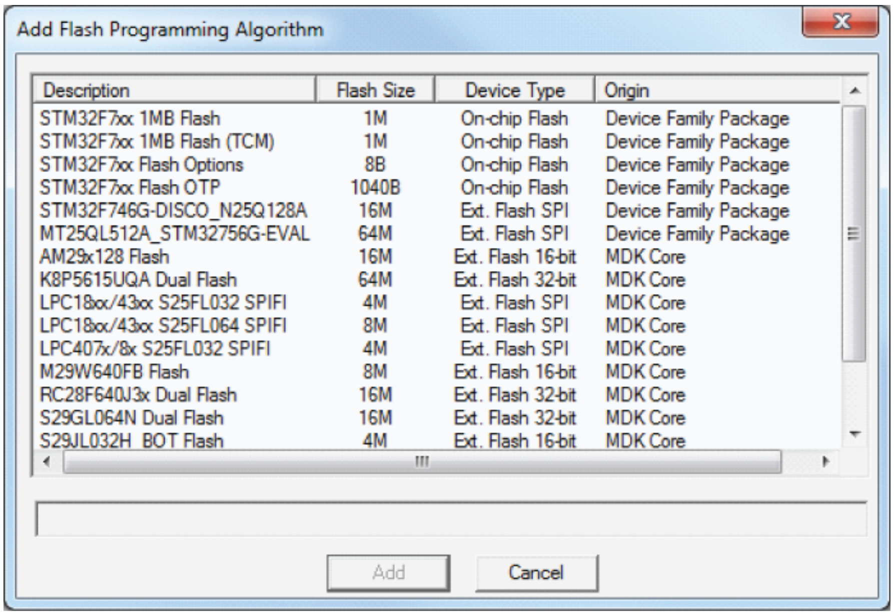
Start: Size:

Add Remove

OK Cancel Apply

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

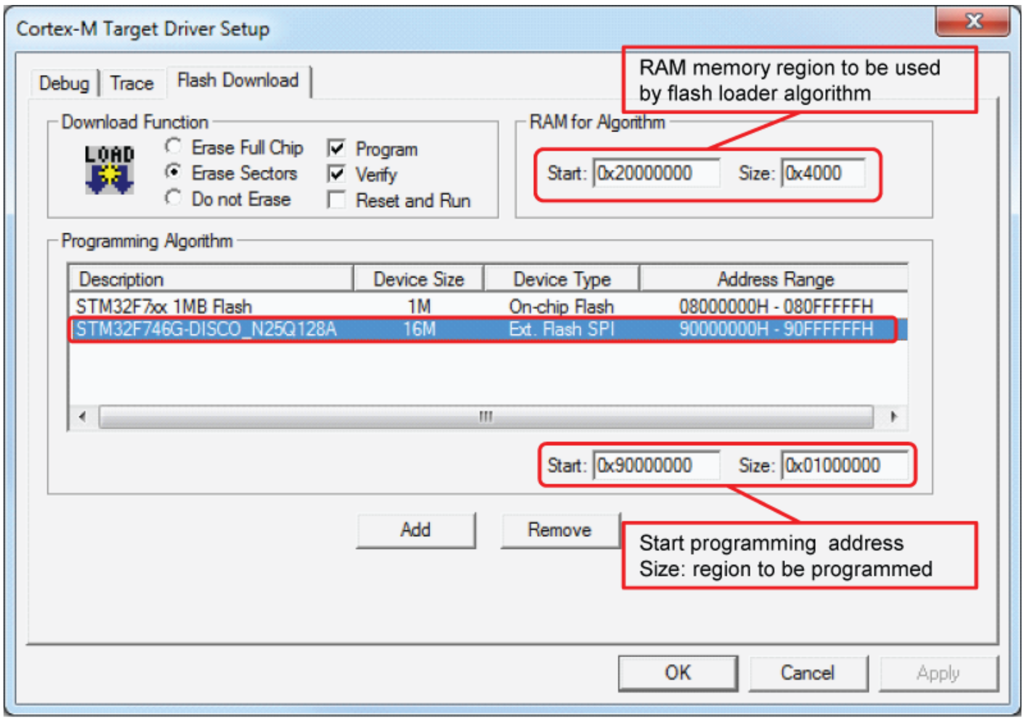
Figure 39. . Selecting Quad-SPI Flash memory programming algorithm



Description	Flash Size	Device Type	Origin
STM32F7xx 1MB Flash	1M	On-chip Flash	Device Family Package
STM32F7xx 1MB Flash (TCM)	1M	On-chip Flash	Device Family Package
STM32F7xx Flash Options	8B	On-chip Flash	Device Family Package
STM32F7xx Flash OTP	1040B	On-chip Flash	Device Family Package
STM32F746G-DISCO_N25Q128A	16M	Ext. Flash SPI	Device Family Package
MT25QL512A_STM32756G-EVAL	64M	Ext. Flash SPI	Device Family Package
AM29x128 Flash	16M	Ext. Flash 16-bit	MDK Core
K8P5615UQA Dual Flash	64M	Ext. Flash 32-bit	MDK Core
LPC18xx/43xx S25FL032 SPIFI	4M	Ext. Flash SPI	MDK Core
LPC18xx/43xx S25FL064 SPIFI	8M	Ext. Flash SPI	MDK Core
LPC407x/8x S25FL032 SPIFI	4M	Ext. Flash SPI	MDK Core
M29W640FB Flash	8M	Ext. Flash 16-bit	MDK Core
RC28F640J3x Dual Flash	16M	Ext. Flash 32-bit	MDK Core
S29GL064N Dual Flash	16M	Ext. Flash 32-bit	MDK Core
S29JL032H_BOT Flash	4M	Ext. Flash 16-bit	MDK Core

Once the corresponding Flash memory loader is added it appears in the programming algorithm list as shown in the figure below:

Figure 40. Quad-SPI Flash memory loader programming algorithm configuration



Download Function: ( ) Erase Full Chip [✓ Erase Sectors] ( ) Do not Erase; [✓] Program, [✓] Verify, [ ] Reset and Run

RAM for Algorithm: Start: 0x20000000 Size: 0x4000

Description	Device Size	Device Type	Address Range
STM32F7xx 1MB Flash	1M	On-chip Flash	08000000H - 080FFFFFH
STM32F746G-DISCO_N25Q128A	16M	Ext. Flash SPI	90000000H - 90FFFFFFH

Start: 0x90000000 Size: 0x01000000

Once the Quad-SPI Flash memory loader is added, programming can be done by clicking on the Load button or pressing the 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.

How to proceed to program Quad-SPI Flash memory only-once

During an application development, the user needs to debug its project and needs to load the code to the internal Flash memory many times. For each load-Flash operation, the Quad-SPI Flash memory is loaded as well, which makes the loading operation too long.

In this case, if the user already loaded data to the Quad-SPI Flash and does not need to repeat the operation, the user can simply generate the QUADSPI 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 Quad-SPI Flash memory loader.

STM32 system workbench

The STM32 system workbench does not support a Quad-SPI external Flash-loader, the user can use the STM32 ST-LINK utility instead (as described previously).

5.2 Storing and erasing data on the fly during running application

5.2.1 Storing data

It is useful in some applications to use the external Quad-SPI Flash memory for data storage. Int hat case, the Quad-SPI interface has to be configured in Indirect-write mode to permit the on-the-fly data storage. Data to be stored can be the result of a processing (like a 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 amount of time, the user can use Status-flag polling mode to poll the memory status register, and once the operation is competed, an interrupt can be enabled.

