How to use the LPDMA for STM32 MCUs

Introduction

This application note relates to the low power DMA (LPDMA). The LPDMA is a system peripheral, and is a single-port manager on the AHB bus. It is used to transfer data between peripherals and/or memories via linked lists. LPDMA channels provide power-efficient data transfers at system level and unload the CPU.

The aim of this document is not to rewrite the LPDMA dedicated section in the reference manual, but to provide some performance-oriented features and programming guidelines to the system developer.

This document is based on the combined capabilities of the LPDMA and of a given peripheral which is assisted by the LPDMA for data transfers. It focuses on the key points the user should consider to optimize system performance and match the application requirements.

This application note explains the rationale, and gives recommendations about:

- LPDMA channel allocation
- LPDMA transfer priority assignment
- LPDMA source/destination programming, in terms of data width

Table 1. Applicable products

Type	Products
Microcontrollers	STM32C5 series

1 General information

This application note applies to the STM32 series microcontrollers that are Arm® Cortex® core-based devices.

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

Reference documents

[1] Reference manual STM32C5 Arm®-based 32-bit MCUs (RM0522)

[2] STM32C55xxx datasheet (DS14928

[3] STM32C562xx datasheet (DS14927)

[4] STM32C5A3xG/593xG/591xG device errata (ES0677)

[5] STM32C531xx/532xx/542xx device errata (ES0676)

[6] STM32C551xx/552xx/562xx device errata (ES0661)

2 LPDMA general guidelines

2.1 LPDMA overview and instances

The LPDMA controller is used to perform programmable data transfers between memory-mapped peripherals and/or memories via linked lists, upon the control of an off-loaded CPU.

The LPDMA is a single 32-bit AHB manager and system peripheral. Most of the peripherals and memories are connected to it, giving a lot of flexibility and increasing system performance when data transfers are required.

A linked list is a programmed data structure in the memory to prepare each LPDMA channel for chaining and scheduling DMA data transfers.

A LPDMA instance consists of a set of concurrent channels, to serve concurrent requests coming from different peripherals in DMA mode (for peripheral-to-memory transfers or for memory-to-peripheral transfers), or/and coming from the software (for memory-to-memory transfers to unload the CPU). Any data transfer is performed in linear addressing mode, either at a fixed memory-mapped address (typically for peripheral register access), or at a contiguously incremented address (typically for SRAM memory access).

The following table depicts the LPDMA instances for each STM32 series, and how many channels can be used for a given LPDMA instance.

Table 2. LPDMA instances

Product	LPDMA instances
STM32C5	**STM32C53xxx/542xx**: LPDMA1 4 channels + LPDMA2 4 channels **STM32C55xxx/562xx**: LPDMA1 8 channels + LPDMA2 4 channels **STM32C59xxx/5A3xx**: LPDMA1 8 channels + LPDMA2 8 channels

In the STM32C5 series:

- At the hardware level, there are 2 instances of the LPDMA with up to 8 channels each, named as LPDMA1 and LPDMA2 in the STM32C5 series microcontrollers. The two instances are equally connected to the AHB bus matrix, peripherals, memories, and any addressed AHB/APB target. They are also equally connected in terms of input signals like DMA requests or DMA triggers, and in terms of output signals like DMA channels transfer complete signals.
- At the software level, the programmer is simplified with the view of the sum of all available LPDMA channels, where all LPDMA1 channels are indexed from 0 to the number of LPDMA1 channels minus one, and LPDMA2 channels are indexed starting from the next incremented integer.

2.2 LPDMA channel allocation

The user must allocate one channel for a LPDMA transfer.

For a given LPDMA instance, any LPDMA channel has same feature and performance, except for the peripheral early termination. A LPDMA channel, when implemented with this feature, can support the early termination of the DMA data transfer from the peripheral which does also support this feature.

Only two STM32 peripheral DMA requests support the early peripheral termination: the JPEG encoder output request (jpeg_tx_dma), and the I3C RX-FIFO request (i3c_rx_dma). These two DMA requests may be present or not in the device. For each STM32 series, the next table depicts the list of LPDMA channel(s) and the list of peripheral request(s) that both support supporting the peripheral early termination.

Table 3. LPDMA channel number with peripheral early termination

Product	Channel number	FIFO size	Addressing mode
STM32C5	LPDMA1	x = 0 and x = 7	i3c1_rx_dma
	LPDMA2	x = 0 and x = 7	

Consequently, for example in the STM32C5 series, for the i3c1_rx_dma, the user must allocate either the LPDMA1 channel 0, LPDMA1 channel 7 (not present in STM32C53xxx/STM32C542xx), LPDMA2 channel 0, or LPDMA1 channel 7 (only present in STM32C59xxx/STM32C5A3xx).

2.3 LPDMA channel priority

A LPDMA has a programmable channel priority mechanism to arbitrate the requested transfers between the concurrent channels before issuing the read plus write data transfer via its 32-bit AHB manager port.

One priority value is assigned to each data transfer for competing versus other concurrent ones. The LPDMA arbiter can then grant and schedule the data transfer on one manager port. At user level, this is prepared at channel level via PRIO[1:0] in LPDMA_CxCR when the channel is not active.

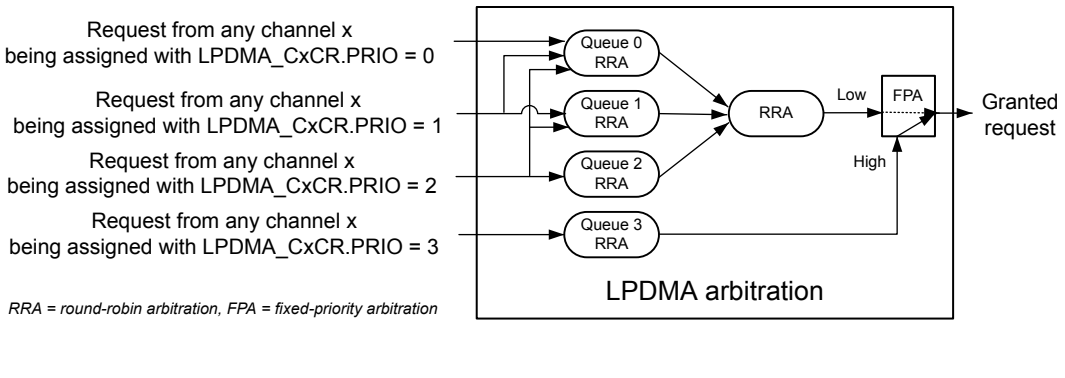
The LPDMA has a single 32-bit manager port for AHB transfers. There is no FIFO inside the LPDMA. The LPDMA performs a data transfer which consecutively combines an AHB single data read transaction at the memory-mapped source address and an AHB single data write at the memory-mapped destination address. This combined read-plus-write transfer is named as a direct transfer (at the opposite of the FIFO-based transfer on a GPDMA and HPDMA).

This direct read-plus-write transfer over its manager port is arbitrated between the different ready LPDMA channels. Priority-based arbitration is implemented as combinational logic and occurs with 0-cycle clock latency between any requested read AHB transaction.

The LPDMA implements programmable arbitration logic that allows the user to adjust the channel bandwidth and latency according to the following rules:

- The priority of the requested direct transfer is programmable from 0 to 3.
- Requests having the same priority are handled with a round-robin arbitration scheme.
- Priority 3 is recommended for time-sensitive requests because it is handled with a fixed higher priority scheme over priorities 0 to 2.
- The residual bandwidth is shared by requests of priority 0 to 2 by implementing a weighted round-robin allocation for these non time-sensitive channels.
- The different weights are monotonically resulting from the programmed channel priorities, queue 0 having the lowest weight.

Figure 1. LPDMA arbitration policy



The user needs to assign the correct level of priority to a LPDMA channel on which peripherals/memories are connected, to get an acceptable service quality for this channel:

- No timing error (no peripheral register underrun/overrun) on peripheral side
- Acceptable latency between the data transfer request and the servicing of this request
- Acceptable impact on service quality of the other channels

Channel priorities should be carefully assigned to fit the application timing requirements. A requested direct transfer of priority 0 to 2 that is ready to be scheduled again on the manager port, may be delayed by the round-robin arbiter for the switch to the execution of another concurrent direct transfer of priority 0 to 2. Such a request is also first preempted by any time-sensitive requested transfers.

