AN4031
Application note

Using the STM32F2, STM32F4 and STM32F7 Series DMA controller

Introduction

This application note describes how to use direct memory access (DMA) controller available in STM32F2, STM32F4 and STM32F7 Series. The DMA controller features, the system architecture, the multi-layer bus matrix and the memory system contribute to provide a high data bandwidth and to develop very low latency response-time software.

This application note also describes some tips and tricks to allow developers to take full advantage of these features and ensure correct response times for different peripherals and subsystems.

STM32F2, STM32F4 and STM32F7 are referred to as “STM32F2/F4/F7 devices” and the DMA controller as “DMA” throughout the document.

In this document STM32F4 Series is selected as illustrative example. DMA behavior is the same over STM32F2, STM32F4 and STM32F7 Series unless otherwise specified.

Reference documents

This application note should be read in conjunction with the STM32F2/F4/F7 reference manuals:

- STM32F205/215 and STM32F207/217 reference manual (RM0033)
- STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 reference manual (RM0090)
- STM32F401xB/C and STM32F401xD/E reference manual (RM0368)
- STM32F410 reference manual (RM0401)
- STM32F411xC/E reference manual (RM0383)
- STM32F412 reference manual (RM0402)
- STM32F446xx reference manual (RM0390)
- STM32F469xx and STM32F479xx reference manual (RM0386)
- STM32F75xxx and STM32F74xxx reference manual (RM0385)
- STM32F76xxx and STM32F77xxx reference manual (RM0410)

Contents

List of tables

List of figures

1 DMA controller description

The DMA is an AMBA advanced high-performance bus (AHB) module that features three AHB ports: a slave port for DMA programming and two master ports (peripheral and memory ports) that allow the DMA to initiate data transfers between different slave modules.

The DMA allows data transfers to take place in the background, without the intervention of the Cortex-Mx processor. During this operation, the main processor can execute other tasks and it is only interrupted when a whole data block is available for processing.

Large amounts of data can be transferred with no major impact on the system performance. The DMA is mainly used to implement central data buffer storage (usually in the system SRAM) for different peripheral modules. This solution is less expensive in terms of silicon and power consumption compared to a distributed solution where each peripheral needs to implement it own local data storage.

The STM32F2/F4/F7 DMA controller takes full advantage of the multi-layer bus system in order to ensure very low latency both for DMA transfers and for CPU execution/interrupt event detection/service.

1.1 DMA transfer properties

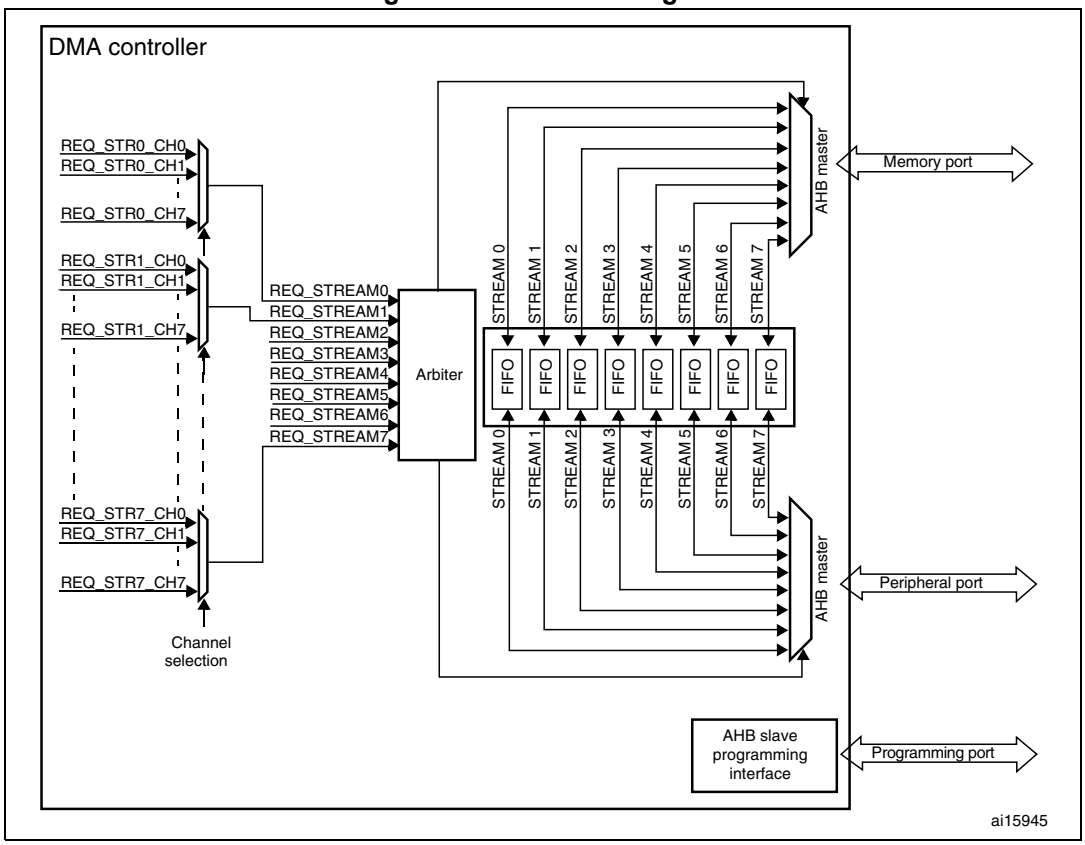
A DMA transfer is characterized by the following properties:

- DMA stream/channel
- Stream priority
- Source and destination addresses
- Transfer mode
- Transfer size (only when DMA is the flow controller)
- Source/destination address incrementing or non-incrementing
- Source and destination data width
- Transfer type
- FIFO mode
- Source/destination burst size
- Double-buffer mode
- Flow control

STM32F2/F4/F7 devices embed two DMA controllers, and each DMA has two port, one peripheral port and one memory port, which can work simultaneously.

Figure 1 shows the DMA block diagram.

Figure 1. DMA block diagram



The following subsections provide a detailed description of each DMA transfer property.

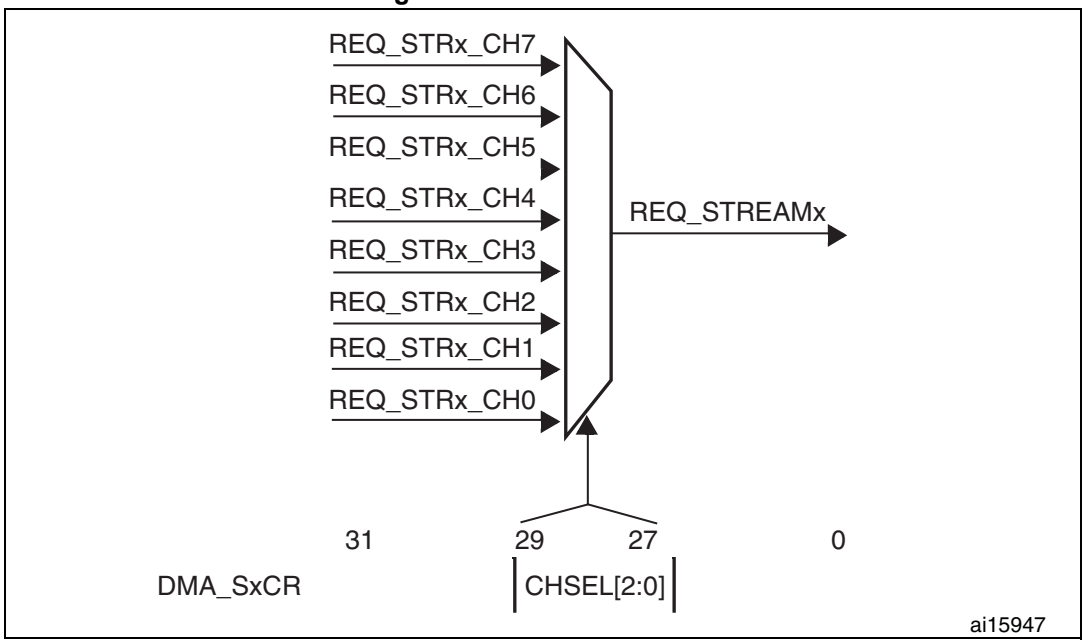
1.1.1 DMA streams/channels

STM32F2/F4/F7 devices embed two DMA controllers, offering up to 16 streams in total (eight per controller), each dedicated to managing memory access requests from one or more peripherals.

Each stream has up to eight selectable channels (requests) in total. This selection is software-configurable and allows several peripherals to initiate DMA requests.

Figure 2 describes the channel selection for a dedicated stream.

Figure 2. Channel selection



Note: Only one channel/request can be active at the same time in a stream.

More than one enabled DMA stream must not serve the same peripheral request.

Table 1 and *Table 2* show STM32F427/437 and STM32F429/439 DMA1/DMA2 requests mapping. The tables give the available configuration of DMA streams/channels versus peripheral requests.

Table 1. STM32F427/437 and STM32F429/439 DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX
Channel 1	I2C1_RX	-	TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX	UART7_TX	TIM3_CH4 TIM3_UP	UART7_RX	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 2. STM32F427/437 and STM32F429/439 DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	SPI6_TX	SPI6_RX	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX	SPI5_TX	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX	SPI5_TX	TIM8_CH4 TIM8_TRIG TIM8_COM

STM32F2/F4/F7 DMA request mapping is designed in such a way that the software application has more flexibility to map each DMA request for the associated peripheral request, and that most of the use case applications are covered by multiplexing the corresponding DMA streams and channels. Refer to DMA1/DMA2 request mapping tables in the reference manual corresponding to the microcontroller you are using (see *Section : Reference documents*).

1.1.2 Stream priority

Each DMA port has an arbiter for handling the priority between other DMA streams. Stream priority is software-configurable (there are four software levels). If two or more DMA streams have the same software priority level, the hardware priority is used (stream 0 has priority over stream 1, etc.).

1.1.3 Source and destination addresses

A DMA transfer is defined by a source address and a destination address. Both the source and destination should be in the AHB or APB memory ranges and should be aligned to transfer size.

1.1.4 Transfer mode

DMA is capable of performing three different transfer modes:

- Peripheral to memory,
- Memory to peripheral,
- Memory to memory (only DMA2 is able to do such transfer, in this mode, the circular and direct modes are not allowed.)