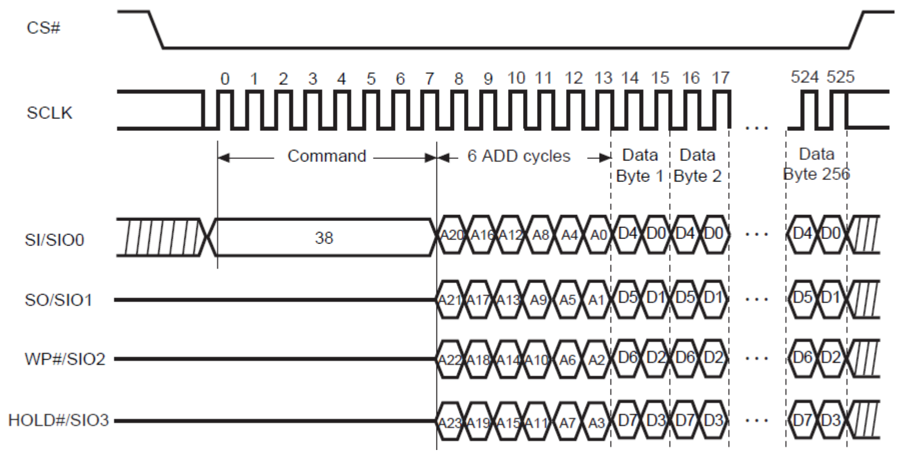
The steps to program an operation are:

1. Configure the Quad-SPI interface in the QUADSPI_CR and QUADSPI_DCR registers.
2. Configure the Flash memory: enable writing to Flash, set the dummy-cycles number depending on the clock speed.
3. Configure the frame format in the QUADSPI_CCR register and start the programming sequence.
4. After the programming sequence has finished, configure the QUADSPI in Automatic-polling mode to check the memory status and to confirm if it is ready.

The following figure shows an example of a page programming sequence where the page size is 256 bytes.

Note: Data byte 1 is written first to the data register QUADSPI_DR, then data byte 256 is the last. When writing a 32-bit word to QUADSPI_DR register, the LSB byte is written first to the FIFO then transmitted first.

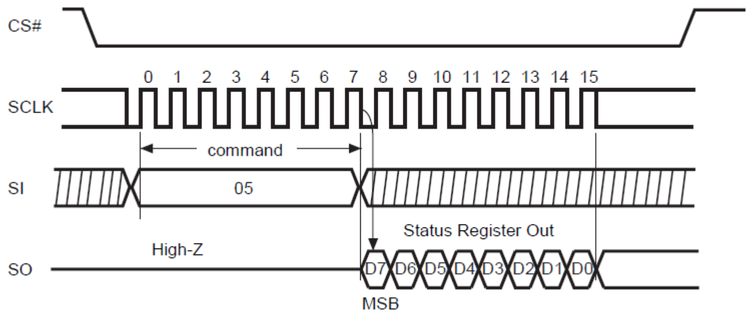
Figure 41. Quad I/O page program sequence (command 0x38)


CS#
SCLK
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
524 525
Command
6 ADD cycles
Data Byte 1
Data Byte 2
Data Byte 256
SI/SIO0
38
A20 A16 A12 A8 A4 A0 D4 D0 D4 D0 D4 D0
SO/SIO1
A21 A17 A13 A9 A5 A1 D5 D1 D5 D1 D5 D1
WP#/SIO2
A22 A18 A14 A10 A6 A2 D6 D2 D6 D2 D6 D2
HOLD#/SIO3
A23 A19 A15 A11 A7 A3 D7 D3 D7 D3 D7 D3

Quad-SPI memory can be programmed using a CPU with interrupts or using DMA.

1. Programming Quad-SPI memory using Indirect-write mode
   When using Indirect-write 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 the FIFO threshold is reached.
2. Programming Quad-SPI memory using Indirect-write mode with DMA
   It is generally recommended to use DMA to program the Quad-SPI memory using Indirect-write mode since it offloads the CPU, nevertheless the final recommendation depends on the user application. 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 has started, no intervention from the CPU is needed and the operation ends autonomously. For more details on DMA usage, refer to Section 3.5.2: DMA usage.
3. Usage of Status-flag polling mode.
   The user can use this mode to poll the memory status register. The figure below shows an example of a status-register reading sequence.

Figure 42. Read status register sequence (command 0x05)



5.2.2 Erasing data

An erasing operation must be performed before every programming operation. Indirect-write mode should be used for the erasing and the programming operations.

An erasing operation and a programming operation require the same configuration, except that for the erasing operation, no data has to be written to the memory. Only the instruction and the address phases are required (data phase is not needed).

As erasing operation takes a period of time, the user can use Status-flag polling mode to poll the memory status register. Once the operation is completed an interrupt can be enabled.

The steps to perform an erasing operation are listed below:

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

Most of the Flash memory devices support a sector erasing and a full-chip erasing operation; and some of them support an additional erasing operation offering more flexibility to users applications. Refer to the manufacturer’s datasheet for more details on the supported erasing operations.

Note: *The 4-bytes address mode must be used if the memory size is larger than 16 Mbytes. The user should choose the 4-byte command from the memory datasheet.*

Sector-erase sequence

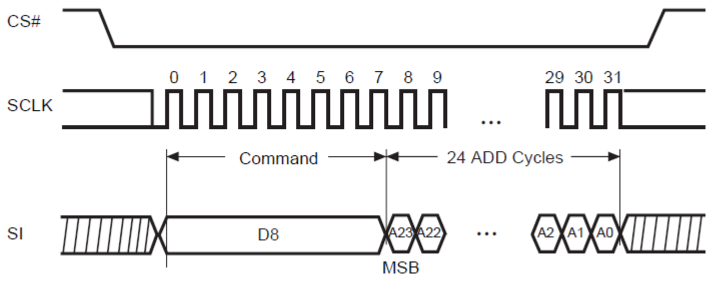
To erase a sector on the memory, a sector-erase command and a starting sector address should be sent.