A LPDMA channel mapped with a request from a critical timer is typically allocated to the time-sensitive queue, so that the related TIMER registers can be updated with the lowest latency. LPDMA requests from other peripherals may be assigned a priority 2 or 1, and the memory-to-memory transfers may be finally assigned with the lowest priority 0 (best effort traffic).

This recommendation about channel priority may be tuned to fit specific application requirements (see the reference manual for more details about arbitration and weighted round-robin priority).

A data transfer is typically performed for each linked-list item of a given channel. The user may have chained it with a next data transfer via the same channel, by having programmed a linked-list data structure in memory. Once the data transfer of a given linked-list item (LLI) is completed, the LPDMA reads/fetches automatically the next linked-list data structure, and updates internally its registers, before executing the next data transfer as described.

The link-transfer priority is given by the assigned priority to its channel, in the same way as the data transfer. The link-transfer consists of a series of up to six 32-bit single reads for the maximal update of the six 32-bit registers. Between every single 32-bit read for the LLI update, the LPDMA takes one extra clock cycle for the arbitration phase.

2.4 LPDMA data width

Given that the system bus is 32 bits wide, it is recommended to set the DMA source and destination data width to 32 bits by default (S/DDW_LOG2[1:0] in LPDMA_CxTR1) to minimize bus utilization.

An LPDMA single transfer (which reads a single data from the source and writes a single data to the destination) performs an elementary programmed data transfer and defines it by:

- A source programmed data width for the read transaction: 8-, 16-, or 32-bit (via SDW_LOG2[1:0] in LPDMA_CxTR1)
- A destination programmed data width for the write transaction: 8-, 16-, or 32-bit (via DDW_LOG2[1:0] in LPDMA_CxTR1)

The LPDMA implementation does not change the programmed data width and issues it unchanged on the AHB manager port.

The LPDMA performs limited data handling between a read operation and a write operation. The LPDMA does not manipulate data between two distinct data transfers from the source before generating the destination transfer. The user can configure only the following data handling during a direct transfer between the read operation and the write operation:

- Add padding bytes as zero or by sign extension to match the destination data width when the destination data width is greater than the source data width.
- Truncate bytes from the source data and align left or right until the destination data width matches the required value. Use this method when the destination data width is less than the source data width.

Such programmed data handling is not intended and cannot be used to minimize the 32-bit AHB system bus load. To optimize system bandwidth usage:

- For memory-to-memory transfers: Use a programmed 32-bit source and destination data width, as the 32-bit AHB system bus and SRAMs natively support this configuration. The source address, destination address, and buffer size must be 32-bit aligned.
- For peripheral-to-memory or memory-to-peripheral transfers: When supported by the peripheral and the application, use a 32-bit source data width when reading the 32-bit peripheral data register, or a 32-bit destination data width when writing to the 32-bit peripheral data register. If 32-bit data width is not supported, use a 16-bit source data width instead of a 1-byte data width.

In any case, programming a 32-bit source data width imposes a constraint on the application. The programmed LPDMA block size to be transferred from the source must be a multiple of the source data width (BNDT[15:0] in the LPDMA_CxBR1 register compared with SDW_LOG2[1:0] in the LPDMA_CxTR1 register). This requirement may not be acceptable. For example, if the source peripheral data register does not support a 32-bit data width, or if the application cannot ensure that the source block size is a multiple of 32-bit words, such as when data is received through a UART byte stream.

2.5 LPDMA request

2.5.1 Software request for memory-to-memory transfers

The LPDMA transfer can be initiated by setting a software request via SWREQ in LPDMA_CxTR2 register. This is the case for memory-to-memory transfers, which do not rely on any hardware request signal coming from the memory.

The memory-to-memory mode is also used for certain timer configurations in DMA mode and for GPIO configuration in DMA mode (see Section 3.5).

2.5.2 Hardware request for peripheral-to-memory and memory-to-peripheral transfers

As an alternative to a software request, a LPDMA transfer may be requested and initiated by an input hardware request coming from a peripheral, for peripheral-to-memory and memory-to-peripheral LPDMA transfers. The user must program SWREQ = 0 and the field REQSEL in LPDMA_CxTR2 register in order to select the proper hardware peripheral request among the product-dependent list of the mapped LPDMA request signals. Refer to the LPDMA implementation section of the product reference manual for this list and mapping.

2.6 LPDMA trigger

An LPDMA transfer can be conditioned by a hardware trigger, as defined by the fields TRIGM, TRIGPOL and TRIGSEL in the LPDMA_CxTR2 register.

The hardware trigger may be driven by a peripheral (such as timers), by the LPDMA itself (channel transfer complete), or by other peripherals. Refer to the product reference manual for more details about the trigger functionalities, and more specifically in the LPDMA implementation section for the product-dependent list of triggers, which can be mapped to the LPDMA.

3 Exhaustive peripherals, memories, and LPDMA configuration

In STM32 devices, most analog and digital peripherals can be used in DMA mode and rely on LPDMA capability to handle data transfers without CPU intervention.

The overall system performance of a LPDMA transfer results from the peripheral configuration and its location (AHB or APB), as well as the allocated LPDMA channel configuration for this transfer: mainly the channel priority and the source/destination data width of the data transfer.

In the following sections, an exhaustive list of peripherals/memories embedded into the STM32 devices is presented to guide the developers for optimal LPDMA configuration, and to get the best benefit in term of performance.

For each peripheral, the following list of items is covered to help with peripheral and LPDMA configuration:

- Peripheral request signals to LPDMA
  - All peripheral requests are of single type, except for the lptimer_ue_dma request coming from LPTIM..
  - lptimer_ue_dma request (lptimer update event) requires a specific LPDMA programming : the control bit BREQ must be set in LPDMA_CxTR2 register. BREQ is present in any channel.
  - I3C receive data request to LPDMA (i3c_rx_dma, of single type) is specific: the number of data to be received during an I3C SDR private read message may be early completed by the external I3C device. This peripheral early termination feature is natively supported by the LPDMA hardware, provided that the PFREQ control bit is set in DMA_CxTR2 register. It is important to note that, depending on the product, the PFREQ bit may be present in some channels only (see), and consequently an appropriate LPDMA channel for i3c_rx_dma request should be selected accordingly.
- Peripheral bus (which AHB/APB)
- Peripheral register access
  - Most of the LPDMA requests are tightly coupled to one specific peripheral register.
  - On the contrary, timers offer the flexibility to address different peripheral registers to fit different application requirements.
- LPDMA access type: read or write
- LPDMA data width: 8-bit, 16-bit or 32-bit

3.1 LPDMA configuration with internal memories and general transfers guidelines

The STM32 devices embed multiple internal SRAMs.

3.1.1 Peripheral-to-memory and memory-to-peripherals transfers

Memory can be targeted as source or/and destination by the LPDMA to perform data transfer during respectively a requested memory-to-peripheral or peripheral-to-memory transfer.

A LPDMA programmed 32-bit data width is preferred to reduce the 32-bit AHB bus load on the manager port to read from or write to the memory. This reduces the traffic to the memory by a ratio of four versus performing byte access, and by half versus handling half-words.

This configuration should be used as much as possible despite that the LPDMA cannot decorrelate the source data width from the destination data width, and cannot perform FIFO queuing, neither data handling like packing nor unpacking between different reads and writes, on the contrary of GPDMA and HPDMA.

The peripheral may not support its peripheral data register to be accessed by 32-bit words or 16-bit half-words (See next section for peripheral details). Using 32-bit as source and destination data width should be first privileged if the accessed peripheral register is a 32-bit data register, else 16-bit should be privileged if the accessed peripheral register is a 16-bit data register, to avoid that only a single byte is transferred on a system bus clock cycle.