1.1.5 Transfer size

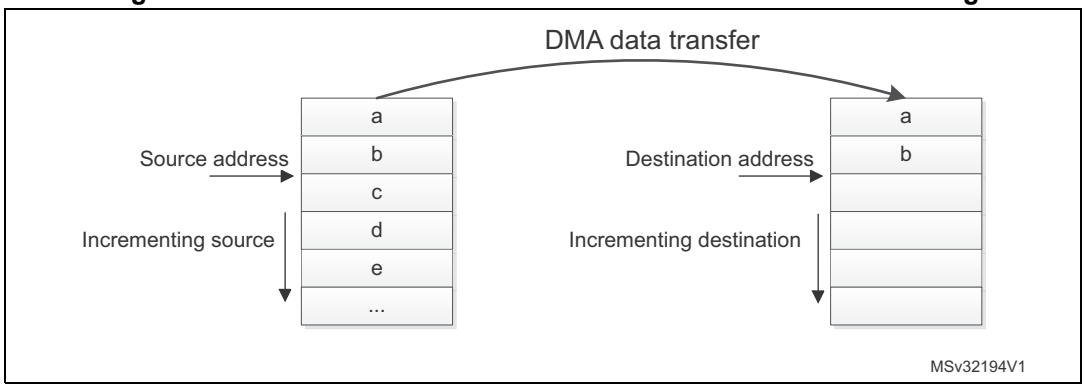
The transfer size value has to be defined only when the DMA is the flow controller. In fact, this value defines the volume of data to be transferred from source to destination.

The transfer size is defined by the DMA_SxNDTR register value and by the peripheral side data width. Depending on the received request (burst or single), the transfer size value is decreased by the amount of the transferred data.

1.1.6 Incrementing source/destination address

It is possible to configure the DMA to automatically increment the source and/or destination address after each data transfer.

Figure 3. DMA source address and destination address incrementing



1.1.7 Source and destination data width

Data width for source and destination can be defined as:

- Byte (8 bits)
- Half-word (16 bits)
- Word (32 bits)

1.1.8 Transfer types

- Circular mode: the Circular mode is available to handle circular buffers and continuous data flows (the DMA_SxNDTR register is then reloaded automatically with the previously programmed value).
- Normal mode: once the DMA_SxNDTR register reaches zero, the stream is disabled (the EN bit in the DMA_SxCR register is then equal to 0).

1.1.9 DMA FIFO mode

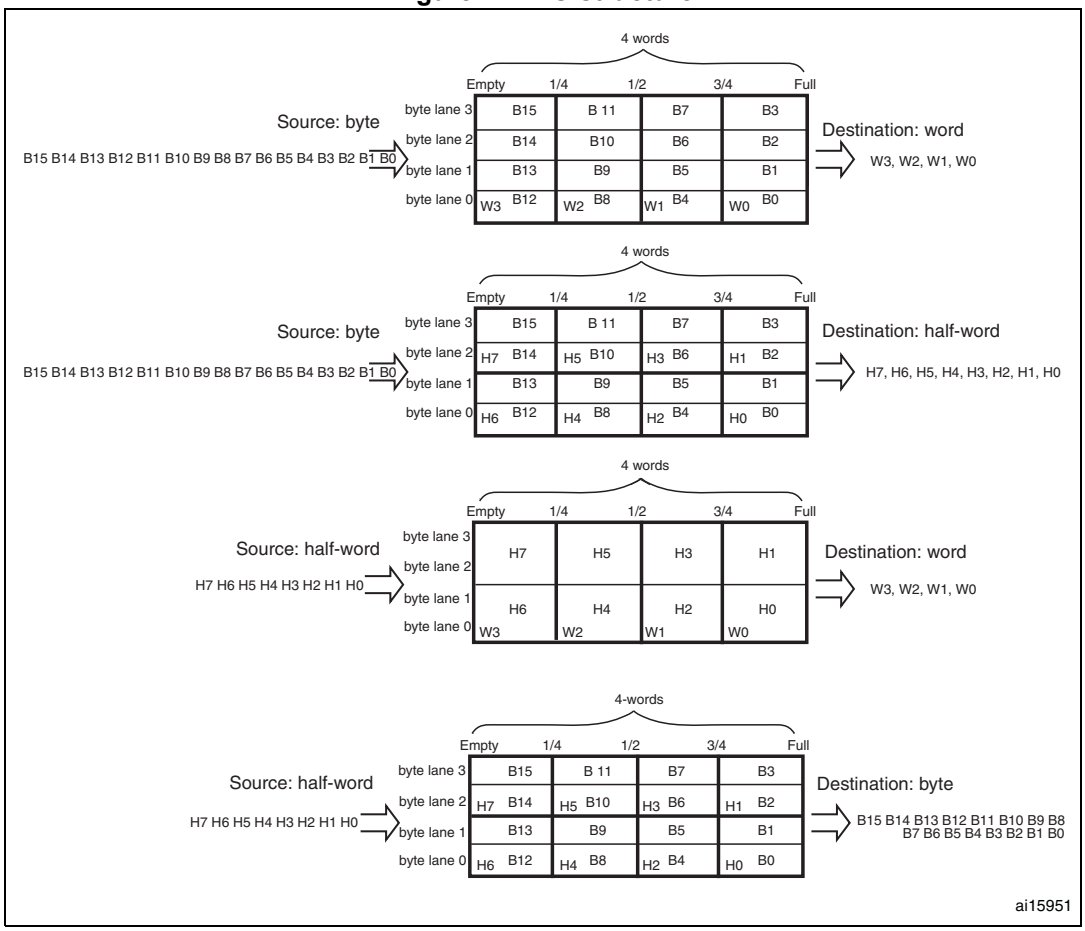
Each stream has an independent 4-word (4 * 32 bits) FIFO and the threshold level is software-configurable between 1/4, 1/2, 3/4 or full. The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

DMA FIFO can be enabled or disabled by software; when disabled, the Direct mode is used. If DMA FIFO is enabled, data packing/unpacking and/or Burst mode can be used. The configured DMA FIFO threshold defines the DMA memory port request time.

The DMA FIFOs implemented on STM32F2/F4/F7 devices help to:

- reduce SRAM access and so give more time for the other masters to access the bus matrix without additional concurrency,
- allow software to do burst transactions which optimize the transfer bandwidth,
- allow packing/unpacking data to adapt source and destination data width with no extra DMA access.

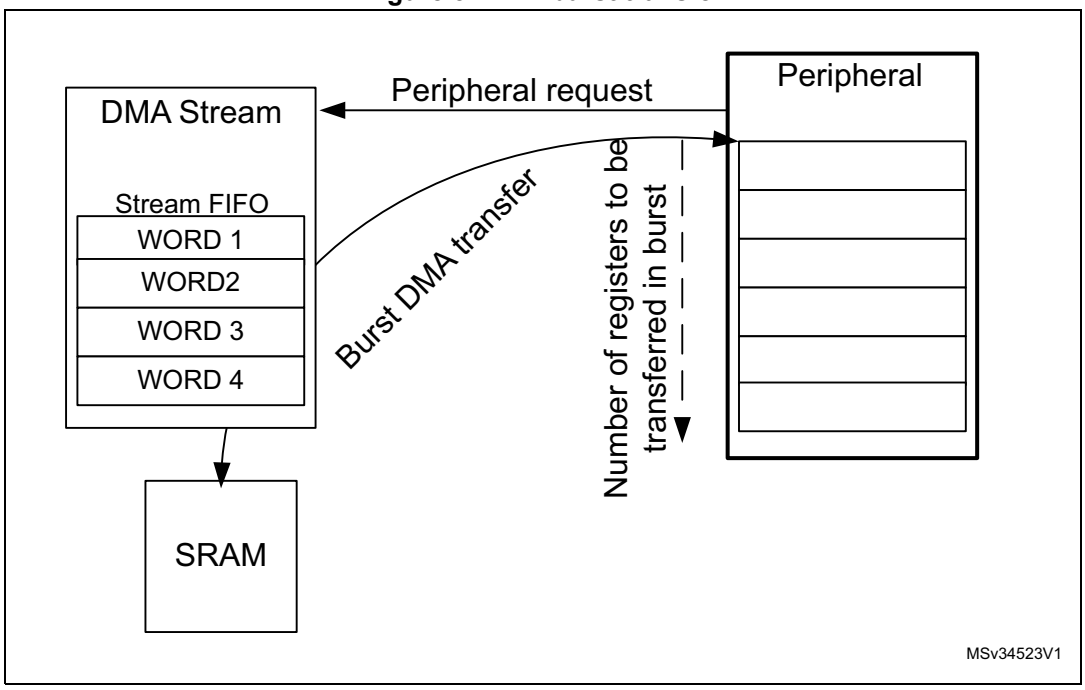
Figure 4. FIFO structure



1.1.10 Source and destination burst size

Burst transfers are guaranteed by the implemented DMA FIFOs.

Figure 5. DMA burst transfer



In response to a burst request from peripheral DMA reads/writes the number of data units (data unit can be a word, a half-word, or a byte) programmed by the burst size (4x, 8x or 16x data unit). The burst size on the DMA peripheral port must be set according to the peripheral needs/capabilities.

The DMA burst size on the memory port and the FIFO threshold configuration must match. This allows the DMA stream to have enough data in the FIFO when burst transfer on the memory port is started. *Table 3* shows the possible combinations of memory burst size, FIFO threshold configuration and data size.

To ensure data coherence, each group of transfers that form a burst is indivisible: AHB transfers are locked and the arbiter of the AHB bus matrix does not remove the DMA master's access rights during the burst transfer sequence.