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 */
```

See below an example of a sector-erasing sequence:

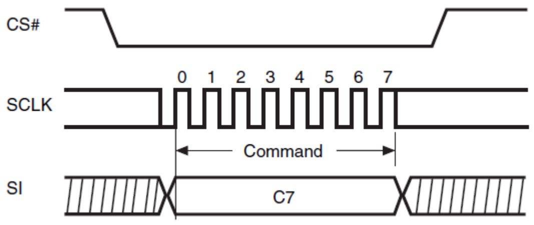
Figure 43. **Sector erase sequence**



Full-chip erase sequence (bulk erase)

The whole memory can be erased by sending a command (no need to send an address). See below an example of a chip-erase sequence:

Figure 44. **Example: full chip-erase sequence**



6 QUADSPI application examples

This section provides some typical QUADSPI use case examples that show 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.

This section describes the following use cases:

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

6.1 Reading data from Quad-SPI memory: graphical application

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

- Frame buffer content generation from the Quad-SPI memory
- Displaying images directly from the Quad-SPI memory

6.1.1 Frame buffer content generation from Quad-SPI memory

This section provides an example where the DMA2D peripheral is reading images stored in the external Quad-SPI 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 STM32CubeF7 firmware package available on the ST website. This example includes one project that has been developed for the STM32F746G-DISCO board.

The project is located on the following path: `STM32Cube_FW_F7_V1.14.0\Projects\ STM32746G-Discovery\Applications\QSPI\QSPI_perfs`.

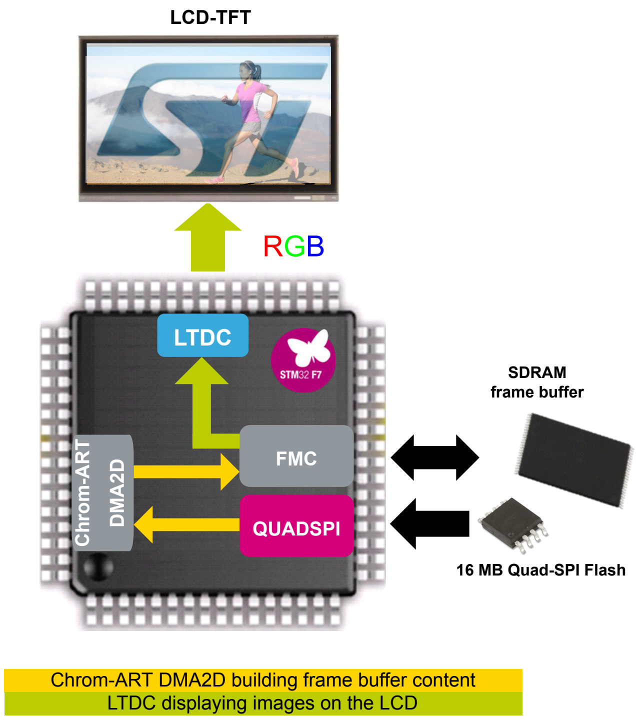
Use case description

This example describes how to use pictures stored on the Quad-SPI 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 Quad-SPI Flash memory. The DMA2D is used to transfer pictures from the Quad-SPI 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 60 Hz.

Figure 45. QUADSPI usage in a graphical application



Storing images and icons to be used for the application

All images and icons that are needed for the application should be stored on the external Quad-SPI memory. Six images and three icons are available in the project, all of them are included in the "`main images.c`" file, where each image 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 Quad-SPI Flash memory.

The icons and the images are defined as constants. In order to store these icons and images to the Quad-SPI memory, they are placed in a dedicated section ".textqspi" as described below:

- Img2and 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 5.1.2: Programming Quad-SPI Flash memory using IDE, to program the Quad-SPI memory using Keil MDK-Arm or IAR EWARM, a dedicated section for all the data to be programmed should be created.

Following the QUADSPI section ".textqspi" in the Keil MDK-Arm scatter file:

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

To program data to Quad-SPI Flash, the user should first check if the Quad-SPI 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 refer to the readme file in the project directory.

Quad-SPI interface and memory configuration

The QUADSPI is configured in Memory-mapped mode to permit the DMA2D to read images and icons from the QUADSPI and to write into the SDRAM through the FMC interface.

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

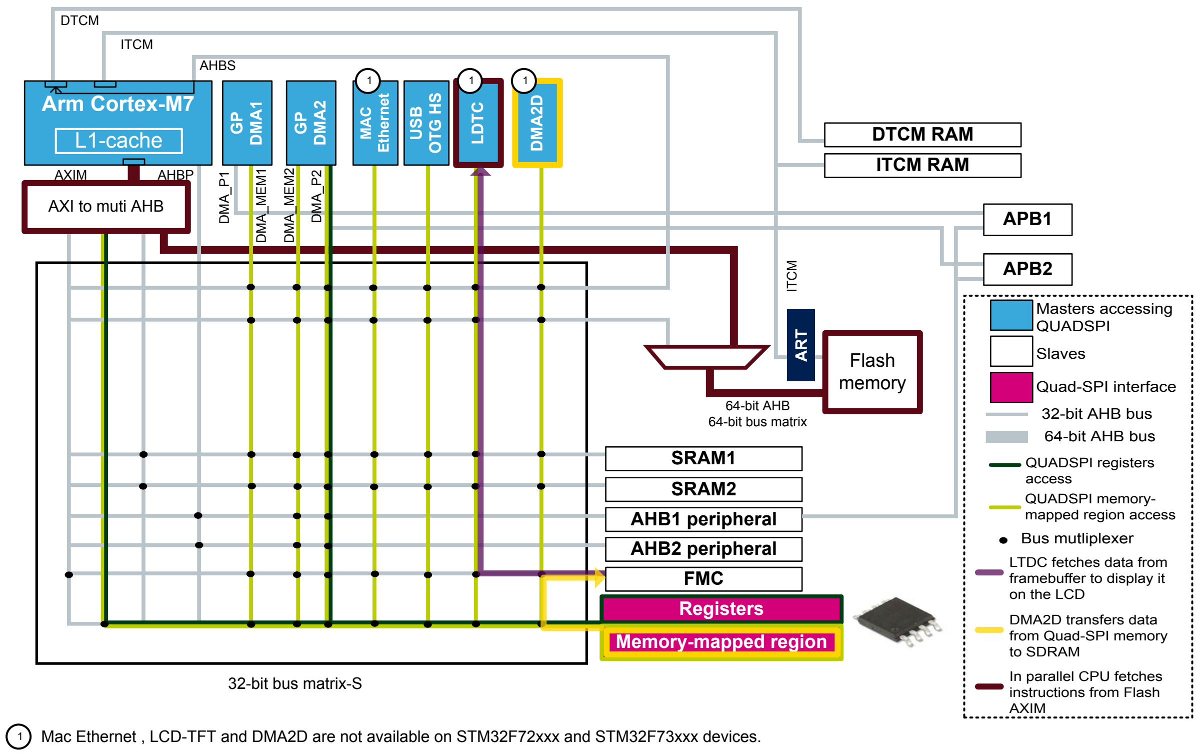
The used read command is the QUAD INPUT/OUTPUT FAST READ (0xEB) which permits to send the address in four lines and read data in four lines while the command is sent in one line.

The STM32F7 series smart architecture permits an offload of the CPU. The DMA2D acts as an AHB master and performs all transfers from the QUADSPI 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 16-Mbyte external Flash memory with the STM32F7x6 device.

The Quad-SPI Flash memory is used as a non-volatile support containing all graphics (icons, images) needed in the application.

Figure 46. **DMA2D reading images from Quad-SPI to build frame buffer content**



6.1.2 Displaying images directly from the Quad-SPI memory

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

This section provides an example where the LTDC peripheral reads an image stored in the external Quad-SPI 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.

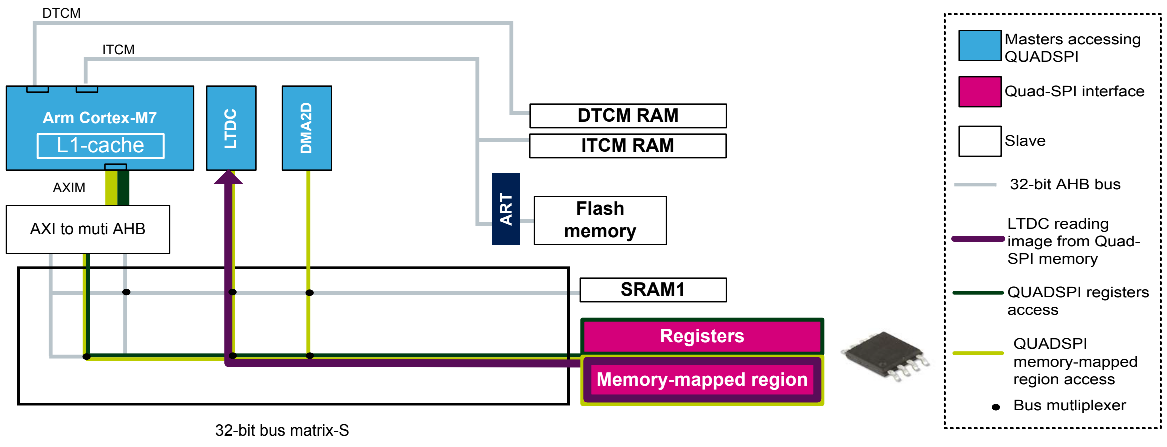
Note: This application note describes only the QUADSPI usage in the use case.

Software demonstration description

An image "screensaver" is located in three different regions: the internal Flash memory, the external Quad-SPI Flash memory at the address 0x9000 0000 and the SDRAM at the address 0xC000 0000. The user must choose from which region the LTDC should display the image.

At first run, the time is displayed on the LCD-TFT for five seconds, then a menu appears showing a box where the user can choose the memory location from which the image is read. If the Quad-SPI Flash memory is selected, then the LTDC reads the image directly from the external Flash memory. See a graphical view of this use case below.

Figure 47. **LTDC reading an image directly from Quad-SPI 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_qaspi section */
const unit16_t RGB565_480x272_QSPI[]={};
```

As explained in Section 5.1.2: Programming Quad-SPI Flash memory using IDE, to program theQuad-SPI memory using Keil MDK-Arm or IAR EWARM, a dedicated section for all the data to be programmed should be created.

Following the QUADSPI section ".RGB565_480x272_qspi" in the Keil MDK-Arm scatter file:

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

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

Quad-SPI interface and memory configuration

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

Once the QUADSPI is configured, to display the image stored in the Quad-SPI memory, the following API is 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 Quad-SPI memory.

6.2 Executing from external Quad-SPI memory: extend internal memory size

Using the external Quad-SPI memory permits an extension of the total available memory space of the application. The STM32F746 device embedded on the Discovery board or the Evaluation boards boards integrates a 1-Mbyte Flash memory; hence connecting an external Quad-SPI Flash memory extends the available memory space to 64 Mbytes for the evaluation board.

This section describes how to use the external Quad-SPI memory to extend the internal Flash memory to enable code execution from the external Quad-SPI 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 Quad-SPI in Memory-mapped mode during the system initialization and before jumping to the Quad-SPI memory code
- Place application’s code in the external Quad-SPI 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. This section focuses on STM32F7_performances project.