Furthermore, even when 32 bit (or 16-bit) data is supported on the peripheral side, there is no byte(s) padding inside the LPDMA up to the block size: the source block data size (BNDT[15:0] in LPDMA_CxBR1 register) must be a multiple of the 32-bit (or 16-bit) source data width. This assumes that the application can guarantee that:

- Received data stream is a multiple of 32-bit words (or 16-bit half-words respectively) in peripheral-to-memory mode.
- Transmitted data stream is a multiple of 32-bit words (or 16-bit half-words respectively) in memory-to-peripheral mode.

3.1.2 Memory-to-memory transfers

Memory can be also used for memory-to-memory transfers to offload the CPU. Then, the recommended 32-bit data width for source and destination should be used for the best performance, since the system bus and the SRAMs natively support transfers of 32-bit words.

It can be noted that a GPIO is accessed in memory-to-memory mode: it must be initiated by a software request, there is no DMA hardware request driven by any GPIO.

The lowest priority (priority 0) is recommended as default for a channel with memory-to-memory transfers in order to:

- Get a best-effort traffic for this type of transfer.
- Allow more bandwidth and lower latency to peripherals in DMA mode with a higher priority 1 or 2 via the higher weighted round-robin arbitration.
- be systematically preempted by the time-sensitive peripherals in DMA mode with the highest priority 3 via the fixed-priority arbitration.

See Section 2.3 for information about channel priority and arbitration.

3.2 LPDMA configuration for analog peripherals

3.2.1 Analog-to-digital converter ADC

STM32 devices can embed one or several analog-to-digital converters.

This analog-to-digital (A/D) converter implementation proposes a DMA mode. The LPDMA request signal is used by the hardware to request a new single converted sample to be read by the LPDMA whenever the peripheral needs it. The LPDMA reads the ADC_DR register. This ADC 16-bit or 32-bit data register must be accessed by a 32-bit word-aligned read access, with a programmed source data width being typically either 8-bit, 16-bit, or 32-bit and with a fixed addressing.

Table 4 summarizes the main peripheral characteristics: hardware request signal to the LPDMA, peripheral bus on which the ADC is connected, and peripheral register from/to which the LPDMA must be programmed to read or write.

Table 4. ADC peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
adcx_dma (x = 1, 2, 3)	AHB2	8-word FIFO	Read ADC_DR	The ADC1/2/3 data register (ADC_DR) must be read with a 32-bit aligned address, whatever the data width.

The table below shows the supported LPDMA source data widths when reading the peripheral register.

Table 5. LPDMA programming for ADC

LPDMA source data width	Comments
8-bit, 16-bit, or 32-bit	Depends on ADC settings.

The ADC generates an analog watchdog trigger signal which can be used as trigger for any LPDMA transfer.

3.2.2 Digital-to-analog converter DAC

STM32 devices can embed a digital-to-analog (D/A) converter DAC.

The digital-to-analog converter DAC is DMA capable without embedded FIFO. Even if the DAC has a dual channel LPDMA capability (completely independent), it can also use a single LPDMA channel to transfer within a single LPDMA channel the data to the dual converter. The DAC proposes then a LPDMA optimization reducing the number of LPDMA channel to use from two to one. It also reduces the number of transactions on the AHB bus by packing the data of the two converters in a single data transfer (in dual mode).

Table 6. DAC peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
dac_ch1_dma/ dac_ch2_dma	AHB2	None	Write (any) DAC_DHRyRx with y = 12, 8 and x = 1, 2	The DAC data hold register must be written at the specific 32-bit aligned address, depending on the used DAC format (8- or 12-bit, left-aligned or right-aligned) and depending on the considered channel (1 or 2).

The table below shows the supported LPDMA destination data widths when writing to the peripheral register.

Table 7. LPDMA programming for DAC

LPDMA destination data width	Comments
8-bit, 16-bit, or 32-bit	Depends on DAC settings.

3.3 LPDMA configuration for communication peripherals

3.3.1 Inter-integrated circuit interface (I2C)

STM32 devices embed several I2C instances.

All transfers are handled in single data mode by the LPDMA controller, since the I2C sends a new LPDMA request each time a byte data needs to be written/ read, or a control word is needed to be written, depending on the considered LPDMA request signal.

The LPDMA performs a 32-bit aligned access to the related data or control register.

The 8-bit data receive register must be read with a programmed 8-bit source data width, with a fixed addressing and a 32-bit aligned access.

The 8-bit data transmit register must be written with a programmed 8-bit destination data width, with a fixed addressing and a 32-bit aligned access.

The 32-bit control word must be written with a programmed 32-bit destination data width, and with a fixed addressing (if consecutive updates are required).

One of the channels with 2-word FIFO can be allocated (as well one of the channels with 8-word FIFO), especially if the memory to be written is an external memory.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, the peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 8. I2C peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
i2cx_rx_dma (x=1, 2)	I²C on APB1	None	Read 8-bit I2C_RXDR	Request for a new received data byte to be read by the LPDMA. Must be read with a 32-bit aligned address.
i2cx_tx_dma (x=1, 2)			Write 8-bit I2C_TXDR	Request for a new data byte to be written by the LPDMA for transmission. Must be read with a 32-bit aligned address.
i2c_evc_dma(1)			Write 32-bit I2C_CR2	Request for a new command word to be written by the LPDMA.

1. *Not applicable for STM32C5*

The table below shows the supported LPDMA settings:

- The source data width when reading the peripheral register
- The destination data width when writing to the peripheral register.

Table 9. LPDMA programming for I2C

LPDMA data width for source and destination	Comments
i2c_tx, i2c_rx: 8-bit i2c_evc: 32-bit(1)	Depends on I2C settings.

1. *Not applicable for STM32C5*

3.3.2 Improved inter-integrated circuit interface (I3C)

The STM32 devices embed one I3C on the APB bus

The I3C interface handles the communication between this device and others: sensors and host processor(s), connected on an I3C bus. The I3C bus is a two-wire, serial single-ended, multidrop bus, intended to improve a legacy I2C bus.

All transfers are handled in single data mode by the LPDMA controller, since the I3C peripheral sends a new LPDMA request each time:

- A control word needs to be written.
- A status word needs to be read.
- A data byte/word needs to be written or read depending on the considered LPDMA request signal.

The LPDMA performs a 32-bit aligned access to the related data or control register.

The 32-bit control word must be written with a programmed 32-bit destination data width, and with a fixed addressing.

The 32-bit status word must be read with a programmed 32-bit source data width, and with a fixed addressing.

The data transmit register can be accessed by bytes (via I3C_TDR) or by words (via I3C_TDWR). It must be written with a programmed respectively 8-bit or 32-bit destination data width, with a fixed addressing, and a 32-bit aligned access.

The allocated DMA channel for the data receive bytes/words must be programmed with PFREQ = 1 in DMA_CxTR2. This is to support that an external I3C device may early complete the data transmission. Since this PFREQ field is present only in some of the channels, one of these channels must be allocated.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, the peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 10. I3C peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
1I3c_rx_dma	APB1 for I3C1	8-byte RX-FIFO	Read 8-bit I3C_RDR or 32-bit I3C_RDWR	Received data must be read with a 32-bit aligned address.
i3c1_tx_dma	None	8-byte TX-FIFO	Write 8-bit I3C_TDR or 32-bit I3C_TDWR	To be transmitted data must be written with a 32-bit aligned address.
i3c1_tc_dma		2-word C-FIFO	Write 32-bit I3C_CR	To be transmitted control word.
i3c1_rs_dma		2-word S-FIFO	Read 32-bit I3C_SR	Received status word

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 11. LPDMA programming for I3C

LPDMA data width for source and destination	Comments
i3c_tx_dma/ i3c_rx_dma: 8-bit or 32-bit i3c_tc_dma/ i3c_rs_dma: 32-bit	Depends on I3C settings.

3.3.3 Universal synchronous/asynchronous/low-power receiver transmitter (USART/UART/LPUART)

STM32 devices embed multiple USART/UART instances. Each instance can have a LPDMA channel linked to the transmit or receive queue.

All transfers are handled in single data mode by the LPDMA controller, since the USART/UART sends a new LPDMA request each time a data needs to be written/read, depending on the considered LPDMA request signal. The LPDMA performs a 32-bit aligned access to the related data register.