Table 3. Possible burst configurations

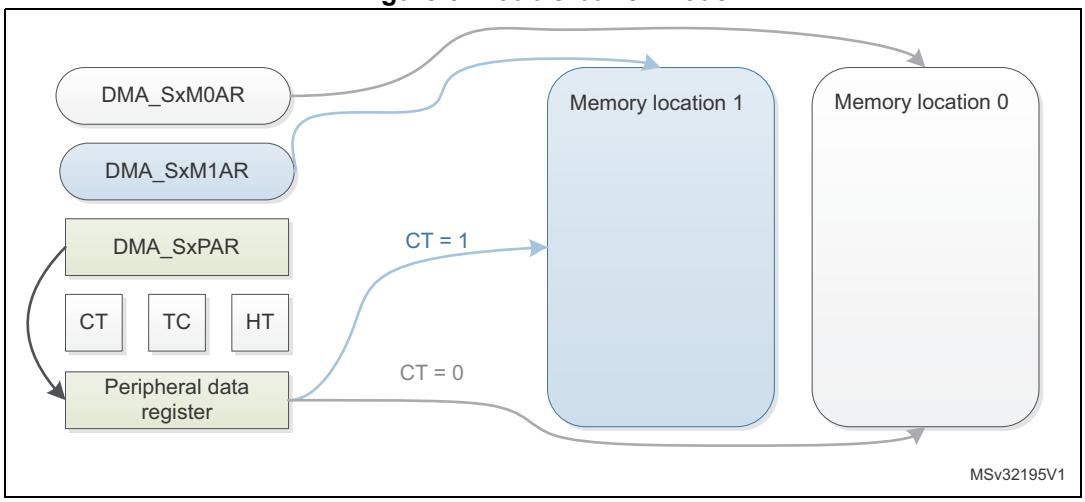
MSIZE	FIFO level	MBURST = INCR4	MBURST = INCR8	MBURST = INCR16
Byte	1/4	1 burst of 4 bytes	forbidden	forbidden
	1/2	2 bursts of 4 bytes	1 burst of 8 bytes	
	3/4	3 bursts of 4 bytes	forbidden	
	Full	4 bursts of 4 bytes	2 bursts of 8 bytes	1 burst of 16 bytes
Half-word	1/4	forbidden	forbidden	forbidden
	1/2	1 burst of 4 half-words		
	3/4	forbidden		
	Full	2 bursts of 4 half-words	1 burst of 8 Half-word	
Word	1/4	forbidden	forbidden	
	1/2			
	3/4			
	Full	1 burst of 4 words		

1.1.11 Double-buffer mode

A double-buffer stream works as a regular (single-buffer) stream, with the difference that it has two memory pointers. When the Double-buffer mode is enabled, the Circular mode is automatically enabled and at each end of transaction (DMA_SxNDTR register reach 0), the memory pointers are swapped.

This allows the software to process one memory area while the second memory area is being filled/used by the DMA transfer.

Figure 6. Double-buffer mode



In Double-buffer mode, it is possible to update the base address for the AHB memory port on-the-fly (DMA_SxM0AR or DMA_SxM1AR) when the stream is enabled:

- When the CT (Current Target) bit in the DMA_SxCR register is equal to 0, the current DMA memory target is memory location 0 and so the base address memory location 1 (DMA_SxM1AR) can be updated.
- When the CT bit in the DMA_SxCR register is equal to 1, the current DMA memory target is memory location 1 and so the base address memory location 0 (DMA_SxM0AR) can be updated.

1.1.12 Flow control

The flow controller is the unit that controls the data transfer length and which is responsible for stopping the DMA transfer.

The flow controller can be either the DMA or the peripheral.

- With DMA as flow controller:

In this case, it is necessary to define the transfer size value in the DMA_SxNDTR register before enabling the associated DMA stream. When a DMA request is served, the transfer size value decreases by the amount of transferred data (depending of the type of request: burst or single).

When the transfer size value reaches 0, the DMA transfer is finished and the DMA stream is disabled.

- With the peripheral as flow controller:

This is the case when the number of data items to be transferred is unknown. The peripheral indicates by hardware to the DMA controller when the last data are being transferred. Only the SD/MMC and JPEG peripherals support this mode.

1.2 Setting up a DMA transfer

To configure DMA stream x (where x is the stream number), the following procedure should be applied:

1. If the stream is enabled, disable it by resetting the EN bit in the DMA_SxCR register, then read this bit in order to confirm that there is no ongoing stream operation. Writing this bit to 0 is not immediately effective since it is actually written to 0 once all the current transfers have finished. When the EN bit is read as 0, this means that the stream is ready to be configured. It is therefore necessary to wait for the EN bit to be cleared before starting any stream configuration. All the stream-dedicated bits set in

the status register (DMA_LISR and DMA_HISR) from the previous data block DMA transfer should be cleared before the stream can be re-enabled.

2. Set the peripheral port register address in the DMA_SxPAR register. The data will be moved from/to this address to/from the peripheral port after the peripheral event.
3. Set the memory address in the DMA_SxMA0R register (and in the DMA_SxMA1R register in the case of a Double-buffer mode). The data will be written to or read from this memory after the peripheral event.
4. Configure the total number of data items to be transferred in the DMA_SxNDTR register. After each peripheral event or each beat of the burst, this value is decremented.
5. Select the DMA channel (request) using CHSEL[2:0] in the DMA_SxCR register.
6. If the peripheral is intended to be the flow controller and if it supports this feature, set the PFCTRL bit in the DMA_SxCR register.
7. Configure the stream priority using the PL[1:0] bits in the DMA_SxCR register.
8. Configure the FIFO usage (enable or disable, threshold in transmission and reception).
9. Configure the data transfer direction, peripheral and memory incremented/fixed mode, single or burst transactions, peripheral and memory data widths, Circular mode, Double-buffer mode and interrupts after half and/or full transfer, and/or errors in the DMA_SxCR register.
10. Activate the stream by setting the EN bit in the DMA_SxCR register.

As soon as the stream is enabled, it can serve any DMA request from the peripheral connected to the stream.

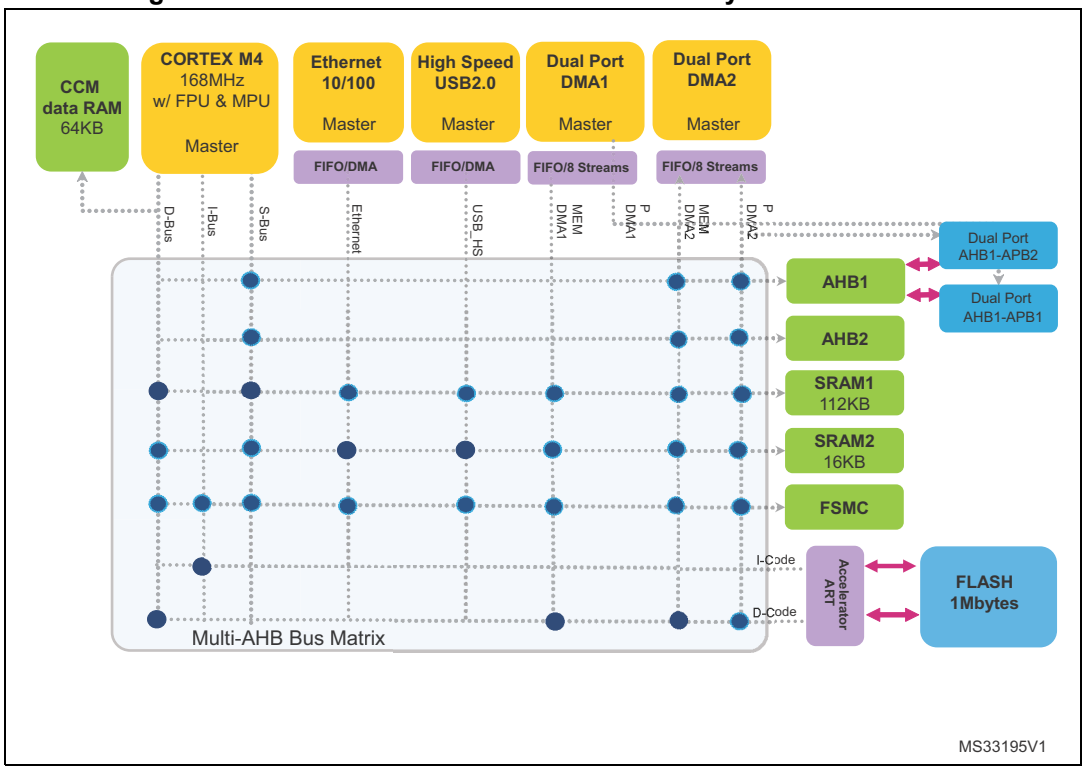
2 System performance considerations

STM32F2/F4/F7 devices embed a multi-masters/multi-slaves architecture:

- Multiple masters:
  - Cortex®-Mx core AHB buses
  - DMA1 memory bus
  - DMA2 memory bus
  - DMA2 peripheral bus
  - Ethernet DMA bus
  - USB high-speed DMA bus
  - Chrom-ART Accelerator bus
  - LCD-TFT bus
- Multiple slaves:
  - Internal Flash interfaces connected to the multi-layer bus matrix
  - Main internal SRAM1 and Auxiliary internal SRAMs (SRAM2, SRAM3 when available on device)
  - AHB1 peripherals including AHB-to-APB bridges and APB peripherals
  - AHB2 peripherals
  - AHB3 peripheral (such as FMC, Quad-SPI peripherals when available on product line)

Masters and slaves are connected via a multi-layer bus matrix ensuring concurrent access and efficient operation, even when several high-speed peripherals work simultaneously. This architecture is shown in the next figure for the case of STM32F405/415 and STM32F407/417 lines.

Figure 7. STM32F405/415 and STM32F407/417 system architecture



2.1 Multi-layer bus matrix

The multi-layer bus matrix allows masters to perform data transfers concurrently as long as they are addressing different slave modules. On top of the Cortex-Mx architecture and dual AHB port DMAs, this structure enhances data transfer parallelism, thus contributing to reduce the execution time, and optimizing the DMA efficiency and power consumption.

2.1.1 Definitions

- AHB master: a bus master is able to initiate read and write operations. Only one master can win bus ownership at a defined time period.
- AHB slave: a bus slave response to master read or write operations. The bus slave signals back to master success, failure or waiting states.
- AHB arbiter: a bus arbiter insures that only one master can initiate a read or write operation at one time.
- AHB bus matrix: a multi-layer AHB bus matrix that interconnects AHB masters to AHB slaves with dedicated AHB arbiter for each layer. The arbitration uses a round-robin algorithm.