The STM32F7_performances project shows STM32F746 device 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 this section focus on describing the code execution from the Quad-SPI memory, only 6_1-Quad-SPI_rwRAM- DTCM and 6_2-Quad-SPI_rwRAM-DTCM configurations are taken into consideration for the demonstration description.

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 hyper terminal through the UART or on the IDE printf viewer.

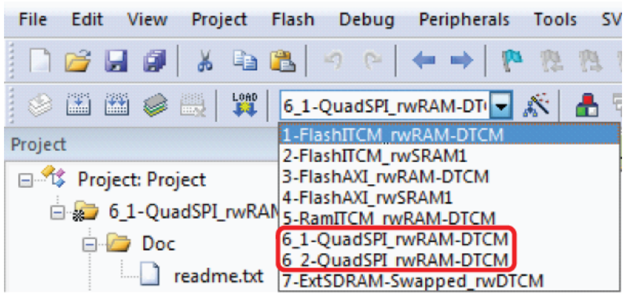
The configuration is displayed and shows the current project configuration, the system frequency, the different configurations of caches, ART, ART-Prefetch (ON/OFF) and the memory configuration in case of an external memory (SDRAM or Quad-SPI).

The software demonstration is developed for STM32756G-EVAL board and can be easily tailored to the STMF746-DISCO board. For more details on this application note 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_V4.0.0\Projects\STM32756G_EVAL\stm32f 7_performances`.

The figure below highlights the two project configurations in Keil MDK-Arm that are described in this document.

Figure 48. Project configurations: executing code from Quad-SPI Flash memory



Projects configuration

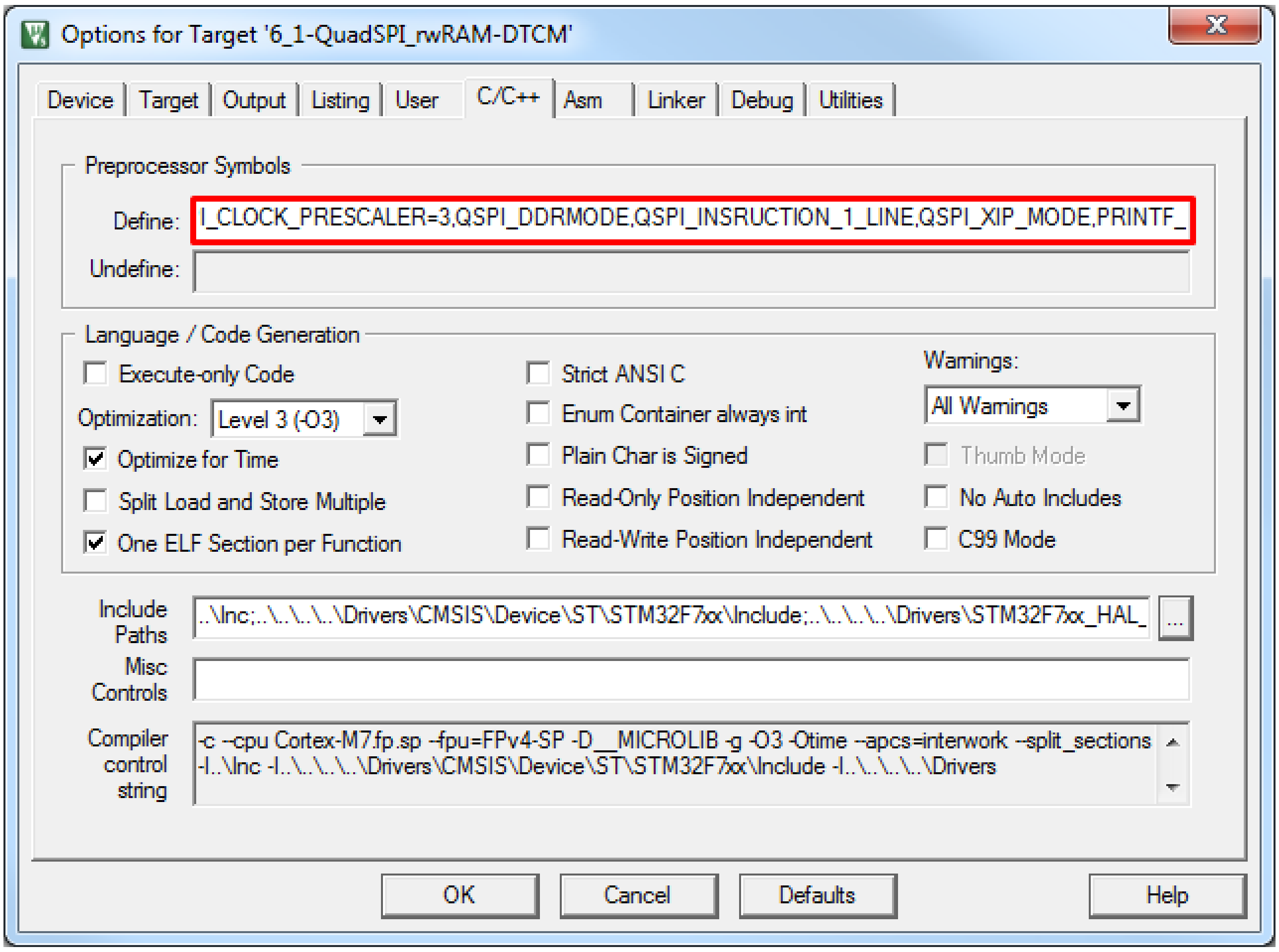
For both project configurations, 6_1-QuadSPI_rwRAM-DTCM and 6_2-QuadSPI_rwRAM- DTCM, the user can change the desired QUADSPI settings in the Options for Target box as shown in the next figure.

The operating system clock during system initialization is 16 MHz, so at this moment the QSPI_CLK = fAHB/1= 16 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 QUADSPI settings:

- QSPI_CLOCK_PRESCALER= 3
- 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 49. **Changing QUADSPI configuration in the project settings**



Define: I_CLOCK_PRESCALER=3,QSPI_DDRMODE,QSPI_INSRUCTION_1_LINE,QSPI_XIP_MODE,PRINTF_

Undefine:

Execute-only Code
Optimization: Level 3 (-O3)
[✓ Optimize for Time]
Split Load and Store Multiple
[✓ One ELF Section per Function]
Strict ANSI C
Enum Container always int
Plain Char is Signed
Read-Only Position Independent
Read-Write Position Independent
Warnings: All Warnings
Thumb Mode
No Auto Includes
C99 Mode

Include Paths: ..\Inc;..\..\..\..\Drivers\CMSIS\Device\ST\STM32F7xx\Include;..\..\..\..\Drivers\STM32F7xx_HAL_

Misc Controls:

Compiler control string:

```
-c --cpu Cortex-M7.fp.sp -fpu=FPv4-SP -D__MICROLIB -g -O3 -Otime --apcs=interwork --split_sections
-I..\Inc -I..\..\..\..\Drivers\CMSIS\Device\ST\STM32F7xx\Include -I..\..\..\..\Drivers
```

6.2.1 Configuring Quad-SPI in Memory-mapped mode during system initialization

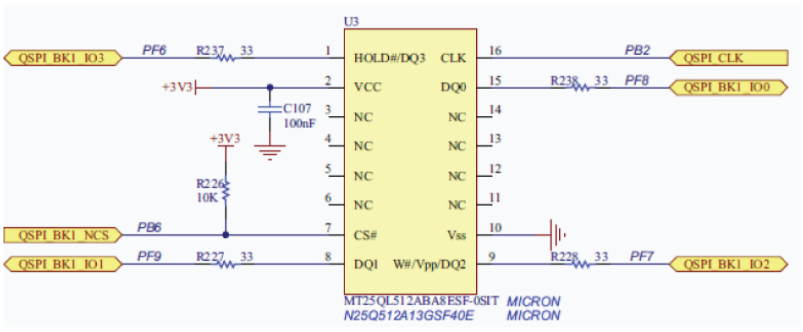
Boot from Quad-SPI Flash memory is not supported, so the user can boot from internal Flash memory, configure the QUADSPI peripheral in Memory-mapped mode and then jump to execution from the external Quad-SPI memory.