The 9-bit data receive register must be read with a programmed 8-, 16-, or 32-bit source data width, with a fixed addressing, and with a 32-bit aligned access, depending on the format.

The 9-bit data transmit register must be written with a programmed 8-, 16-, or 32-bit destination data width, with a fixed addressing, and with a 32-bit aligned access, depending on the format.

The USART/UART/LPUART have transmit and receive FIFO in order to avoid as much as possible overrun/ underrun situation which may break the data transfer flow. Table 12Table 21 summarizes the main peripheral characteristics: the different hardware request signals to the LPDMA, the peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/ write.

Table 12. U(S)ART and LPUART peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
(lp)u(s)art_rx_dma	APB1 for USART2/3/6 and UART4/5/7 APB2 for USART1 APB3 for LPUART1	8x 12-bit width RXFIFO (9 bits for Rx data + 3 bits for errors) 8x 9-bit width TXFIFO(1)	Read USART_RDR or LPUART_RDR	Request for a new received data to be read by the LPDMA. Must be read with a 32-bit aligned address, whatever is the data width.
(lp)u(s)art_tx_dma			Write USART_TDR or LPUART_TDRR	Request for a new data to be written by the LPDMA for transmission. Must be written with a 32-bit aligned address, whatever is the data width.

1. *FIFO can be enabled/disabled by software.*

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 13. LPDMA programming for U(S)ART and LPUART

LPDMA data width for source and destination	Comments
16-bit or 32-bit read/write in 9-bit mode and 8-bit, 16-bit or 32-bit read/write in 8-bit and 7-bit modes	Depends on U(S)ART/LPUART settings.

3.3.4 Serial peripheral interface (SPI)

STM32 devices embed multiple SPI instances. All of them can have a LPDMA channel linked to the transmit or receive queue.

All transfers are handled in single data mode by the LPDMA controller, since the peripheral sends a new LPDMA request each time a data is needed to be written/read, depending on the considered LPDMA request signal. Then LPDMA performs a 32-bit aligned access to the related data register.

The data receive register must be read with a programmed 8-, 16-, or 32-bit source data width, with a fixed addressing, and a 32-bit aligned access.

The data transmit register must be written with a programmed 8-, 16-, or 32-bit destination data width, with a fixed addressing, and a 32-bit aligned access.

The SPI has a transmit and a receive FIFO in order to avoid as much as possible overrun/ underrun situation which may break the data transfer flow.

The SPI can pack/unpack the received/to-be-transmitted byte-stream versus the written/read 32-bit word stream at the data register level. It is recommended to optimize the data transfer to reduce the overall system latency, and to decrease the bus bandwidth on the LPDMA manager. Decrease the load on the APB peripheral bus is also recommended when applicable. It is recommended to perform 32-bit data width access to the peripheral register to decrease by a ratio of 4 the bus traffic vs a byte-stream access on the peripheral side.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 14. SPI peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
spi_rx_dma (x = 1,2,3)	STM32C5 (APB2 for SPI1 APB1 for SPI2/3)	16-byte RXFIFO and TXFIFO for SPI1, SPI2 and SPI3(1)	Read SPI_RXDR	Request for a new received data to be read by the LPDMA. Must be read with a 32-bit aligned address, whatever is the data width.
spi_tx_dma (x = 1,2,3)			Write SPI_TXDR	Request for a new data to be written by the LPDMA for transmission. Must be written with a 32-bit aligned address, whatever is the data width.

1. *FIFO can be enabled/disabled by software.*

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 15. LPDMA programming for SPI

LPDMA data width for source and destination	Comments
32-bit recommended. If not applicable, 8-bit or 16-bit.	Depends on SPI settings.

3.4 LPDMA configuration for mathematical peripherals

3.4.1 CORDIC coprocessor

There is one instance of the CORDIC in the STM32C5 devices.

The CORDIC provides hardware acceleration of mathematical functions commonly used in motor control, metering, signal processing, and many other tasks, with 16- or 32-bit fixed-point. The LPDMA transfers can take place in read and write mode.

The CORDIC allows multiple register read/write by the LPDMA:

- ARG1 and possibly ARG2 as input arguments (depending on NARGS in CORDIC_CSR)
  If both arguments need to be input for computation, they are input through two single LPDMA requests.
- RES1 and RES2 as primary and secondary results, respectively (depending on NRES in CORDIC_CSR)
  If both computed results need to be read by the LPDMA, the CORDIC asserts two LPDMA requests in single mode.

The data width can be one of the following:

- 16-bit format
  The primary argument ARG1 is written to the least significant half-word whereas the secondary argument ARG2 is written to the most significant half-word. Thanks to this data packing, the argument is sent within a single LPDMA data transfer, which improves the bus bandwidth by minimizing the amount of DMA transfers to perform.
- 32-bit format
  There are two LPDMA data accesses to fill the two arguments (if CORDIC configuration requires two arguments).

The table below summarizes the main peripheral characteristics: hardware request signal to the LPDMA, peripheral bus on which the peripheral is connected, and peripheral register from/to which the LPDMA must be programmed to read/write.

Table 16. CORDIC peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
cordic_rd_dma	AHB1	None	Read CORDIC_RDATA	Request for a new computed output data to be read by the LPDMA. Must be read with a 32-bit aligned address.
cordic_wr_dma			Write CORDIC_WDATA	Request for a new input data to be written by the LPDMA for computation. Must be written with a 32-bit aligned address.

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 17. LPDMA programming for CORDIC

LPDMA data width for source and destination	Comments
16-bit or 32-bit	Depends on CORDIC settings.

3.5 LPDMA configuration for timers

3.5.1 Low-power timers (LPTIM)

The STM32 devices embed multiple low-power timers.

The low-power timers are 16-bit timers that benefit from the ultimate developments in power-consumption reduction. Thanks to their clock source diversity, the LPTIMx are able to keep running in all power modes except Standby mode.

All transfers are in single data mode.

The LPTIM is LPDMA capable with two types of requests:

- Input capture
  - From timer channel 1, lptimx_ic1_dma
  - From timer channel 2, lptimx_ic2_dma
    Then the LPDMA reads the 16-bit LPTIMx_CCRy (y = 1, 2) registers to get the new latched counter value in input capture mode.
- Update event
  - An lptimx_ue_dma request is a LPDMA block-type request, and requires a specific hardware connection between the timer and the LPDMA to be activated by programming BREQ = 1 (and SWREQ = 0) in LPDMA_CyTR2 (y is the allocated LPDMA channel).
  - The block consists of up to three single writes to the LPTIMx to update up to two of the following registers, in order to update the generation of a PWM signal: LPTIMx_CCR1 (capture compare register 1 for the duty cycle), LPTIMx_CCR2 (capture compare register 2 for the duty cycle), and LPTIMx_RCR (repetition counter register), followed by a mandatory update of the LPTIMx_ARR for the period.

Any LPTIMx generates two output signals (lptimx_ch1 and lptimx_ch2), which are used as trigger for any LPDMA transfer and/or for a DAC conversion.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, the peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 18. LPTIM peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
lptimx_icy_dma (x = 1 and y = 1, 2)	LPTIM1 on APB3	None	Read 16-bit/32-bit LPTIMx_CCRy	Request to read the capture compare register on an input capture event (connected input signal). Must be a 32-bit aligned address access.
lptim_ue_dma			Write up to 3x 16-bit or 3x 32-bit registers among LPTIMx_CCR1, LPTIMx_CCR2, LPTIMx_RCR and mandated LPTIMx_ARR	Request to up to three writes of the LPTIMx registers on an update event. Must be a 32-bit aligned address access.

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 19. LPDMA programming for LPTIM

LPDMA data width for source and destination	Comments
16-bit or 32-bit	Depends on LPTIM settings. A channel with priority 3 is recommended if the application requires a minimized latency whatever is other concurrent traffic.

3.5.2 Basic timers (TIM6/7)

The basic timers consist of a 16-bit auto-reload upcounter driven by a programmable prescaler. They can be used as generic timers for time-base generation.