2.1.2 Round-robin priority scheme

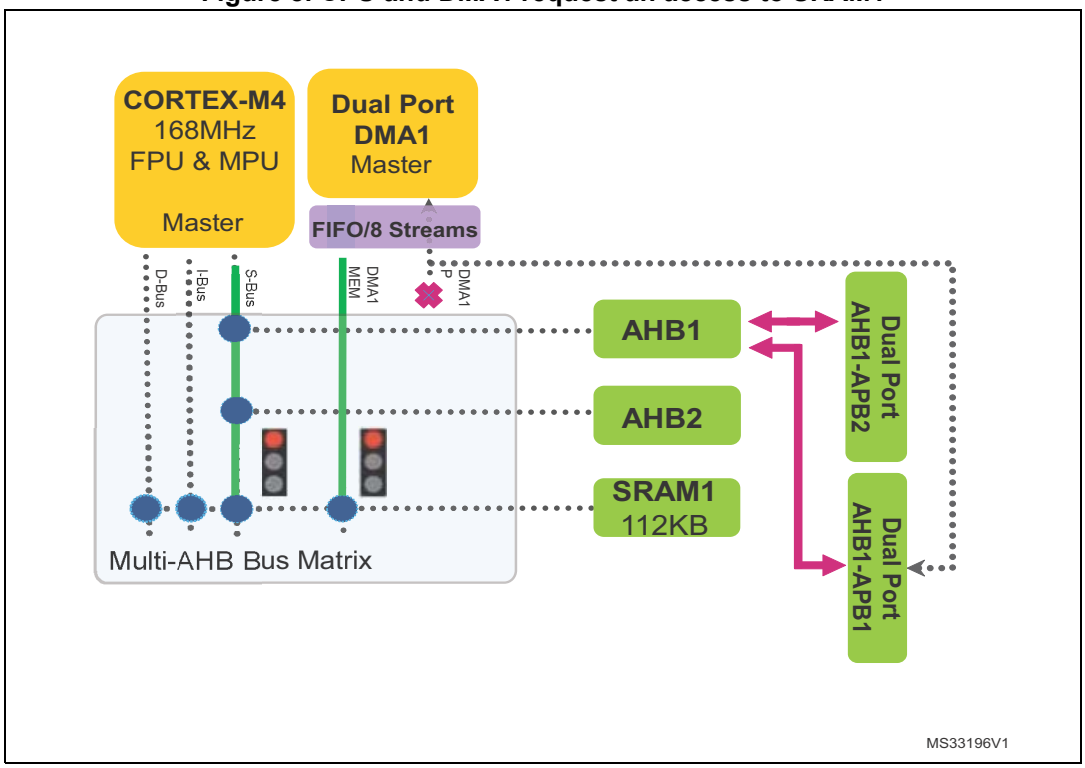
A round-robin priority scheme is implemented at bus matrix level in order to ensure that each master can access any slave with very low latency:

- Round-robin arbitration policy allows a fair distribution of bus bandwidth.
- Maximum latency is bounded.
- Round-robin quantum is 1x transfer.

Bus matrix arbiters intervene to solve access conflicts when several AHB masters try to access the same AHB slave simultaneously.

In the following example (*Figure 8*), both the CPU and DMA1 try to access SRAM1 to read data.

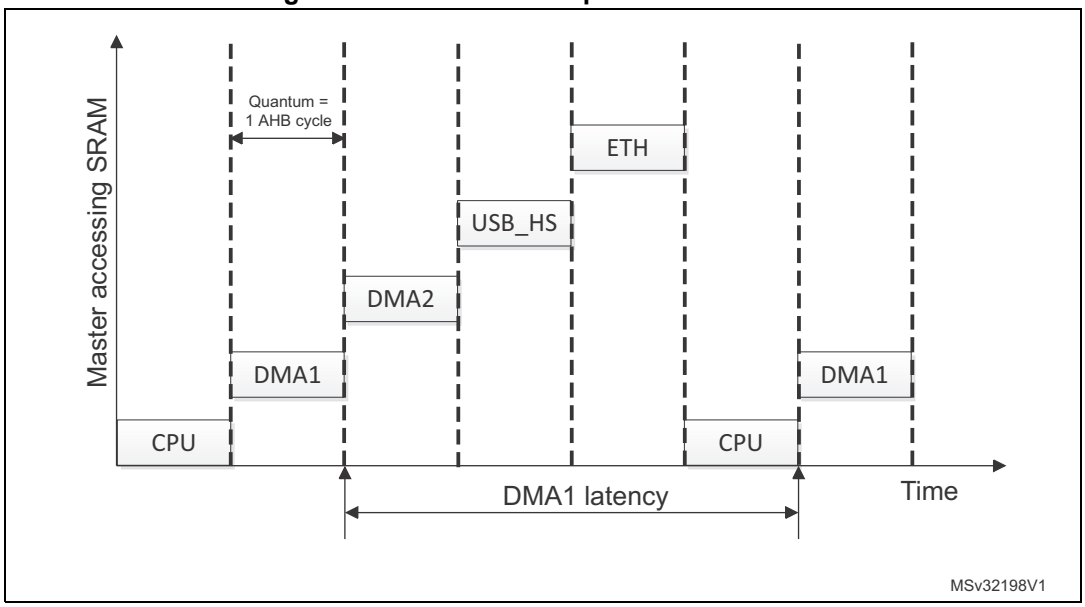
Figure 8. CPU and DMA1 request an access to SRAM1



In case of bus access concurrency as in the above example, a bus matrix arbitration is required. The round-robin policy is then applied in order to solve the issue: if the last master which wan the bus was the CPU, during the next access DMA1 wins the bus and accesses SRAM1 first. The CPU has then the rights to access SRAM1.

This proves that the transfer latency associated to one master depends on the number of other pending master requests to access the same AHB slave. In the following example (*Figure 9*), five masters try to access simultaneously SRAM1.

Figure 9. Five masters request SRAM access



The latency associated to DMA1 to win the bus matrix again and access SRAM1 (for example) is equal to the execution time of all pending requests coming from the other masters.

2.1.3 BusMatrix arbitration and DMA transfer delays worst case

The latency seen by the DMA master port on one transaction depends on the other masters' transfer types and lengths.

For instance, if we consider previous DMA1 & CPU example (*Figure 8*) with concurrency to access SRAM, latency on the DMA transfer varies depending on the CPU transaction length.

If bus access is first granted to the CPU and the CPU is not performing a single data load/store, the DMA wait time to gain access to SRAM can expand from one AHB cycle for a single data load/store to N AHB cycles, where N is the number of data words in the CPU transaction.

The CPU locks the AHB bus to keep ownership and reduces latency during multiple load/store operations and interrupts entry. This enhances firmware responsiveness but it can result in delays on the DMA transaction.

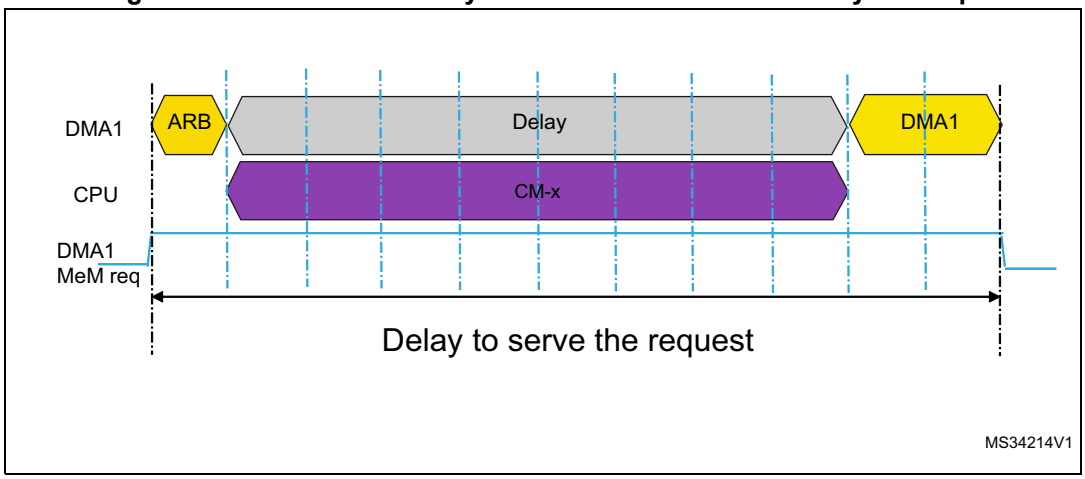
Delay on DMA1 SRAM access when in concurrency with CPU depends on the type of transfer:

- CPU transfer issued by interrupt (context save): 8 AHB cycles
- CPU transfer issued by cache controller (256-bit cache line fill/eviction): 8 AHB cycles[a]
- CPU transfer issued by LDM/STM instructions: 14 AHB cycles[b]
  - Transfers of up to 14 registers from/to memory

a. Only for STM32F7xx devices

b. Latency due to transfer issued by LDM/STM instructions can be reduced by configuring compiler to split load/store multiple instructions into single load/store instructions.

Figure 10. DMA transfer delay due to CPU transfer issued by interrupt



The above figure details the case of a DMA transfer delayed by a CPU multi-cycle transfer due to an interrupt entry. DMA memory port is triggered to perform a memory access. After arbitration, AHB bus is not granted to DMA1 memory port but to CPU. An additional delay is observed to serve the DMA request. It is 8 AHB cycles for a CPU transfer issued by interrupt.

The same behavior can be observed with other masters (like DMA2, USB_HS, Ethernet…) when addressing simultaneously the same slave with a transaction length different from one data unit.

In order to improve DMA access performance over BusMatrix, it is recommended to avoid bus contention.