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

GPIOs configuration

As shown in the following figure, the Quad-SPI Flash memory is connected in Quad I/O mode, so six GPIOs have to be configured for the Quad-SPI interface.

Figure 50. Quad-SPI Flash memory connection in STM32756-EVAL board



The Quad-SPI interface GPIOs configuration in the SystemInit_ExtmemCtl() function is described below:

```
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 QUADSPI peripheral

In order to configure the Quad-SPI memory, the QUADSPI peripheral should be enabled so it can communicate with the external memory.

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

Quad-SPI memory configuration

Once the QUADSPI peripheral is enabled, it is possible to communicate with the Quad-SPI 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.

The indirect Quad-SPI mode has to be used for external memory configuration.

- Resetting the Quad-SPI memory:
  Before resetting the Quad-SPI memory registers, the RESET ENABLE command 0x66 should be sent in 1-0-0 mode, so 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 Quad-SPI memory registers, the RESET command 0x99 should be issued in 1-0-0 mode, 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:
  In both project configurations, the Quad-SPI memory is configured to receive commands in one line (QSPI_INSTRUCTION_1_LINE defined), but the memory can be configured to receive the commands in four lines. If this is the desired mode, the user should use the QSPI_INSTRUCTION_4_LINESdefine instead of the QSPI_INSTRUCTION_1_LINE.
  To enable the QUADSPI 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 this case, the QUADSPI 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 writen 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 */
  ```

QUADSPI peripheral configuration

- Configure QUADSPI peripheral (QUADSPI_CCR register):
  Once the Quad-SPI memory is configured, the Quad-SPI interface should be configured in 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 Quad-SPI 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.
  Only seven dummy-cycles are configured for QUADSPI peripheral, however it is eight cycles on the Quad-SPI 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, there are 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 QUADSPI peripheral (QUADSPI_CR register)
  The user can choose to operate either in DDR or in SDR mode, depending on the project configuration. The configuration 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 Quad-SPI 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
```

6.2.2 Placing application code in external Quad-SPI memory

The code to be loaded in the Quad-SPI 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.

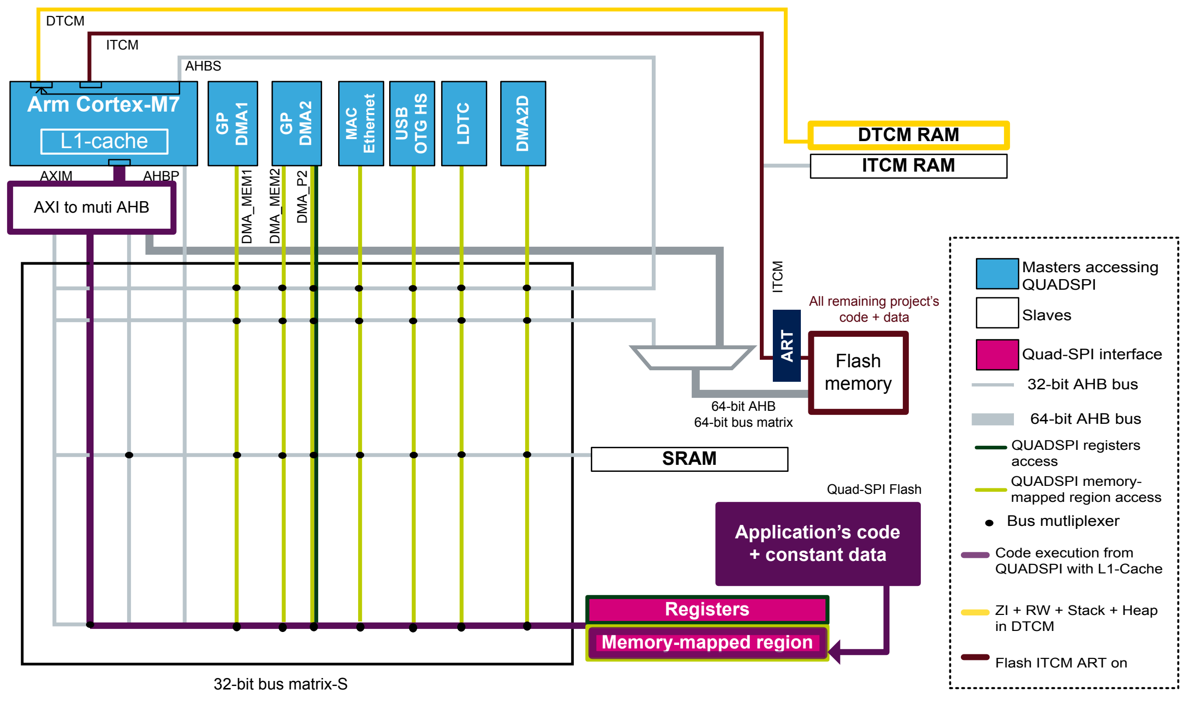
To place this code in the external Quad-SPI memory a dedicated load region should be created in the linker file of the project. Two project configurations are available in the project: one where the code of the application and the constant data are both placed in the Quad-SPI memory, and the other where the code of the application is placed in theQuad-SPI 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 Quad-SPI memory: 6_1-Quad-SPI_rwRAM-DTCM

In this project configuration, the application code and its related constant data are both placed in the Quad-SPI 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. The following figure describes the 6_1-Quad-SPI_rwRAM-DTCM project configuration.

Figure 51. 6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in Quad-SPI memory



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

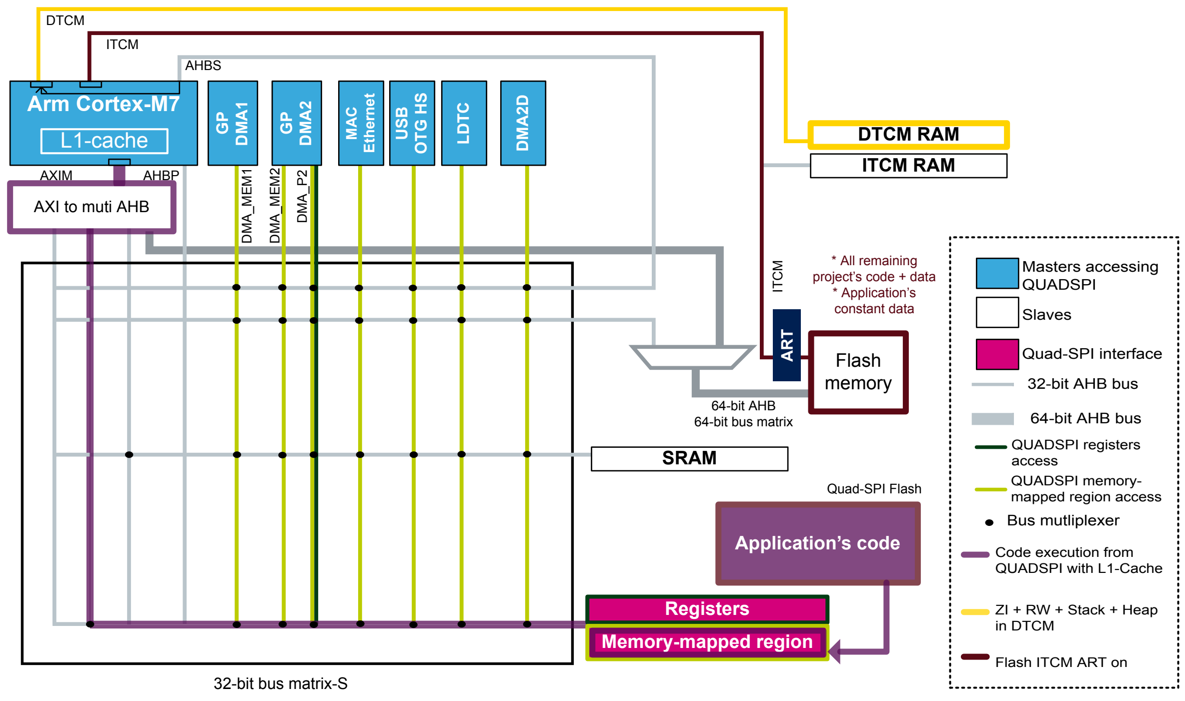
```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
LR_IROM1    0x00200000 0x00100000    {    ; load region size_region
    ER_IROM1    0x00200000 0x00100000    {    ; 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_f
ft_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 Quad-SPI 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 Quad-SPI memory while its related constant data is placed in the Flash ITCM. The Cortex®-M7have to fetch code from the Quad-SPI 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. The figure below describes the 6_2-Quad-SPI_rwRAM-DTCM project configuration.