The timer update request to the LPDMA, (timx_upd_dma with x = 6, 7), occurs when the counter overflows (when a programmed period of time elapses). The LPDMA performs a 32-bit write to TIMx_ARR (x = 6, 7) and including 4 dithering bits), in order to update the timer output period and waveform for the next counter overflow.

See Section 2.3 for more details, especially for time-sensitive usage of the timer.

Both TIM6 and TIM7 generate a trigger signal, respectively tim6_trgo and tim7_trgo, which may be used as trigger for any LPDMA transfer and/or for a DAC conversion.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, peripheral bus on which the peripheral is connected, and corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 20. TIM6/7 peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
timx_upd_dma/ timx_up_dma (x = 6, 7)	APB1	None	Write 32-bit TIMx_ARR	Request on an update event (on a counter overflow)

The table below shows the supported LPDMA destination data width when writing to the peripheral register.

Table 21. LPDMA programming for TIM6/7

LPDMA destination data width	Comments
32-bit	A channel with priority 3 is recommended if the application requires a minimized latency whatever is other concurrent traffic.

3.5.3 General-purpose timers

The STM32 devices embed multiple general-purpose timers.

The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler. TIM2/3/4/5/15/16/17 can be used for various purposes such as measuring the pulse lengths of input signals (input capture), or generating output waveforms (output compare, PWM, possibly complementary PWM with dead-time insertion).

All read/write transfers are handled in single data mode by the LPDMA controller, since the timer sends a new LPDMA request each time a 32-bit or 16-bit data/control is needed to be written/read.

A timer request can be used for the following actions:

- A 32-bit write to TIMx_ARR in basic mode, to update the output waveform and period when counter over/ underflows.
- A 32-bit read to TIMx_CCRy (y = 1 to 4), to get the latched counter value in input capture mode.
- A 32-bit write to TIMx_CRRy (y = 1 to 4), to update the output compare register in output compare mode (duty cycle).
- A 16-bit/32-bit write to TIMx_RCR (x = 15, 16, 17), in PWM output/modulation mode, to update the repetition counter while the period is given by TIMx_ARR, and duty cycle is given by TIMx_CCR.
- A 32-bit read to TIMx_CCR1, and a 32-bit read to TIMx_CCR2 in input PWM mode to get the latched counter for respectively the period and the pulse width.
- A series of 32-bit writes to the intermediate TIMx_DMAR register in timer ‘burst’ mode, or a series of 32-bit reads from the intermediate TIMx_DMAR register in timer ‘burst’ mode.
  The timer supports multiple single accesses to redirect the value to/from the timer registers through one entry point. This timer capability to write multiple timer registers through a single entry-point is called ‘burst’. The LPDMA can perform the data transfers in single mode multiple times upon any programmed hardware event/request (among timx_chy_dma, timx_upd_dma, timx_trg_dma, timx_com_dma possible events).

A timer request to LPDMA is quite programmable and flexible: it is related to the occurrence of a programmed event versus the real-time value of the timer counters. A timer request is not tightly coupled to write/read a specific timer register. Any timer register may be read/written depending on the application (the contrary of the other peripherals).

It is recommended to allocate:

- The port 0 for peripheral register access (direct connection to AHB/APB bridge).
- The port 1 for memory access (to balance traffic), a channel with 2-word FIFO, and a high priority (to get a transfer latency as less as possible sensitive to other traffic).

See Section 2.3 for more details, especially for time-sensitive usage of the timer.

For some application needs, a timer request to LPDMA can be used to initiate a data transfer not involving timer register read/write, but other memory-mapped location, like for memory-to-memory transfers. In this case, the LPDMA port allocation strategy must not be considered with respect to the timer register location, but with the source/destination locations.

TIM2 and TIM15 generate a trigger signal (respectively tim2_trgo and tim15_trgo) which can be used as trigger for any LPDMA transfer.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, peripheral bus on which the peripheral is connected, and corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 22. TIM2/3/4/5/15/16/17 peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
timx_chy_dma/ timx_ccy_dma (x = 2 to 5 and y = 1 to 4) or (x = 15 and y = 1, 2) or (x = 16, 17 and y = 1)	APB1 for TIM2/3/4/5 APB2 for TIM15/16/17	None	32-bit write TIMx_ARR in basic mode or 32-bit write TIMx_CCRy in output compare/PWM mode or 32-bit read TIMx_CCRy in input capture mode or a series of 32-bit writes to TIMx_DMAR or a series of 32-bit writes to TIMx_DMAR	Request on an input capture event or an output compare event
timx_upd_dma/timx_up (x = 2 to 5, 15 to 17)				Request on an update event
timx_trg_dma/ timx_trig_dma (x = 2 to 5, 15)				Request on a trigger event
tim15_com_dma				Request on a commutation event

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 23. LPDMA programming for TIM2/3/4/5/15/16/17

LPDMA data width for source and destination	Comments
32-bit	Depends on TIM settings. A channel with priority 3 is recommended if the application requires a minimized latency whatever is other concurrent traffic.

3.5.4 Advanced-control timers (TIM1/8)

The advanced-control timers consist of a 16-bit auto-reload counter driven by a programmable prescaler. TIM1/8 can be used for various purposes such as measuring the pulse lengths of input signals (input capture), or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

All read/write transfers must be handled in single data mode by the LPDMA controller, since the timer sends a new LPDMA request each time a 32-bit or 16-bit data/control is needed to be written/read.

A timer request can be used for the following actions:

- A 32-bit write to TIMx_ARR in basic mode, to update the output waveform and period when counter over/underflows.
- A 32-bit read to TIMx_CCRy (x = 1, 8 and y = 1 to 4), to get the latched counter value in input capture mode.
- A 32-bit write to TIMx_CRRy (x = 1, 8 and y = 1 to 4), to update the output compare register and duty cycle in output compare mode.
- A 16-bit/32-bit write to TIMx_RCR (x = 1, 8), in PWM output/modulation mode, to update the repetition counter while period is given by TIMx_ARR, and duty cycle is given by TIMx_CCR.
- A 32-bit read to TIMx_CCR1 and a 32-bit read to TIMx_CCR2, in input PWM mode, to get the latched counter for respectively the period and the pulse width.
- A series of 32-bit writes to the intermediate TIMx_DMAR register in timer ‘burst’ mode, or a series of 32-bit reads from the intermediate TIMx_DMAR register in timer ‘burst’ mode. The timer supports multiple single accesses to redirect the value to/from the timer registers through one entry point. This timer capability to write multiple timer registers through a single entry-point is called 'burst'. The LPDMA can perform the data transfers in single mode multiple times upon any programmed hardware event/request (among timx_chy_dma, timx_upd_dma, timx_trg_dma, timx_com_dma possible events).

A timer request to LPDMA is quite programmable and flexible: it is related to the occurrence of a programmed event versus the real-time value of the timer counters. A timer request is not tightly coupled to write/read a specific timer register, and any timer register can be read/written, depending on the application (the contrary of the other peripherals).

See Section 2.3 for more details, especially for time-sensitive usage of the timer.

For some application needs, a timer request to LPDMA can be used to initiate a data transfer not involving timer registers read/write, but other memory-mapped location, like for memory-to-memory transfers. In this case, the LPDMA port allocation strategy must be not be considered with respect to the timer registers location, but with the source and destination locations.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, peripheral bus on which the peripheral is connected, and corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 24. TIM1/8 peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
timx_chy_dma/ timx_ccy_dma (x = 1, 8 and y = 1 to 4)	APB2	None	No tightly coupled timer register read/write Typically a 32-bit read/write to TIMx_CCRy or a 16-/32-bit read/write to TIMx_RCR or a series of 32-bit read/writes to TIMx_DMAR (x = 1, 8 and y = 1 to 4)	Request on an input capture event or an output compare event
timx_upd_dma/ timx_up_dma (x = 1, 8)				Request on an update event
timx_trg_dma/ timx_trig_dma (x = 1, 8)				Request on a trigger event
timx_com_dma (x = 1, 8)				Request on a commutation event

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 25. LPDMA programming for TIM1/8

LPDMA data width for source and destination	Comments
32-bit (or 16-bit for TIMx_RCR)	Depends on TIM settings. A channel with priority 3 is recommended if the application requires a minimized latency whatever is other concurrent traffic.