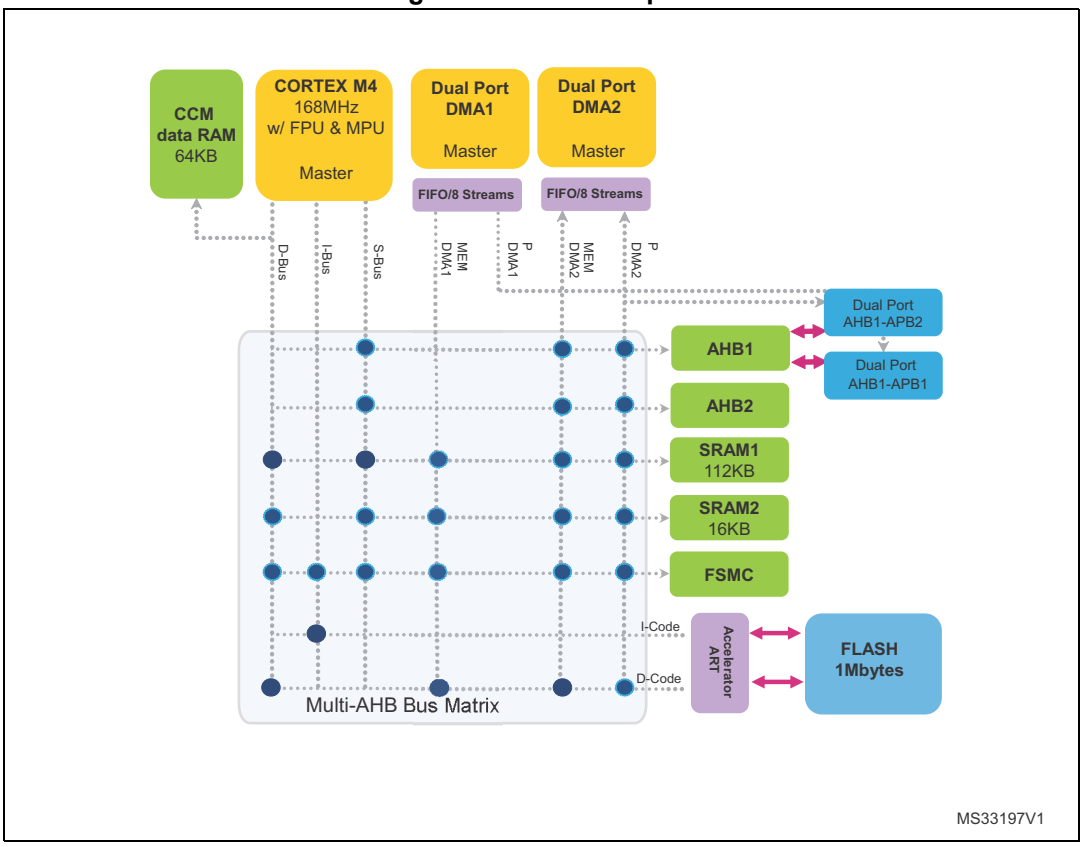
2.2 DMA transfer paths

2.2.1 Dual DMA port

STM32F2/F4/F7 devices embed two DMAs. Each DMA has two ports, a memory port and a peripheral port, which can operate simultaneously not only at DMA level but also with other system masters, using the external bus matrix and dedicated DMA paths.

The simultaneous operation allows to optimize DMA efficiency and to reduce response time (wait time between request and data transfer).

Figure 11. DMA dual port



For DMA2:

- The MEM (memory port) can access AHB1, AHB2, AHB3 (External memory controller, FSMC), SRAMs, and Flash memory through the bus matrix.
- The Periph (peripheral port) can access:
  - AHB1, AHB2, AHB3 (External memory controller, FSMC), SRAMs, and Flash memory through the bus matrix,
  - the AHB-to-APB2 bridge through a direct path (not crossing the bus matrix).

For DMA1:

- The MEM (memory port) can access AHB3 (External memory controller, FSMC), SRAMs, and Flash memory through the bus matrix.
- The Periph (peripheral port) can only access the AHB-to-APB1 bridge through a direct path (not crossing the bus matrix).

2.2.2 DMA transfer states

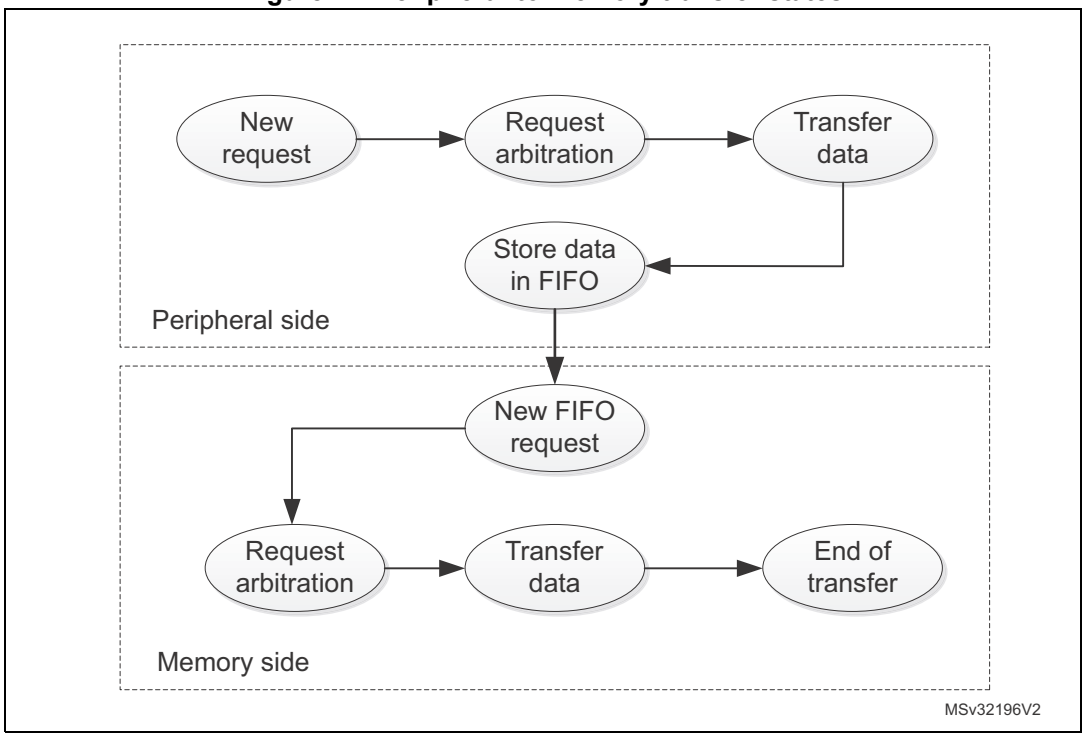
This section explains the DMA transfer steps at the peripheral port level and also at the memory port level:

- For a peripheral-to-memory transfer:

In this transfer mode, DMA requires two bus accesses to perform the transfer:

- One access over the peripheral port triggered by the peripheral's request,
- One access over the memory port which can be triggered either by the FIFO threshold (when FIFO mode is used) or immediately after peripheral read (when Direct mode is used).

Figure 12. Peripheral-to-memory transfer states

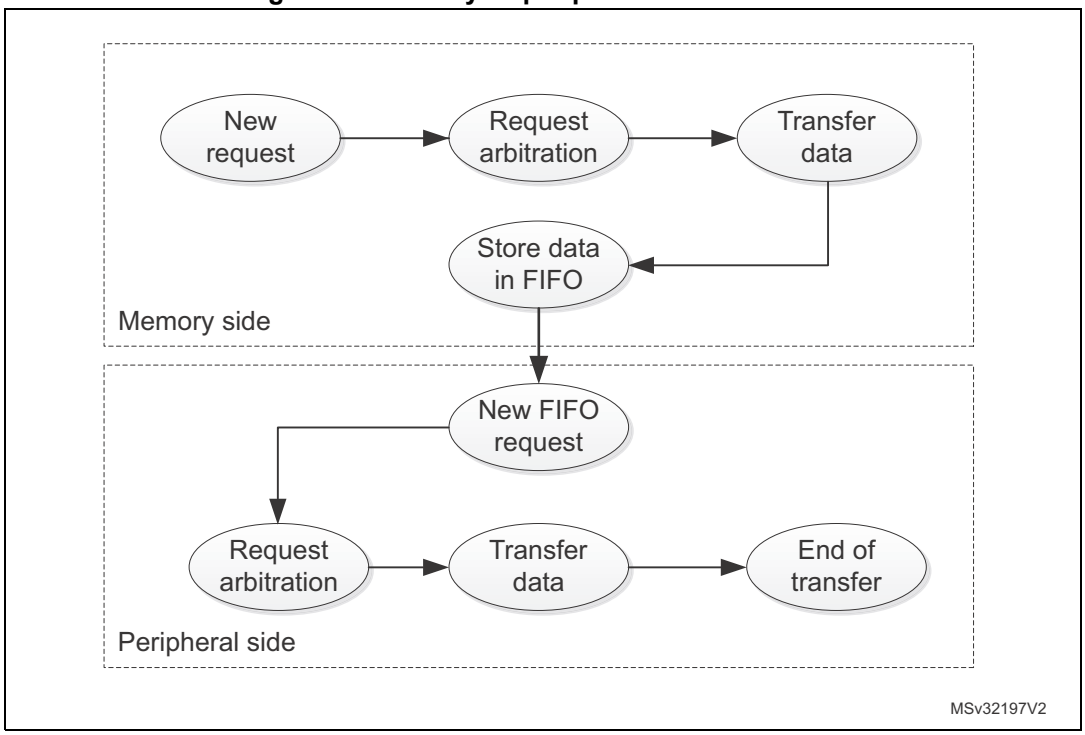


- For a memory-to-peripheral transfer:

In this transfer mode, DMA requires two bus accesses to perform the transfer:

- DMA anticipates the peripheral's access and reads data from the memory and stores it in FIFO to ensure an immediate data transfer as soon as a DMA peripheral request is triggered.
- When a peripheral request is triggered, a transfer is generated on the DMA peripheral port.

Figure 13. Memory-to-peripheral transfer states



2.2.3 DMA request arbitration

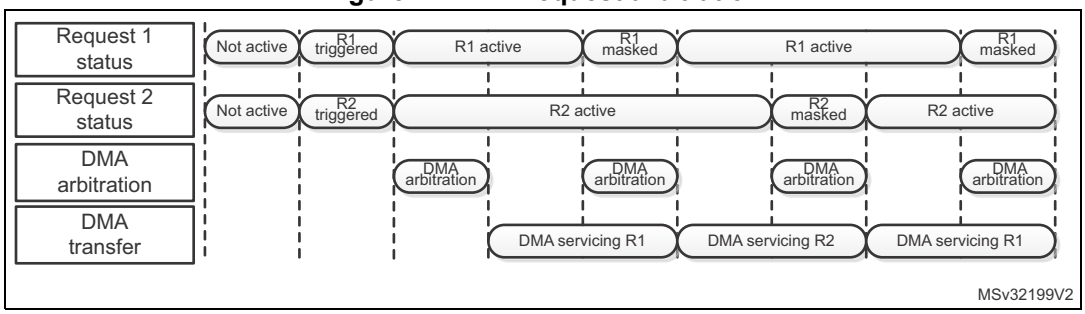
As described in *Section 1.1.2: Stream priority*, the STM32F2/F4/F7 DMA embeds an arbiter that manages the eight DMA stream requests based on their priorities for each of the two AHB master ports (memory and peripheral ports) and launches the peripheral/memory access sequences.