Figure 52. 6_2-Quad-SPI_rwRAM-DTCM project configuration: only code in Quad-SPI memory



To place code and constant data in the Quad-SPI 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_f
ft_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)
}
}
```

Performance 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 QUADSPI is configured in DDR 1-4-4 mode with SIOO enabled and QSPI_CLK = 54 MHz.

The table below shows the obtained results for FFT demonstration for MDK-ARM in each configuration.

Table 15. **Execution performances versus configuration**

Feature configuration	Memory location configuration	CPU cycle number[1]
-	5-RAMITCM_rwRAM-DTCM	112428
I-cache + D-cache ON (constant data in Quad-SPI memory)	6_1-Quad-SPI_rwRAM-DTCM	171056
I-cache+ ART + ART-PF ON (constant 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 Quad-SPI 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.

By comparing 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), it is seen that it is they are close in terms of performances.

This is an example of how important it is to benefit from the STM32F7x5/F7x6 smart architecture (in this example) in order to improve the execution performances from the external Quad-SPI memory. For more details on how to improve the execution performances from Quad-SPI memory, refer to Section 7.1: How to get the best performances.

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

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

The STM32Cube firmware package provides two examples of reading and programming the Quad-SPI 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. This section describes 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 is to be programmed.*

6.3.1 QUADSPI indirect write: programming Quad-SPI memory using DMA

This section describes the STM32756G-EVAL example which is available on the STM32CubeF7 in the “Projects\STM32756G_EVAL\Examples\QSPI\QSPI_ReadWrite_DMA” directory.

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

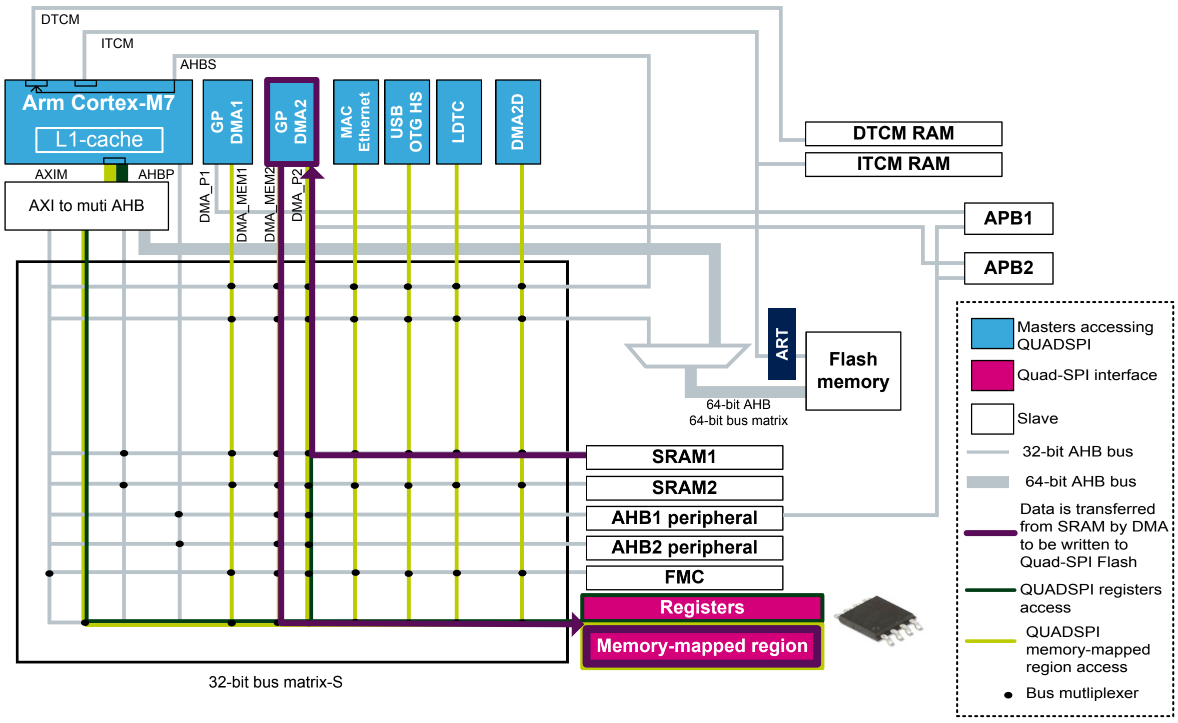
In this example, the following sequence is done in a forever loop:

1. Sector 1 of the Quad-SPI 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.

This section focus on writing to the Quad-SPI memory. Thanks to the STM32F7x5/F7x6 smart architecture, the DMA can be used to program or to read the Quad-SPI 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 the figure below, the DMA reads the data “aTxBuffer” from the SRAM and writes it to the Quad-SPI memory, in the meanwhile the CPU can execute code from the internal Flash.

Figure 53. **Indirect write mode: programming Quad-SPI memory using DMA**



QUADSPI GPIO and DMA configuration

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

The QUADSPI global interrupt is enabled. The GPIOs configuration is done with respect to the Quad-SPI 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.

QUADSPI peripheral configuration

The QUADSPI 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 memory size is set in QUADSPI_DCR register: the memory size is 64 Mbytes = 2[FSIZE+1] = 2[19+1] so FSIZE = 0x19.
- Chip-select high time (CSHT) is set to one cycle.
- QUADSPI peripheral is enabled by setting EN bit in QUADSPI_CR register.

Quad-SPI 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. Number of bytes to be written = QUADSPI_DLR+1.

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 (S7NDTR = QUADSPI_DLR+1)
- The DMA transfer is enabled by setting the DMAEN bit in the QUADSPI_CR register.

The address used in the QUADSPI_AR register is 0x0000 0000 as writing is performed in the first sector. In this way the programming sequence is immediately started once DMAEN bit is set.

Since DMA transfer to the FIFO is faster than the QUADSPI bus, the QUADSPI controls the transfer flow by setting the FIFO threshold flag (FTF) each time there is (FTHRESH + 1) free bytes available to be written to the FIFO. DMA transfers are initiated only when FTF flag is set.

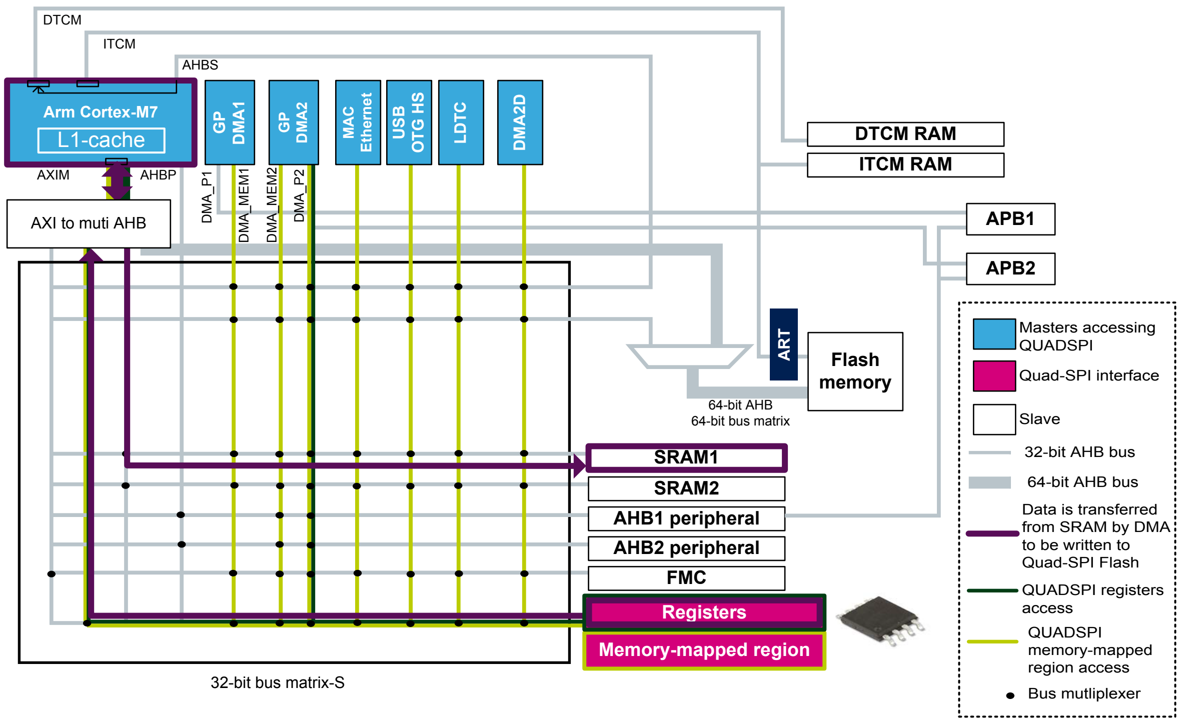
6.3.2 QUADSPI indirect write: programming Quad-SPI memory using interrupts

This example is available for all the STM32 embedding a QUADSPI interface. The applicable products are listed in Table 2. QUADSPI availability and features across STM32 families.

This section describes the STM32756G_EVAL example which is available on the STM32CubeF7 in the directory `Projects\STM32756G_EVAL\Examples\QSPI\QSPI_ReadWrite_IT`.

This example shows how to program theQuad-SPI 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 54. **Indirect write mode: programming Quad-SPI memory using interrupt**



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

Quad-SPI memory confirguration

The QUADSPI 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 => The Quad-SPI interrupts the CPU to write into the FIFO. Each time FTF flag is set (each time there is (FTHRESH + 1), free bytes available to be written to the FIFO).
- Sample shifting delay is disabled.
- Flash size is set in QUADSPI_DCR register: the memory size is 64 Mbytes = 2[FSIZE+1] = 2[19+ 1] so FSIZE = 0x19.
- Chip-select high time (CSHT) is set to one cycle.
- QUADSPI peripheral is enabled by setting EN bit in QUADSPI_CR register.

QUADSPI peripheral 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 QUADSPI transfer error TEIE, FIFO threshold FTIE and transfer complete TCIEInterrupts

Note that the used address is 0x0000 0000 in the QUADSPI_AR register as writing is performed in the first sector.

