

Introduction to DMA controller for STM32 MCUs

Introduction

Every STM32 family microcontroller features at least one DMA controller intended to offload some data transfer duties from the Cortex® CPU core. This document describes general guidelines about the usage of the basic DMA controller found in most entry-level, mainstream, and low-power STM32 products. The goal is to explain the bus sharing principles and provide hints on efficient usage of the DMA transfer.

Table 1. Applicable products

Type	Product series and part number
Microcontrollers	<ul style="list-style-type: none">• STM32F0 series, STM32F1 series, STM32F3 series• STM32C0 series• STM32G0 series, STM32G4 series• STM32L0 series, STM32L1 series, STM32L4 series, STM32L4+ series, STM32L5 series• STM32U0 series

1 General information

The STM32F0/F1/F3/Cx/Gx/Lx/U0 series 32-bit microcontrollers are based on the Arm[®] Cortex[®]-M processor.

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



2 Bus bandwidth

Bus bandwidth is defined as the amount of data that the bus can transfer in a fixed amount of time, typically per second. It is determined by the clock speed, the bus width, and the bus management overhead. Any STM32 product considered by this document features a 32-bit data width. The clock speed is configurable by the user. The following section provides information about the bus overhead management, and is presenting the bus topology.

2.1 Bus architecture

Generally the DMA transfer spans across three main bus components: an AHB bus matrix, the AHB bus, and the APB bus(es). A DMA controller is connected to the AHB bus matrix via one dedicated AHB master port, as the CPU and possibly another DMA controller. A peripheral served by the DMA is either connected as a slave AHB bus to the bus matrix, or is connected as a slave APB bus after an AHB to APB bridge. A memory is a slave AHB target connected to an AHB bus of the AHB bus matrix.

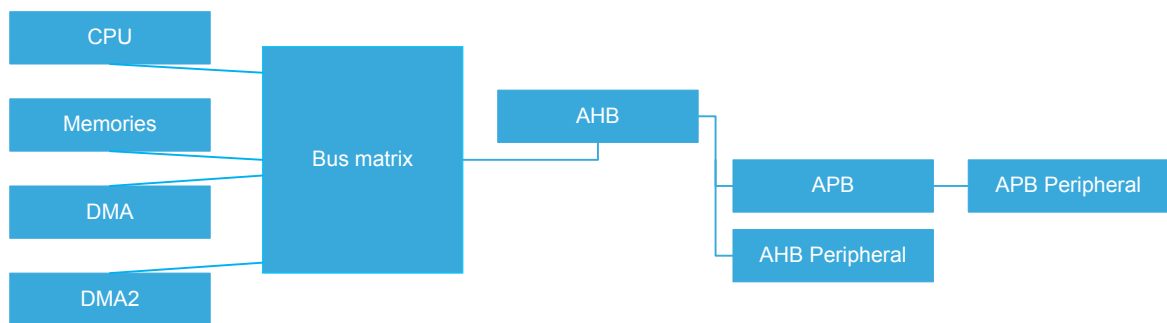
The bus matrix uses master/slave organization. Only CPU and the DMA act as masters, all other connected parts are accessed as slaves.

All the bus resources are shared using round-robin arbitration mechanisms ensuring bus assignment without blocking any process. The arbitration is designed with low latency as a goal, avoiding situations blocking the bus by a single process for too long. The bus is only assigned to the DMA master for single word transfers (shortest possible round-robin quantum) and then the arbitration reassigns the resource.

The assignment of the available bus resources can be focused on bandwidth (bus is assigned for a longer period, minimizing overhead) or on low latency in sharing (simple and fast switching between tasks). The requirements in MCU use cases favor the latter choice. This way it is impossible to use the whole bus bandwidth for a single DMA transfer, but sharing is efficient.

Figure 1. DMA block diagram for products with DMA and DMA2

DMA2 is only present on more complex products (see [Table 2. Differences in DMA implementation](#) for details).



DT48598V2

2.2 Bus matrix

Within the bus matrix, as long as two concurrent AHB transfers are submitted by two separate AHB masters targeting a different AHB bus matrix slave port of the bus matrix, there is no bus matrix arbitration. For example, when the CPU reads instructions from flash memory while the DMA reads data from another memory, they are not limiting each other in any way. Arbitration only takes place when two masters need to access the same slave memory or peripheral.

2.3 AHB

A product has one or more AHB buses, connecting the AHB peripherals to the bus matrix. In some documents all the connections to the bus matrix or within it are referred as AHB, but in this document AHB is referred as a 32-bit wide connection between the bus matrix and the peripherals. Most peripherals are attached to the APB. Only a selected few are directly on the AHB (for example SDIO, AES).

An AHB bus does not provide the data (aka layer) parallelism of the bus matrix, but it runs on the same system clock speed, and provide moderate bandwidth, thanks to its pipelined data/address protocol.

When the CPU initiates a data transfer meanwhile the DMA is transferring a block of data from a source to a destination, the round-robin AHB bus matrix halts the DMA bus access and inserts the CPU access, causing the DMA transfer to have a longer latency.

2.4 APB

A product has one or more 32-bit APB buses.