When more than one DMA request is active, DMA needs to arbiter internally between the active requests and decide which request is to be served first.

The following figure shows two circular DMA requests triggered at the same time by DMA stream “request 1” and by DMA stream “request 2” (requests 1 and 2 could be any DMA peripheral request). At the next AHB clock cycle, the DMA arbiter checks on the active pending requests and grants access to the “request 1” stream which has the highest priority.

The next arbitration cycle occurs during the last data cycle of the “request 1” stream. At that time, “request 1” is masked and the arbiter sees only “request 2” as active, so access is reserved to “request 2” this time, and so on.

Figure 14. DMA request arbitration



General recommendations:

- The high-speed/high-bandwidth peripherals must have the highest DMA priorities. This ensures that the maximum data latency is respected for these peripherals and over-/under-run conditions are avoided.
- In case of equal bandwidth requirements, it is recommended to assign a higher priority to the peripherals working in Slave mode (which have no control on the data transfer speed) compared with the ones working in Master mode (which may control the data flow).
- As the two DMAs can work in parallel based on the bus matrix multi-layer structure, high-speed peripherals' requests can be balanced between the two DMAs when possible.

2.3 AHB-to-APB bridge

STM32F2/F4/F7 devices embed two AHB-to-APB bridges, APB1 and APB2, to which the peripherals are connected.

2.3.1 Dual AHB-to-APB port

The AHB-to-APB bridge is a dual-port architecture that allows access through two different paths:

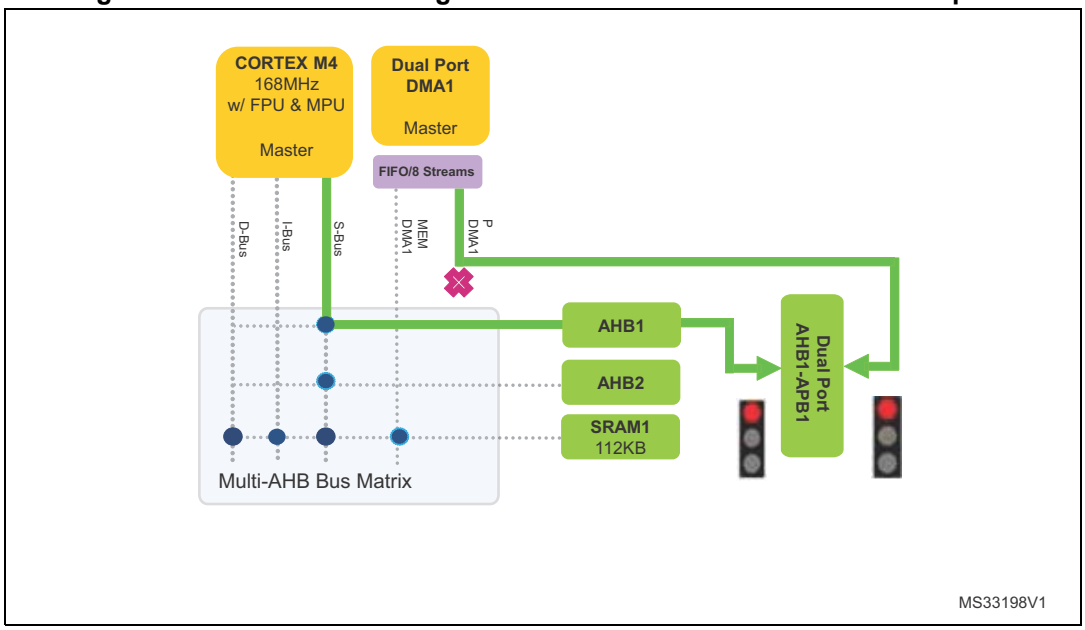
- A direct path (not crossing the bus matrix) that can be generated from DMA1 to APB1 or from DMA2 to APB2; in this case, access is not penalized by the bus matrix arbiter.
- A common path (through the bus matrix) that can be generated either from the CPU or from DMA2, which needs the bus matrix arbitration to win the bus.

2.3.2 AHB-to-APB bridge arbitration

Due to DMA's direct paths implementation on these products, an arbiter is implemented at the AHB-to-APB bridge level to solve concurrent access requests.

The following figure illustrates a concurrent access request at an AHB-APB1 bridge generated by the CPU (accessed through the bus matrix) and DMA1 (accessed through direct path).

Figure 15. AHB-to-APB1 bridge concurrent CPU and DMA1 access request



To grant bus access, the AHB-APB bridge applies the round-robin policy:

- Round-robin quantum is 1x APB transfer.
- Max latency on DMA peripheral port is bounded (1 APB transfer).

Only the CPU and DMAs can generate a concurrent access to the APB1 and APB2 buses:

- For APB1, a concurrent access can be generated if the CPU, DMA1 and/or DMA2 request simultaneous access.
- For APB2, a concurrent access can be generated if the CPU and DMA2 request simultaneous access.

3 How to predict DMA latencies

When designing a firmware application based on a microcontroller, the user must ensure that no underrun/overrun can occur, and that's why knowing the exact DMA latency for each transfer is mandatory to check if the internal system can sustain the total data bandwidth required for the application.

3.1 DMA transfer time

3.1.1 Default DMA transfer timing

As described in *Section 2.2.2*, to perform a DMA transfer from peripheral to memory, two bus accesses are required:

- One access over peripheral port triggered by peripheral request, which needs:
  - DMA peripheral port request arbitration
  - Peripheral address computation
  - Reading data from the peripheral to DMA FIFO (DMA source)
- One access over memory port which can be triggered by the FIFO threshold (when FIFO mode is used) or immediately after peripheral read (when Direct mode is used), which needs:
  - DMA memory port request arbitration
  - Memory address computation
  - Writing loaded data in SRAM (DMA destination)

When transferring data from memory to peripheral, two accesses are also required as described in *Section 2.2.2*:

- First access: DMA anticipates peripheral access and reads data from memory and stores it in FIFO to ensure an immediate data transfer as soon as DMA peripheral request is triggered. This operation needs:
  - DMA memory port request arbitration
  - Memory address computation
  - Reading data from memory to DMA FIFO (DMA source)
- Second access: when peripheral request is triggered, a transfer is generated on DMA peripheral port. This operation needs:
  - DMA peripheral port request arbitration
  - Peripheral address computation
  - Writing loaded data at peripheral address (DMA destination)

As a general rule, the total transfer time by DMA stream T_S is equal to:

$T_S = T_{SP}$ (peripheral access/transfer time) + T_{SM} (memory access/transfer time)

With:

T_{SP} is the total timing for DMA peripheral port access and transfer which is equal to:
$T_{SP} = t_{PA} + t_{PAC} + t_{BMA} + t_{EDT} + t_{BS}$

Where:

Table 4. Peripheral port access/transfer time versus DMA path used

Description	Through bus matrix		DMA's direct paths
	To AHB peripherals	To APB peripherals	
t_{PA}: DMA peripheral port arbitration	1 AHB cycle	1 AHB cycle	1 AHB cycle
t_{PAC}: peripheral address computation	1 AHB cycle	1 AHB cycle	1 AHB cycle
t_{BMA}: bus matrix arbitration (when no concurrency)(1)	1 AHB cycle	1 AHB cycle	N/A
t_{EDT}: effective data transfer	1 AHB cycle(2) (3)	2 APB cycles	2 APB cycle
t_{BS}: bus synchronization	N/A	1 AHB cycle	1 AHB cycle

1. In the case of STM32F401/STM32F410/STM32F411/STM32F412 line, t_{BMA} is equal to zero.
2. For FMC, an additional cycle can be added depending on the external memory used. Additional AHB cycles are added depending on external memory timings.
3. In case of burst, the effective data transfer time depends on the burst length (INC4 t_{EDT}= 4 AHB cycles).

- T_{SM} is the total timing for DMA memory port access and transfer which is equal to:
  $T_{SM} = t_{MA} + t_{MAC} + t_{BMA} + t_{SRAM}$

Where:

Table 5. Memory port access/transfer time

Description	Latency
t_{MA}: DMA memory port arbitration	1 AHB cycle
t_{MAC}: memory address computation	1 AHB cycle
t_{BMA}: bus matrix arbitration (when no concurrency)(1)	1 AHB cycle(2)
t_{SRAM}: SRAM read or write access	1 AHB cycle

1. In the case of STM32F401/STM32F410/STM32F411/STM32F412 line, t_{BMA} is equal to zero.
2. For consecutive SRAM accesses (while no other master accesses the same SRAM in-between), t_{BMA} = 0 cycle.

3.1.2 DMA transfer time versus concurrent access

Additional latency can be added to the DMA service timing described in *Section 3.1.1* when several masters try to access simultaneously to the same slave.

For peripheral and memory worst-case access/transfer time, the following factors impact the total delay time for DMA stream service:

- When several masters are accessing the same AHB destination simultaneously, the DMA latency is impacted; the DMA transfer cannot start until the bus matrix arbiter grants access to the DMA as described in *Section 2.1.2*.
- When several masters (DMA and CPU) are accessing the same AHB-to-APB bridge, the DMA transfer time is delayed due to the AHB-to-APB bridge arbitration as described in *Section 2.3.2*.

3.2 Examples

3.2.1 ADC-to-SRAM DMA transfer

This example is applicable to products STM32F2, STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 lines.

The ADC is configured in continuous triple Interleaved mode. In this mode, it converts continuously one analog input channel at the maximum ADC speed (36 MHz). The ADC prescaler is set to 2, the sampling time is set to 1.5 cycles, and the delay between two consecutive ADC samples of the Interleaved mode is set to 5 cycles.

The DMA2 stream0 transfers the ADC converted value to an SRAM buffer. DMA2 access to ADC is done through direct path; however, DMA access to SRAM is done through the bus matrix.