Once that the FTF flag is set, an interrupt is generated to the CPU. The CPU jumps from the main code to the interrupt routine HAL_QSPI_IRQHandler() located in stm32f7xx_hal_qspi.c and checks the source of the interrupt, then the CPU starts transferring data to the FIFO. Once finished, the CPU clears the FTF flag, exits the HAL_QSPI_IRQHandler() and returns to the main code.

6.4 Erasing-data example

This section describes how to erase the Quad-SPI Flash memory. As previously mentioned, the Indirect mode should be used for Quad-SPI Flash memory erasing.

All the provided Quad-SPI 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`.

Quad-SPI memory configuration

Before performing an erasing operation, a write-enable command has 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 = 0x0000 0000.

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.

6.5 Hardware implementation example

This section provides some hardware implementation examples of connecting the Quad-SPI Flash memory to the STM32 based on the existing ST Discovery and Evaluation boards. The following table summarizes the different STM32 boards embedding Quad-SPI Flash memory.

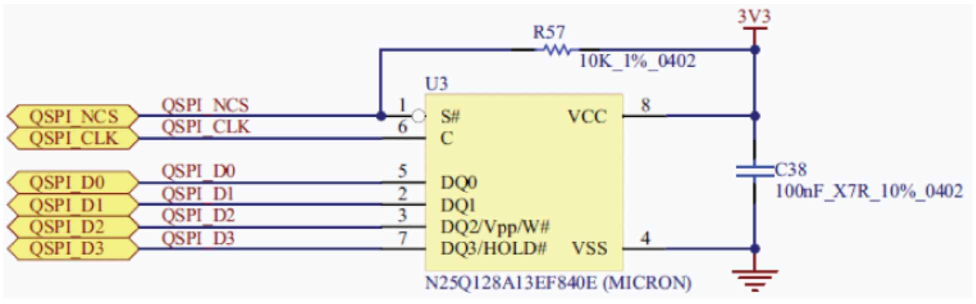
Table 16. Different STM32 boards embedding Quad-SPI Flash memory

Product families	Board	Quad-SPI Flash model	Size (Mbytes)
STM32L475	B-L475E-IOT01A	MX25R6435F	8
STM32L476	STM32L476G-DISCO	N25Q128A13EF840E	16
	STM32L476G-EVAL	N25Q256A13EF840E	32
STM32L496	STM32L496G-DISCO	MX25R6435FM2IL0	8
STM32F412	STM32F412G-DISCO	N25Q128A13EF840F	16
STM32F413	STM32F413H-DISCO	-	16
STM32F446	STM32446E-EVAL	N25Q256A13EF840E	32
STM32F469	STM32F469I-DISCO	N25Q128A13EF840F	16
	STM32469I-EVAL	MT25QL512ABA8ESF-0SIT	64
STM32F479	STM32479I-EVAL	MT25QL512ABA8ESF-0SIT	64
STM32F723	STM32F723e-DISCO	MX25L51245G	64
STM32F746	STM32F746G-DISCO	N25Q128A13EF840E	16
	STM32746G-EVAL	MT25QL512ABA8ESF-0SIT	64
STM32F750	STM32F7508-DISCO	-	16
STM32F756	STM32756G-EVAL	N25Q512A13GSF40E	64
STM32F769	STM32F769I-EVAL	MT25QL512ABA8ESF-0SIT	64
	STM32F769I-DISCO	MT25QL512ABB1EW9 /MX25L51245G	64
STM32F779	STM32F779I-EVAL	N25Q512A13GSF40E	64
STM32H743/STM32H753	STM32H7xxI-EVAL	MT25TL01GHBB8ESF-0SIT	128

STM32F746G-DISCO discovery board

The figure below shows an example of a connected MICRON Quad-SPI Flash memory in QuadI/O mode on the STM32F746G-DISCO discovery board.

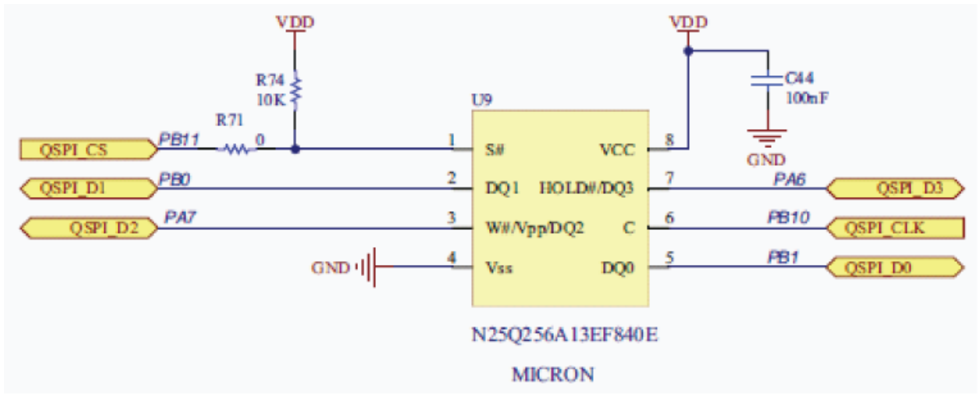
Figure 55. Quad-SPI memory connection on the STM32F746G-DISCO discovery board



STM32L476G-EVAL board

The figure below shows an example of how to connect MICRON Quad-SPI Flash memory in QuadI/O mode on the STM32L476G-EVAL discovery board.

Figure 56. Quad-SPI memory connection on the STM32L476G-EVAL board



7 Performance and power

This section presents some recommendations on how to improve performance and how to decrease power consumption for applications using the Quad-SPI interface.

7.1 How to get the best performances

7.1.1 Write performance

Since the writing speed of the Quad-SPI Flash memories is considerably low, there is limited benefit when optimizing the transmission speed. It is beneficial to use burst (page programming) writing to reduce the command overhead. In addition, DMA can be used to offload the CPU.

7.1.2 Read performance

This section describes how to get the optimum reading performances using QUADSPI peripheral features by following the described recommendations.

Configure QUADSPI at maximum speed

One of the most important parameters that permits to boost the read performances is the QUADSPI clock that should be as high as possible. As previously mentioned (see Section 4.3.2: Good PCB design allows maximum QUADSPI speed), the maximum reachable QUADSPI operating speed depends mainly on the PCB design quality so the user should optimize the PCB design.

For instance, if the user hardware allows the QUADSPI to operate at 60 MHz in DDR Quad I/O mode, then for a sequential access case, an image can be read at 60000000/1024/1024= 57.22 Mbyte/s. In this case for a 4 Kbytes image, the total transfer time is of 4107 cycles: eight cycles for command + three cycles for the address (24 bits in DDR mode) + 4096 cycles for the 4 Kbytes image.

Another important parameter to be considered by the user when selecting the Quad-SPI memory device, is the maximal clock frequency supported by the Quad-SPI 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 QUADSPI operates at its maximum speed. In that case the user can use the DDR mode to double reading speed but at the same operating QUADSPI clock.

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

Use Quad I/O mode

Using Quad I/O mode permits a boost in 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 permits to double 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 Quad-SPI mode two bytes are read each QSPI_CLK cycle.

Reduce command overhead

Each access to Quad-SPI 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 for Indirect mode
  Since each access to Quad-SPI memory needs to send a command and an address, it is beneficial to perform large burst transfers rather than small repetitive transfers; this action permits reduction of command overhead.
- Sequential access in Memory-mapped mode
  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 QUADSPI clock speed.
  In general it is the case where Quad-SPI memory is used for storage applications such as graphics or multimedia contents (see examples in Section 6.1: Reading data from Quad-SPI memory: graphical application).
- Consider timeout counter
  The user should consider that enabling timeout counter in Memory-mapped mode may increase command overhead. When timeout occurs, the QUADSPI rises chip-select. After that, to read from the Quad-SPI 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 3.4.3: Timeout counter).