3.6 LPDMA configuration for cryptographic peripherals

3.6.1 Hash processor (HASH)

The HASH is a fully compliant implementation of the secure hash algorithm (SHA-1, SHA-224, SHA-256), the MD5 (message-digest algorithm 5) hash algorithm, and the HMAC (keyed-hash message authentication code) algorithm. HMAC is suitable for applications requiring message.

The HASH implementation proposes a DMA mode. The HASH generates LPDMA requests allowing the user to perform single LPDMA write transfers. The LPDMA request signal is used by the hardware to request a new input to be written by the LPDMA whenever the peripheral needs it. The request is to write a 32-bit single data value. The LPDMA writes the HASH_IN register. This HASH register must be accessed by with a destination source data width being 32-bit with a fixed addressing.

The HASH implementation uses a 16-word FIFO for the HASH_IN register.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, peripheral bus on which the peripheral is connected, and corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 26. HASH peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
hash_in_dma	AHB2	16-word FIFO	Write 32-bit HASH_IN	HASH input data register (HASH_IN) is written by a single word.

The table below shows the supported LPDMA destination data width when writing to the peripheral register.

Table 27. LPDMA programming for HASH

LPDMA destination data width	Comments
32-bit	-

3.6.2 AES hardware accelerator (AES)

The AES hardware accelerator encrypts/decrypts data, using an algorithm and implementation fully compliant with the advanced encryption standard (AES) defined in federal information processing standard (FIPS) publication 197.

The AES supports CTR, GCM, GMAC, CCM, ECB, and CBC chaining modes for 128- or 256-bit key size.

The AES performs 128-bit block cryptographic processing, and implements a 128-bit input buffer/FIFO and a 128-bit output buffer/FIFO.

All transfers must be handled in single data mode by the LPDMA controller, since the peripheral sends a new LPDMA request each time an input/output 32-bit data is needed to be written/read. Then LPDMA performs a read/write access to the related 32-bit data register, with a programmed source/destination data width of 32-bit and a fixed addressing.

The table below summarizes the main peripheral characteristics: different hardware request signals to the LPDMA, the peripheral bus on which the peripheral is connected, and the corresponding peripheral register from/to which the LPDMA must be programmed to read/write.

Table 28. AES peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
aes_in_dma	AHB2	128-bit/4-word input buffer	Write 32-bit AES_DINR	Request for a new 32-bit input data to be written by the LPDMA
aes_out_dma		128-bit/4-word output buffer	Read 32-bit AES_DOUTR	Request for a new 32-bit output data to be read by the LPDMA

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 29. LPDMA programming for AES

LPDMA data width for source and destination	Comments
32-bit	-

3.6.3 Secure AES coprocessor (SAES)

The SAES is the secure AES version. All data described in are valid. The only difference is the LPDMA request signal names (saes_in_dma and saes_out_dma).Section 3.6.2

3.7 LPDMA configuration for external memories

3.7.1 Extended-SPI interface (XSPI)

STM32C59xx/C5Axx devices embed one instance (XSPI1) of the Extended-SPI interface .

The XSPI supports most external memories, such as serial PSRAMs, serial NAND and serial NOR Flash memories, HyperRAM™ and HyperFlash™ memories.

XSPI in indirect mode proposes a DMA mode for read/write data transfers to its peripheral registers. XSPI embeds a 32-byte FIFO.

The LPDMA request signal, xspi1_dma, is used by the hardware to request a new single to be read/written by the LPDMA whenever the peripheral needs it. The LPDMA reads/writes the XSPI_DR data register. This data register must be accessed by 32-bit aligned read access, with a programmed source/destination data width being either 8-bit, 16-bit or 32-bit and with a fixed addressing.

The selected number of bytes of the data width of the peripheral data register must also be set accordingly on the XSPI side by programming the FIFO threshold FTHRES[4:0] in the XSPI_CR register.

Using a 32-bit data width is preferred for minimizing the AHB bus load, unless the application requires a byte-level or half-word data management.

The table below summarizes the main peripheral characteristics: hardware request signal to the LPDMA, peripheral bus on which the XSPI is connected, and peripheral register from/to which the LPDMA must be programmed to read or write.

Table 30. XSPI peripheral information

Peripheral request to LPDMA	Peripheral bus	Peripheral FIFO (unit, size)	Peripheral register access	Comments
xspi1_rx_dma	AHB_XSPI (dedicated target of bus matrix)	32-byte FIFO	Read/Write 8-bit, 16-bit or 32-bit XSPI_DR	It is recommended to access XSPI_DR with a 32-bit data width when it is allowed by the application to use efficiently the system bus bandwidth. FTHRES[4:0] is the XSPI_CR register must be set accordingly. Must be a 32-bit aligned address access.

The table below shows the supported LPDMA settings:

- The source data widths when reading the peripheral register
- The destination data widths when writing to the peripheral register.

Table 31. LPDMA programming for XSPI

LPDMA data width for source and destination	Comments
8-bit, 16-bit or 32-bit	Depends on XSPI settings.

4 System performance

The microcontroller architecture needs to be understood to control that the different LPDMA requested transfers can be efficiently performed: transferred in due time (with an acceptable latency and completion time versus the application requirements) while optimizing the bus bandwidth efficiency and minimizing the resource utilization/ load (bus/resource utilization, bus overhead, AHB/APB frequencies).

The previous sections refined the LPDMA characteristics and provided user guidelines to get the best performances for serving a LPDMA data transfer. The following sections describe other key points which impact this performance.

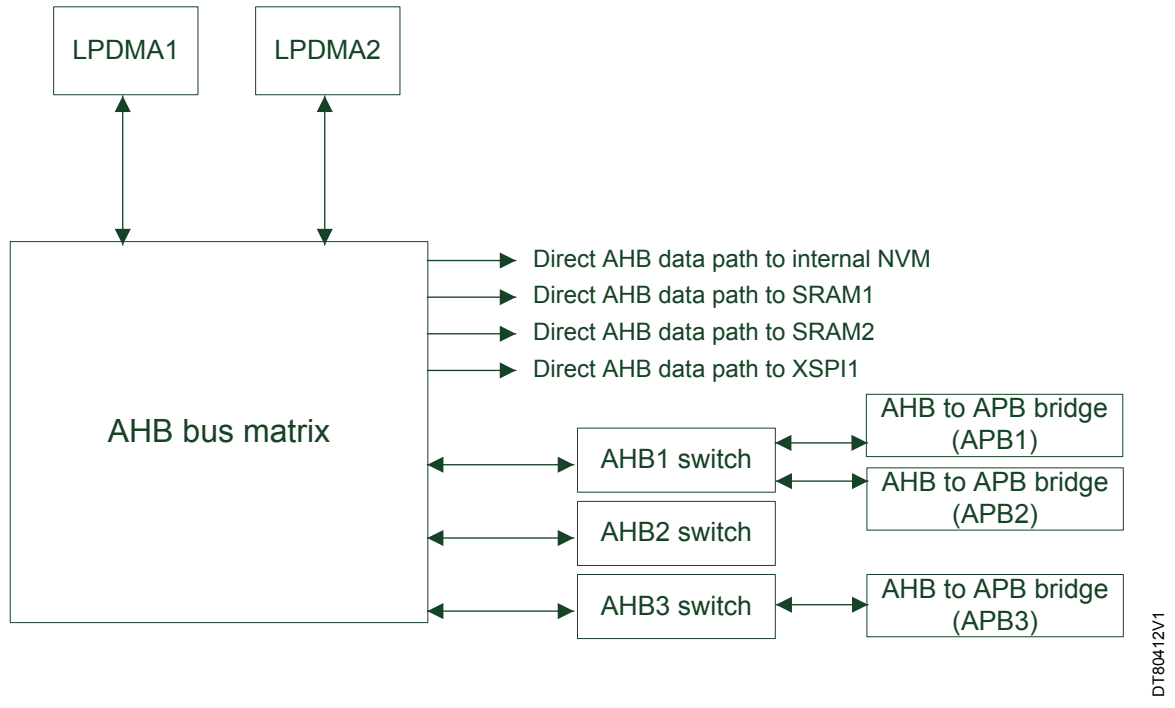
4.1 LPDMA

The LPDMA has a single 32-bit manager port for AHB transfers. There is no FIFO inside the LPDMA. The LPDMA performs a data transfer which consecutively combines an AHB single data read transaction at the memory-mapped source address and an AHB single data write at the memory-mapped destination address. This direct read-plus-write transfer over its manager port is arbitrated between the different ready LPDMA channels.