Table 6. DMA peripheral (ADC) port transfer latency

AHB/APB2 frequency Transfer time	F_{AHB} = 72 MHz/ F_{APB2} = 72 MHz AHB/APB ratio = 1	FAHB = 144 MHz/ FAPB2 = 72 MHz AHB/APB ratio = 2
t_{PA}: DMA peripheral port arbitration	1 AHB cycle	1 AHB cycle
t_{PAC}: peripheral address computation	1 AHB cycle	1 AHB cycle
t_{BMA}: bus matrix arbitration	N/A(1)	N/A(1)
t_{EDT}: effective data transfer	2 AHB cycles	4 AHB cycles
t_{BS}: bus synchronization	1 AHB cycle	1 AHB cycle
T_{SP}: total DMA transfer time for peripheral port	5 AHB cycles	7 AHB cycles

1. DMA2 accesses ADC through direct path: no bus matrix arbitration.

Table 7. DMA memory (SRAM) port transfer latency

CPU/APB2 frequency Transfer time	FAHB = 72MHz/ FAPB2=72MHz AHB/APB ratio = 1	FAHB = 144MHz/ FAPB2=72MHz AHB/APB ratio = 2
t_{MA}: DMA memory port arbitration	1 AHB cycle	1 AHB cycle
t_{MAC}: memory address computation	1 AHB cycle	1 AHB cycle
t_{BMA}: bus matrix arbitration	1 AHB cycle(1)	1 AHB cycle(1)
t_{SRAM}: SRAM write access	1 AHB cycle	1 AHB cycle
T_{SM}: total DMA transfer time for memory port	4 AHB cycles	4 AHB cycles

1. In case of DMA multiple access to SRAM, the bus matrix arbitration is equal to 0 cycle if no other master accessed to the SRAM in-between.

In this example, the total DMA latency from the ADC DMA trigger (ADC EOC) to write the ADC value on SRAM is equal to 9 AHB cycles for AHB/APB prescaler equals 1 and 11 AHB cycles for AHB/APB prescaler equals 2.

Note: When using FIFO, the DMA memory port access is launched when reaching the FIFO level configured by the user.

3.2.2 SPI full duplex DMA transfer

This example is applicable to products STM32F2, STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 lines, and is based on the SPI1 peripheral.

Two DMA requests are configured:

- DMA2_Stream2 for SPI1_RX: this stream is configured to be the highest priority in order to serve in time the SPI1 received data, and transfer it from the SPI1_DR register to the SRAM buffer.
- DMA2_Stream3 for SPI1_TX: this stream transfers data from the SRAM buffer to the SPI1_DR register.

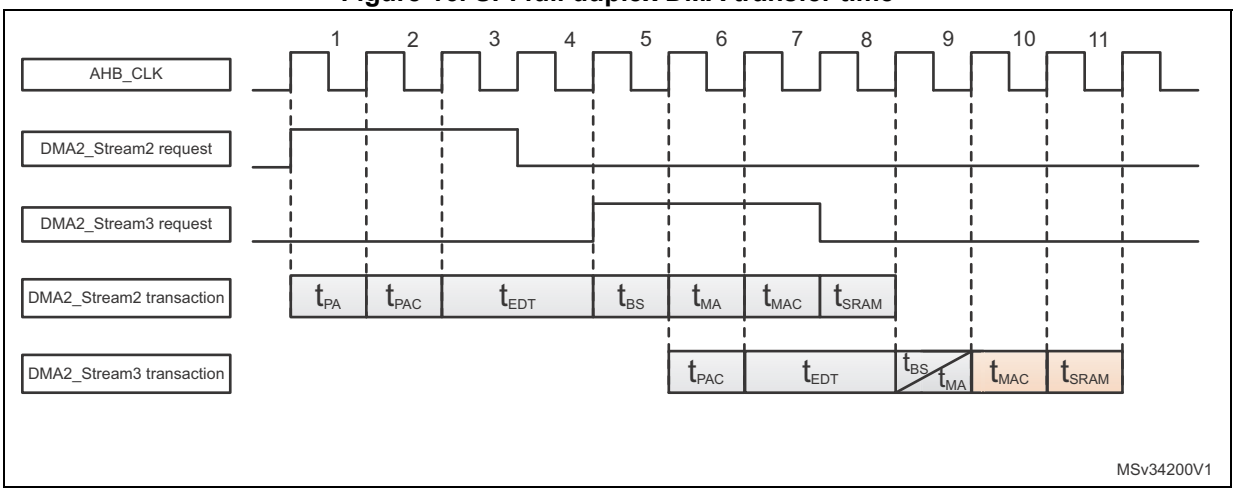
The AHB frequency is equal to the APB2 frequency (84 MHz) and SPI1 is configured to operate at the maximum speed (42 MHz). DMA2_Stream2 (SPI1_RX) is triggered before DMA2_Stream3 (SPI1_TX), which is triggered two AHB cycles later.

With this configuration, the CPU is polling infinitely on the I2C1_DR register. Knowing that the I2C1 peripheral is mapped on APB1 and that the SPI1 peripheral is mapped on APB2, the system paths are the following:

- Direct path for DMA2 to access APB2 (not through bus matrix),
- CPU accesses APB1 through bus matrix.

The aim is to demonstrate that the DMA timings are not impacted by the CPU polling on APB1. The following figure summarizes the DMA timing for Transmit and Receive modes, as well as the time scheduling for each operation:

Figure 16. SPI full duplex DMA transfer time



This figure illustrates the following conclusions:

- CPU polling on APB1 is not impacting the DMA transfer latency on APB2.
- For the DMA2_Stream2 (SPI1_RX) transaction, at the eighth AHB clock cycle, there is no bus matrix arbitration since it is supposed that the last master that accessed the SRAM is DMA2 (so no re-arbitration is needed).
- For the DMA2_Stream3 (SPI1_TX) transaction, this stream anticipates the read from SRAM and writes it on the FIFO and then, once triggered, the DMA peripheral port (destination is SPI1) starts operation.
- For DMA2_Stream3, the DMA peripheral arbitration phase (1 AHB cycle) is executed during the DMA2_Stream2 bus synchronization cycle.

This optimization is always executed like this when the DMA request is triggered before the end of a current DMA request transaction.

4 Tips and warnings while programming the DMA controller

4.1 Software sequence to disable DMA

To switch off a peripheral connected to a DMA stream request, it is mandatory to:

1. switch off the DMA stream to which the peripheral is connected,
2. wait until the EN bit in DMA_SxCR register is reset (“0”).

Only then can the peripheral be safely disabled. DMA request enable bit in the peripheral control register should be reset ("0") to guarantee that any pending request from peripheral side is cleared.

Note: In both cases, a Transfer Complete Interrupt Flag (TCIF in DMA_LISR or DMA_HISR) is set to indicate the end of transfer due to the stream disable.

4.2 DMA flag management before enabling a new transfer

Before enabling a new transfer, the user must ensure that the Transfer Complete Interrupt Flag (TCIF) in DMA_LISR or DMA_HISR is cleared.

As a general recommendation, it is advised to clear all flags in the DMA_LIFCR and DMA_HIFCR registers before starting a new transfer.

4.3 Software sequence to enable DMA

The following software sequence applies when enabling DMA:

1. Configure the suitable DMA stream.
2. Enable the DMA stream used (set the EN bit in the DMA_SxCR register).
3. Enable the peripheral used. DMA request enable bit in the peripheral control register should be set ("1").

Note: If the user enables the used peripheral before the corresponding DMA stream, a “FEIF” (FIFO Error Interrupt Flag) may be set due to the fact the DMA is not ready to provide the first required data to the peripheral (in case of memory-to-peripheral transfer).

4.4 Memory-to-memory transfer while NDTR=0

When configuring a DMA stream to perform a memory-to-memory transfer in normal mode, once NDTR reaches 0, the Transfer Complete is set. At that time, if the user sets the enable bit (EN bit in DMA_SxCR) of this stream, the memory-to-memory transfer is automatically re-triggered again with the last NDTR value.

4.5 DMA peripheral burst with PINC/MINC=0

DMA Burst feature with peripheral address increment (PINC) or memory address increment (MINC) disable allows to address internal or external (FSMC) peripherals supporting Burst

(embedding FIFOs). This mode ensures that this DMA stream cannot be interrupted by other DMA streams during its transactions.

4.6 Twice-mapped DMA requests

When the user configures two (or more) DMA streams to serve the same peripheral request, software should ensure that the current DMA stream is completely disabled (by polling the EN bit in the DMA_SxCR register) before enabling a new DMA stream.

4.7 Best DMA throughput configuration

When using STM32F4xx with reduced AHB frequency while DMA is servicing a high-speed peripheral, it is recommended to put the stack and heap in the CCM (which can be addressed directly by the CPU through D-bus) instead of putting them on the SRAM, which would create an additional concurrency between CPU and DMA accessing the SRAM memory.

4.8 DMA transfer suspension

At any time, a DMA transfer can be suspended to be restarted later on or to be definitively disabled before the end of the DMA transfer.

There are two cases:

- The stream disables the transfer with no later-on restart from the point where it was stopped: there is no particular action to do, except to clear the EN bit in the DMA_SxCR register to disable the stream and to wait until the EN bit is reset. As a consequence:
  - The DMA_SxNDTR register contains the number of remaining data items at the moment when the stream was stopped so that the software can determine how many data items have been transferred before the stream was interrupted.
- The stream suspends the transfer in order to resume it later by re-enabling the stream: to restart from the point where the transfer was stopped, the software has to read the DMA_SxNDTR register after disabling the stream (EN bit at “0”) to know the number of data items already collected. Then:
  - The peripheral and/or memory addresses have to be updated in order to adjust the address pointers.
  - The SxNDTR register has to be updated with the remaining number of data items to be transferred (the value read when the stream was disabled).
  - The stream may then be re-enabled to restart the transfer from the point where it was stopped.

Note: In both cases, a Transfer Complete Interrupt Flag (TCIF in DMA_LISR or DMA_HISR) is set to indicate the end of transfer due to the stream interruption.

4.9 Take benefits of DMA2 controller and system architecture flexibility

The idea behind this section is to show how take benefits from flexibility offered by STM32 architecture and DMA controller. As an illustration of this flexibility we will see how to invert DMA2 AHB peripheral and memory ports and preserve correct managing of peripherals data transfers. In order to achieve this and take over control of regular DMA2 behavior we need to review working model of DMA2.