  Timeout counter permits decreasing power consumption (see Section 7.2.1: Use timeout counter), 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 14. Executing non-sequential code from Quad-SPI, a command and an address are sent to the memory every new read sequence. In this case, the user should enable the SIOO feature in order to reduce the command overhead (see Section 3.4.1: Send instruction only-once (SIOO)).

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 Quad-SPI Flash memory, the user should follow the previously described read performance recommendations.

Executing from the Quad-SPI 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 51. 6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in Quad-SPI memory, placing both the code and the read-only data in the Quad-SPI 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 Quad-SPI memory while the code can be located in the Flash-TCM. This action permits 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 Quad-SPI 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 Quad-SPI memory.

For the STM32F7x5/F7x6, it is recommended to enable the Cortex®-M7 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 permit any AHB master to access the Quad-SPI memory without CPU intervention.

7.2 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 Quad-SPI memory also can be put in low-power mode (also known as deep power-down mode).

For most Quad-SPI 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.

7.2.1 Use timeout counter

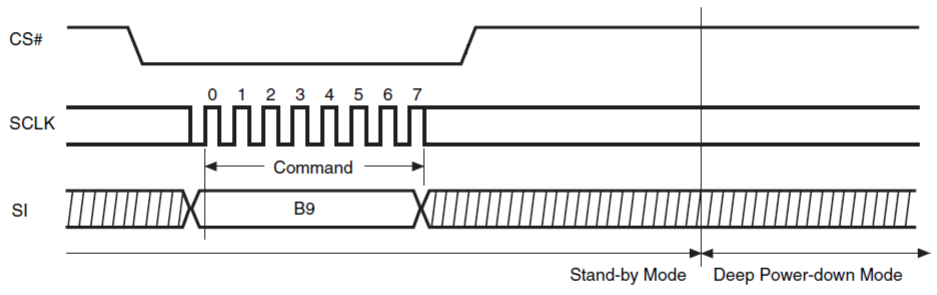
The timeout nCS feature can be used to avoid any extra power-consumption in the external Flash memory. When the clock is stopped for a long time, the timeout counter can release the nCS pin to put the external Flash memory in a lower-consumption state after a period of timeout elapsed without any access (see Section 3.4.3: Timeout counter).

7.2.2 Put the Quad-SPI 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.

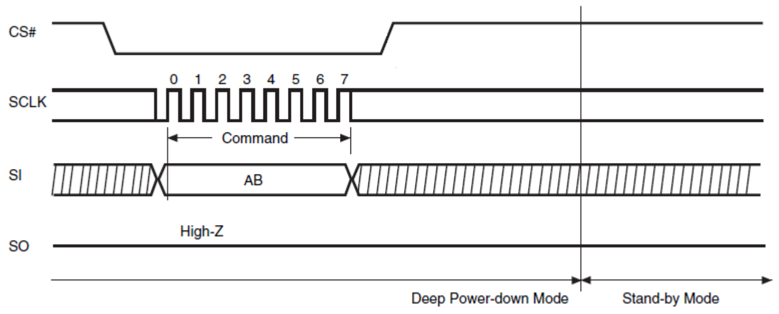
Before entering a low-power mode or when the Quad-SPI memory is not used, it can be put in DPD mode in order to reduce the overall application's power-consumption. For several memory brands the command 0xB9 should be sent to the external serial-Flash memory to enter the deep power-down mode. The following figure shows the DPD sequence:

Figure 57. Deep power-down (DPD) sequence (command B9)



To exit DPD mode, the RELEASE FROM DEEP POWERDOWN command (0xAB) should be sent. The figure below shows the RPD sequence:

Figure 58. 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 externalQuad-SPI Flash memory and to cancel its current consumption.

7.2.3 Quad-SPI Flash memories supporting DPD mode

Many Quad-SPI memory brands support the DPD mode, here below an example:

- WINBOND: W25Q64CV
- Spansion: S25FL032P
- MICRON: MT25QL512AB
- Macronix: MX25L12865F

8 Supported devices

The STM32 Quad-SPI interface has a very flexible frame format that permits the following:

- 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 bytes
- Send one, two, three or four alternate-byte
- Send up to 31 dummy clock cycles.

In addition, STM32 Quad-SPI interface permits sending any command, so the user can program the desired command in the QUADSPI_CCR register in the INSTRUCTION[7:0] field.

TheSTM32 Quad-SPI interface is fully configurable in terms of frame format and hardware and it supports most Quad-SPI memory in the market.

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

9 Conclusion

The STM32 devices provide a very flexible and useful Quad-SPI interface, which fits memory hungry applications at a lower development cost. The QUADSPI 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 Quad-SPI interface performances and flexibility, which allows lower development costs and faster time to market.

Revision history

Table 17. **Document revision history**

Date	Version	Changes
01-Apr-2016	1	Initial release.
02-May-2019	2	Document updated to enlarge scope to other product families. All sections are impacted by the update.
28-Apr-2020	3	Updated: • Document title: *Quad-SPI interface on STM32 microcontrollers and microprocessorsTable 1:Applicable products* • *Section 1: General information* • *Section2.1: QUADSPI availability and features across STM32 families* • *Table 3:Benefits of using STM32 Quad-SPI interface* • *Figure 1: System architecture: STM32L4 Series* • *Figure 2: System architecture: STM32F4 Series* • *Figure 3: System architecture: STM32F7 Series* • *Figure 4: System architecture: STM32H7 Series* • *Section 2.3.5: System architecture: STM32WB35xx and STM32WB55xx devices* • *Figure 5: System architecture:STM32WB35xx and STM32WB55xx* • *Figure 45: DMA2D reading images from Quad-SPI to build frame buffer content* • *Figure 46: LTDC reading an image directly from Quad-SPI memory* • *Figure 50: 6_1-Quad-SPI_rwRAM-DTCM project configuration: code and data in Quad-SPI memory* • *Figure 51: 6_2-Quad-SPI_rwRAM-DTCM project configuration: onlycode in Quad-SPI memory* • *Figure 52: Indirect write mode: programming Quad-SPI memory using DMA* • *Figure 53: Indirect write mode: programming Quad-SPI memory using interrupt*
09-Dec-2021	4	• All sections of this document were updated. • Added STM32G4 Series devices information.
25-Feb-2026	5	Updated: • Document title. • Section 4.2.2: Quad-SPI flash memory parameter configuration (QUADSPI_DCR register) • Table 1. Applicable products • Table 2. QUADSPI availability and features across STM32 families • Table 3. Benefits of using STM32 Quad-SPI interface

Contents

List of tables

List of figures