Priority-based arbitration is implemented as combinational logic and occurs with 0-cycle clock latency between any requested read AHB transaction.

The LPDMA is connected to the AHB bus matrix. There may be more than one LPDMA instance in a STM32 device (see Section 2.1) and then the LPDMA bus manager is connected to a specific bus subordinate of the AHB matrix. Next figure shows a typical example of a STM32 device with two LPDMA instances.

Figure 2. LPDMA manager port connection



The strategy and guidelines to be applied for the LPDMA channel priority assignment are detailed in Section 2.3.

The 32-bit AHB bus architecture consists of:

- A dedicated address bus
- A separated bus for read and write data
- 1-cycle pipelined transfers in terms of address and data (on a same clock cycle, the address of a transfer occurs meanwhile the data phase of the previous transfer)

In terms of performances, LPDMA benefits from the pipelined architecture of the AHB protocol for the address phase and data phase.

4.1.1 LPDMA performance in memory-to-memory transfers

When there is one single channel active per LPDMA instance, then memory-to-memory performance results (at the LPDMA level) in a:

- Latency for the execution of one direct read-plus-write transfer: 3 clock cycles
  - 1 cycle for address read
  - 1 cycle for data read and address write
  - 1 cycle for data write
- Number of consumed AHB bus cycles per direct transfer: 3 clock cycles
  - Including one idle bus clock cycle between back-to-back transfers
  - Throughput between back-to-back direct transfers: 1/3 of the AHB bandwidth

When there are N concurrent active channels per LPDMA instance, then for a given channel memory-to-memory performance results (at the LPDMA level) in a:

- Latency for the execution of one granted direct read-plus-write transfer: 3 clock cycles
- Number of consumed AHB bus cycles per direct transfer: 2 clock cycles
  - No idle bus clock cycle: 0-cycle arbitration penalty and immediate switch to another pending transfer Address read of the channel occurs in the same cycle as the data write of the previous channel
  - Throughput between back-to-back direct transfers: 1/(2*N) of the AHB bandwidth (assuming a same weighted round-robin priority for all channels)

4.1.2 LPDMA performance in peripheral-to-memory and memory-to-peripheral transfers

Back-to-back data transfers to/from the peripheral are limited in performance by the fact that at least four AHB clock cycles must occur between two occurrences of the same peripheral request, due to the protocol of the request/acknowledge hardware signals between a peripheral and the LPDMA.

This means that for transfers with a peripheral (LPDMA in memory-to-peripheral mode or in peripheral-to-memory mode), back-to-back direct transfers can occur every 4 clock cycles provided that the peripheral is able to sustain such a throughput on its DMA output request.

4.2 Bus matrix

The multilayer AHB bus matrix allows parallel access to several shared AHB subordinates from several different AHB managers. The bus matrix improves the transfer parallelism and contributes to reducing the transfer execution time by optimizing the shared resources utilization.

4.2.1 AHB bus definitions

- AHB manager: a bus manager initiates read/write AHB single/burst transfer. Only one manager can get bus ownership at a given time.
- AHB subordinate: a bus subordinate responds to the AHB transfer from a manager. The bus subordinate routes signals back to the manager to inform about the access success, failure or waiting states.
- AHB arbiter: a bus arbiter ensures that only one manager can access a subordinate at a given time.
- AHB bus matrix: a switch matrix that interconnects AHB managers to AHB subordinates. It implements a round-robin AHB arbiter for each access to a subordinate.

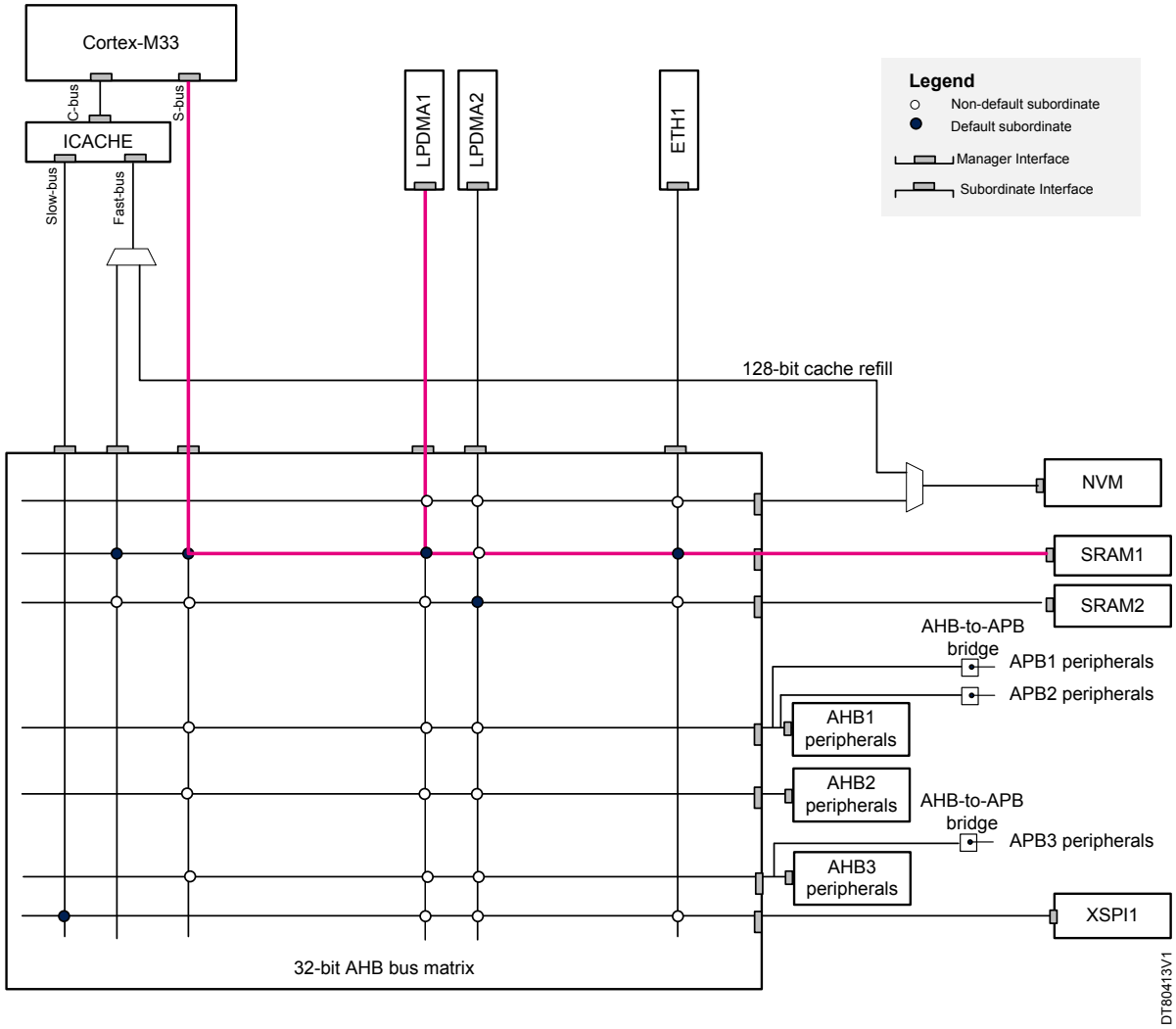
4.2.2 Bus matrix round-robin arbitration

A bus matrix arbitration is implemented for each subordinate to arbitrate its access from concurrent managers. This round-robin arbitration allows a fair distribution of the shared subordinate overtime (see the example in Figure 3) and bounded latency access. The round-robin algorithm is implemented with the following rules:

- A round-robin quantum is one AHB transfer. There is bus switching on every transfer if another manager port waits for access to a subordinate.
- When accessing a subordinate, a manager port gets:
  - 1-cycle arbitration penalty when it first accesses the subordinate (provided that the bus is free / idle)
  - 1-cycle re-arbitration penalty if the previously accessed subordinate differs
  - 0-cycle re-arbitration penalty for LPDMA to be granted to access again the same subordinate with a back-to-back transaction, if there is no other manager waiting for the shared subordinate
- This policy applies whatever the subordinate is a default one or a non-default one.