As both DMA2 ports are connected to AHB BusMatrix, and having symmetric connection with AHB slaves, this architecture lets traffic flow in one or another direction over the Peripheral and Memory ports depending on software configuration.

4.9.1 Inverting transfers over DMA2 AHB ports consideration

Software has flexibility to configure DMA2 stream transfer mode according to its needs. Depending on this configuration one DMA2 AHB port would be programmed in the Read direction, and the other in the Write direction.

Table 8 shows DMA AHB port direction vs. transfer mode configuration.

Table 8. DMA AHB port direction vs. transfer mode configuration

Transfer mode	DMA2 AHB memory port	DMA2 AHB peripheral port
Memory to Peripheral	**Read** direction	**Write** direction
Peripheral to Memory	**Write** direction	**Read** direction
Memory to Memory	**Write** direction	**Read** direction

Now focusing on traffic flow, as shown in *Section 2.2.2: DMA transfer states*, Transfers on Peripheral port are triggered by peripheral requests, Transfers on Memory port are triggered either by the FIFO threshold (when FIFO mode is used) or immediately after peripheral read (when Direct mode is used).

When managing peripherals with DMA2 memory port we need to take care of pre-triggered transfers, buffered transfers, and last data management:

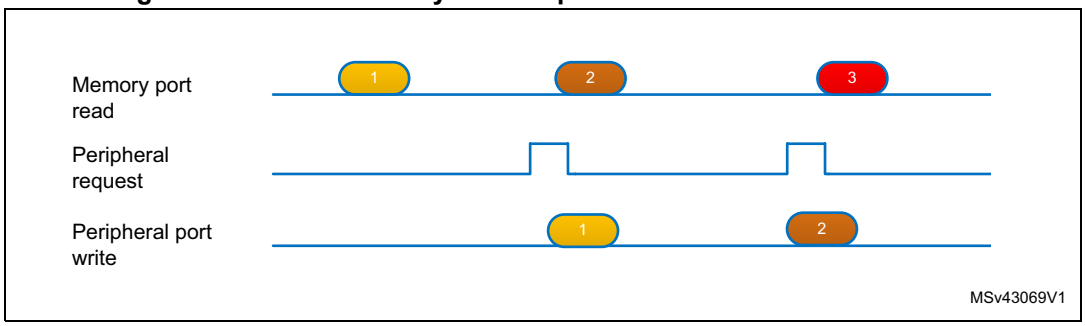
Pre-triggered transfer:

As described in *Section 2.2.2: DMA transfer states* when memory to peripheral transfer mode is configured (reading data over memory port) DMA anticipates the peripheral's access and reads data as soon as DMA stream is enabled. One data is buffered in direct mode and up to 4 x 32-bit words when DMA FIFO is enabled.

When managing peripheral reads over DMA2 memory port, software must ensure that peripheral is enabled before enabling DMA in order to guarantee validity of first DMA access.

Figure 17 illustrates DMA accesses over memory and peripheral ports vs. peripheral triggers.

Figure 17. DMA in Memory-to-Pheripheral transfer mode



Managing last data read

DMA controller features a 4 x 32-bit words FIFO per stream is used to buffer the data between AHB ports. When managing peripheral reads over DMA memory port, software must ensure that 4x extra words are read from the peripheral. This is to guarantee that last valid data are transferred-out from DMA FIFO

Buffered transfers when DMA direct mode is disabled:

When writing data over memory port to a peripheral in indirect mode (FIFO mode enabled), the software needs to take care that access over this port is triggered by programmed FIFO threshold. When threshold is reached data are transferred-out from FIFO to destination over memory port.

When writing to a register (for example GPIOs not having a FIFO), the data from the DMA FIFO will be written successively to destination.

Last but not least, when swapping peripheral management from peripheral port to memory port, the software needs to re-consider transfer size and address increment configuration.

As described in *Section 1.1.5: Transfer size*, transfer size is defined by peripheral side transfer width (byte, half-word, word) and by the number of data items to be transferred (value programmed in DMA_SxNTDR register). According to new DMA configuration when inverting ports the programmed value in DMA_SxNTDR register may need to be adjusted.

4.9.2 Example for inverting Quad-SPI transfers over DMA2 AHB ports consideration

In this example DMA_S7M0AR is programmed with Quad-SPI Data register address, DMA_S7PAR programmed with data buffer address (e.g Buffer in SRAM). In the DMA_S7CR register the DMA2 stream direction must be configured in Peripheral to Memory transfer mode when writing to Quad-SPI. DMA2 stream direction must be configured in Memory to Peripheral transfer mode when reading from Quad-SPI.

4x Extra words (32-bits) are needed for read operation, in order to guarantee that the last data is transferred-out from DMA FIFO to RAM memory.

Table 9. Code Snippet

Write operation	Read operation
/* Program M0AR with QUADSPI data register address */ DMA2_Stream7->M0AR = (uint32_t)&QUADSPI->DR; /* Program PAR with Buffer address */ DMA2_Stream7->PAR = (uint32_t)&u32Buffer[0]; /* Write number of data items to transfer */ DMA2_Stream7->NDTR = 0x100; /* Configure DMA : MSIZE=PSIZE=0x02 (Word), CHSEL=0x03 (QUADSPI), PINC=1, DIR=0x00 */ DMA2_Stream7->CR = DMA_SxCR_PSIZE_1 | DMA_SxCR_MSIZE_1 | 3ul<<25 | DMA_SxCR_PINC; /* Enable DMA request generation */ QUADSPI->CR |= QUADSPI_CR_DMAEN; /* Write the DLR Register */ QUADSPI->DLR = (0x100* 4)-1; /* Write to QUADSPI DCR */ QUADSPI->CCR = QUADSPI_CCR_IMODE_0| QUADSPI_CCR_ADMODE_0| QUADSPI_CCR_DMODE| QUADSPI_CCR_ADSIZE| QUAD_IN_FAST_PROG_CMD; /* Write the AR Register */ QUADSPI->AR = 0x00ul; /* Enable the selected DMA2_Stream7 by setting EN bit */ DMA2_Stream7->CR |= (uint32_t)DMA_SxCR_EN; /* Wait for the end of Transfer */ while((QUADSPI->SR & QUADSPI_SR_TCF) != QUADSPI_SR_TCF);	/* Program M0AR with QUADSPI data register address */ DMA2_Stream7->M0AR = (uint32_t)&QUADSPI->DR; /* Program PAR with Buffer address */ DMA2_Stream7->PAR = (uint32_t)&u32Buffer[0]; /* Write number of data items to transfer */ DMA2_Stream7->NDTR = 0x100; /* Configure DMA : MSIZE=PSIZE=0x02 (Word), CHSEL=0x03 (QUADSPI), PINC=1, DIR=0x01 */ DMA2_Stream7->CR = DMA_SxCR_PSIZE_1 | DMA_SxCR_MSIZE_1 | 3ul<<25 | DMA_SxCR_PINC| DMA_SxCR_DIR_0; /* Enable DMA request generation */ QUADSPI->CR |= QUADSPI_CR_DMAEN; /* Write the DLR Register */ QUADSPI->DLR = ((0x100+4)* 4)-1; /* Write to QUADSPI DCR */ QUADSPI->CCR = QUADSPI_CCR_IMODE_0| QUADSPI_CCR_ADMODE_0| QUADSPI_CCR_DMODE| QUADSPI_CCR_ADSIZE| QUADSPI_CCR_FMODE_0|QUAD_OUT_FAST_READ_CMD; /* Write the AR Register */ QUADSPI->AR = 0x00ul; /* Enable the selected DMA2_Stream7 by setting EN bit */ DMA2_Stream7->CR |= (uint32_t)DMA_SxCR_EN; /* Wait for the end of Transfer */ while((DMA2_Stream7->CR & DMA_SxCR_EN) == DMA_SxCR_EN);

Note: The limitation of the data corruption when DMA2 is managing in parallel AHB and APB2 transfers (refer to product errata sheets to identify impacted STM32F2/F4 MCUs) can be overcome by swapping DMA2 peripheral and memory ports, as described in this section.

4.10 STM32F7 DMA transfer and cache maintenance to avoid data incoherency

When the software is using cacheable memory regions for the DMA source/or destination buffers, it must trigger a cache clean before starting a DMA operation to ensure that all the data are committed to the subsystem memory. When reading the data from the peripheral after the completion of DMA transfer, the software must perform a cache invalidate before reading the updated memory region.

It is preferable to use non-cacheable regions for DMA buffers. The software can use the MPU to set up a non-cacheable memory block to use as a shared memory between the CPU and DMA.

5 Conclusion

The DMA controller is designed to cover most of the embedded use case applications by:

- Giving flexibility to firmware to choose the suitable combination between 16 streams X 16 channels (eight for each DMA),
- Reducing the total latency time for a DMA transfer, thanks to dual AHB port architecture, and direct path to APB bridges avoiding CPU stall on AHB1 access when DMA is servicing low-speed APB peripherals,
- FIFOs implementation on DMA allows more flexibility to firmware to configure different data sizes between source and destination, and speeds-up transfers when using incremental burst transfer mode.

6 Revision history

Table 10. Document revision history

Date	Revision	Changes
04-Feb-2014	1	Initial release.
06-Aug-2015	2	Added *Section : Reference documents*. Document scope extended to STM32F2 and STM32F4 Series and list of reference manuals updated. Removed note 1 in *Table 1: STM32F427/437 and STM32F429/439 DMA1 request mapping* and *Table 2: STM32F427/437 and STM32F429/439 DMA2 request mapping*. Removed Table *DMA1 request mapping for STM32F401 line* and Table *DMA2 request mapping for STM32F401 line.* Updated *Section 4.1: Software sequence to disable DMA* and *Section 4.3: Software sequence to enable DMA*.
23-Jun-2016	3	Added: – *Section 4.9: Take benefits of DMA2 controller and system architecture flexibility* – *Section 4.10: STM32F7 DMA transfer and cache maintenance to avoid data incoherency* – Updated: – *Introduction* – *Section 2: System performance considerations* – *Section 2.1.3: BusMatrix arbitration and DMA transfer delays worst case* – *Figure 14: DMA request arbitration* Removed: – *Table 1: Applicable products*