Figure 3 shows an example where both the CPU and LPDMA try to access SRAM1.

Figure 3. Example of CPU and LPDMA requesting access to SRAM1

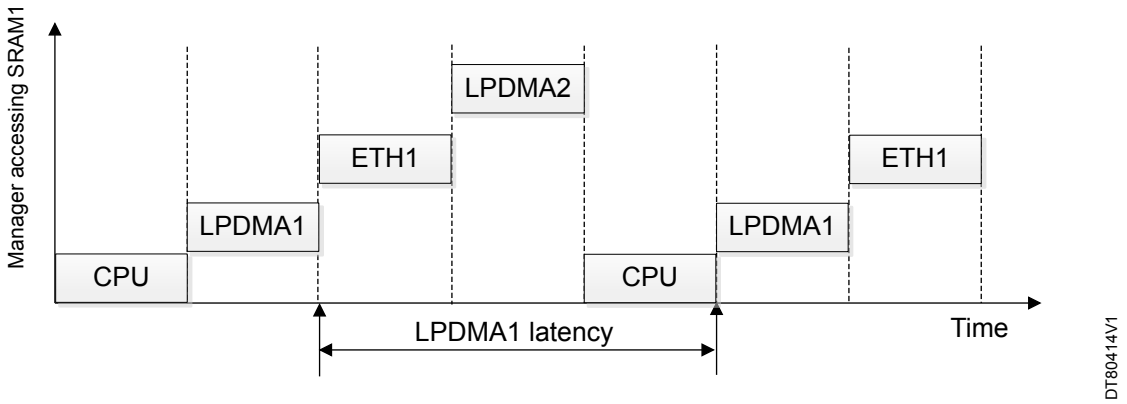


In case of concurrent access from the CPU and the LPDMA1, the bus matrix arbitration rules the access to the SRAM1. If the last access is from the CPU, the LPDMA1 wins the bus during the next access to SRAM1. After the CPU can access again SRAM1.

Figure 2 and Figure 3 show how are connected LPDMA1 and LPDMA2 with the AHB matrix and its subordinates for the STM32C5 series. For other STM32 series, see the memory and bus architecture section of the reference manual: it describes the LPDMA instance(s) connections with the AHB matrix and different SRAMs.

The transfer latency initiated by one manager to access a subordinate depends on the number of other pending transfers initiated by other managers to access the same AHB subordinate. The figure below details another example where four managers requesting to access SRAM1 simultaneously.

Figure 4. Example of four managers requesting an access to SRAM1



The latency associated with the LPDMA AHB direct transfer (AHB single read transaction plus AHB single write transaction) to access the bus and SRAM1 in the example above, is increased by the sum of execution times of all pending requests coming from the other managers. If the ETH1 accesses the SRAM1 with an AHB burst transaction for example, the LPDMA1 access latency is increased by the latency of the ETH1 burst.

Note: The latency due to CPU transfers issued by LDM/STM instructions can be reduced by configuring the compiler to split load/store multiple instructions into single load/store instructions.

4.3 AHB bus switch

STM32 devices embed several AHB switches to decode/multiplex the AHB signals.

A bus switch adds no cycle penalty to a transfer latency/execution time.

4.4 SRAM

4.4.1 Transfer latency and execution time

LPDMA write is implemented as a non-bufferable write. In STM32 devices, bufferable write access is restricted to CPU accessing external memory via the data cache, if any.

SRAM introduces no additional wait-state for AHB access.

For completion at the SRAM bus interface, due to 1-cycle arbitration of the AHB matrix when the manager accesses a non-default subordinate, Table 32 gives the latency of a programmed direct read-plus-write transfer with SRAM, assuming that the subordinate SRAM is not concurrently accessed by another bus manager than the LPDMA. Else the latency for the execution of the granted AHB transfer(s) from the concurrent granted manager(s) by the AHB matrix round-robin arbitration should be added.

Table 32. LPDMA read-plus-write transfer to SRAM latency

LPDMA1/2 transfers from and to SRAM1/2	Direct read-plus-write latency (cycles) (if no other manager accesses the same subordinate)
LPDMA1 <-> SRAM1	3
LPDMA1 -> SRAM2	3
LPDMA2 -> SRAM2	3
SRAM1 -> LPDMA1 -> SRAM2	4
SRAM1 -> LPDMA2 -> SRAM2	4

4.4.2 Transfer rate/throughput and bandwidth

Beside this SRAM transfer latency, it is important to consider the performance in terms of how fast (consecutive) transfers can happen (maximum achievable data throughput).

SRAM throughput is the number of transferred data per second. For a given AHB operating frequency, the SRAM throughput is the number of (recommended 32-bit) data transferred per AHB clock cycles.

SRAM introduces no penalty. A transfer rate of one 32-bit read or write per cycle can then be sustained.

The table below provides the achieved transfer rate for copying a buffer via a LPDMA channel

in memory-to-memory mode, when both the source buffer and the destination buffer are in SRAM.

The table highlights that the user should privilege and program a 32-bit data width for LPDMA source and destination, to benefit from the 32-bit system bus. Else 16-bit data width should be privileged with a performance degradation ratio of ½. 8-bit data width consumes 4x more AHB bus clock cycles.

Table 33. LPDMA in memory-to-memory mode, SRAM (read+write) transfer rate and throughput

Programmed LPDMA source & destination data width	Number of cycles (per direct transfer)	Transfer rate with a 100 MHz AHB clock (Mtransfer/s)	Throughput with a 100 MHz AHB clock (MByte/s)
8-bit	3 to 4	25 to 33	25 to 33
16-bit			50 to 66
32-bit			100 to 133

4.5 AHB-to-APB bridges

An APB peripheral generates a hardware request to the LPDMA. The allocated and configured LPDMA channel generates a direct read-from-memory-plus-write transfer from the corresponding APB register (in memory-to-peripheral mode), or to the APB register (in peripheral-to-memory mode).

Section 3 lists the APB peripherals working in DMA mode, and their corresponding LPDMA programming. It is recommended to maximize the programmed data width when possible for best performance.

4.5.1 AHB-to-APB clock ratio

The transfer latency to an APB peripheral register is impacted by the clock ratio between APB and AHB clocks. The best case is achieved with a ratio of 1. When the APB clock runs lower than the AHB clock, the latency is increased.

4.5.2 AHB-to-APB bridge transfer latency

Table 34 shows the LPDMA (half direct) transfer latency for a single read or a single write from the AHB interface of the AHB-to-APB bridge to the APB peripheral register. This applies when there is no concurrent or pending transfer from another manager to an APB1 or APB2 peripheral.

Table 34. AHB-to-APB bridge: read or write AHB transaction latency to APB register

AHB/APB clock ratio	Read latency (AHB clock cycles)	Write latency (AHB clock cycles)
1:1	4	4 to 5
1:2	5 to 6	
1:4	9 to 12	
1:8	17 to 24	

As seen previously, LPDMA adds no clock-cycle latency, and the bus matrix may add one clock cycle. Consequently, the table below details the LPDMA transfer latency for completion of a single read or a single write to an APB peripheral register, provided that:

- Neither another LPDMA channel, nor other bus managers generate concurrent traffic to the addressed APB1/2/3 bus.
- The APB peripheral does not insert any wait state.

Table 35. **LPDMA read or write transfer latency to APB register**

AHB/APB clock ratio	Read latency (AHB clock cycles)	Write latency (AHB clock cycles)
1:1	4 to 6	
1:2	5 to 7	
1:4	9 to 13	
1:8	17 to 25	

Revision history

Table 36. **Document revision history**

Date	Version	Changes
03-Mar-2026	1	Initial release.

Contents

List of tables

List of figures