A DMA data transfer from or to an APB peripheral is first crossing the bus matrix, and the AHB to APB bridge. Within an APB bus, any peripheral is competing with each other and a transfer can occur when the bus is idle or ready.

An APB bus is meant to connect and share several APB peripherals with low bandwidth requirements. APB clock speeds can typically be tuned down from the AHB speed using configurable clock dividers. A high divider ratio yields lower power consumption, but at the cost of lower bandwidth and higher latency. Moreover, the APB buses are connected to AHB using an AHB to APB bridge. Latency added by the AHB:APB bridges is prolonging the bus access period, also reducing the bandwidth usable on the AHB bus.

3 Product specific features

3.1 DMAMUX

In STM32C0, STM32G0, STM32G4, STM32L5, STM32L4Rx, and STM32L4Sx products, the DMA capabilities are enhanced by a DMA request multiplexer (DMAMUX). DMAMUX adds a fully configurable routing of any DMA request from a given peripheral in DMA mode to any DMA channel of the DMA controllers. DMAMUX does not add any clock cycle between the DMA request sent by the peripheral and the DMA request received by the configured DMA channel. It features synchronization of DMA requests using dedicated inputs. DMAMUX is also capable of generating requests from its own trigger inputs or by software.

More detailed information about the multiplexer can be found in the respective reference manuals (RM0432, RM0438, RM0440, RM0444, RM0454, RM0490).

3.2 Presence of TrustZone® security features

The security of the DMA in L5 is leveraged by Arm®TrustZone®-aware attributes configurable independently for each channel. Each channel can be configured as privileged and or secure. Transfers from source and to destination have independent secure flags to match with slave settings such as flash memory watermarks.

Secure attributes specific to a bus master, such as CPU core or DMA are not seen by other masters. The DMA is not affected by the CPU SAU/IDAU settings. It is however upholding its own security settings in cooperation with other components, mainly DMAMUX, and GTZC, capable of triggering an alert interrupt in case of access rights violation.

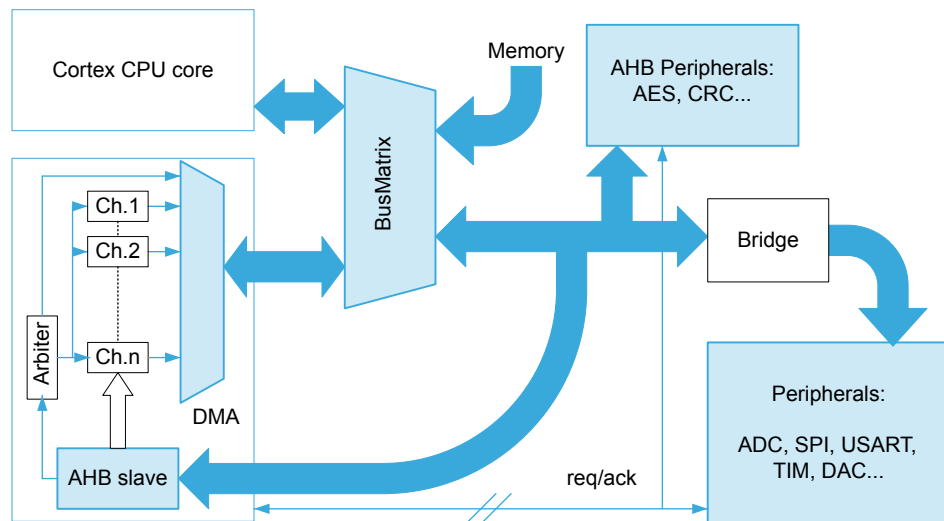
By default the DMA channels are not secure. It is important that the secure code removes the secure channel configuration when the DMA transfer is finished, because the nonsecure code cannot do so.

For more information about TrustZone® security implementation on STM32L5, refer to the reference manual RM0438 or the application note AN5347 available on www.st.com.

4 DMA controller

A DMA controller consists of an arbiter part, assigning channels to the DMA AHB master part. In products where DMAMUX is implemented, channels are assigned to peripherals freely. Typically the channels are preconfigured for different types of block-based data transfers used in the application and activated during application execution as needed. A block-based data transfer consists of a programmed number of data transfers from a source to a destination, as well as a programmed incremented or fixed addressing, start address and data width, each being independent for the source and the destination. The configuration is programmed via the AHB slave port of the DMA controller.

Figure 2. DMA block diagram



DT46599V2

There are two distinct types of transfer available:

- Memory-to-peripheral or peripheral-to-memory**
 When a peripheral is configured in DMA mode, each transferred data is autonomously managed at hardware, and data level between the DMA and the related peripheral via a dedicated DMA request/acknowledge handshake protocol. The mapping of the different DMA request signals, from a given peripheral to a given DMA channel, is listed either inside the DMA section of the reference manual, or in the DMAMUX implementation section of the RM0432, RM0444, and RM0490 product reference manual.
- Memory-to-memory**
 The transfer requires no extra control signal, it is activated by software. A channel is allocated by software, for example to initialize a large RAM data block from flash memory. It is then likely to compete for access to the flash memory with CPU instruction fetching. In any case, DMA arbitration between channels is reconsidered between every transferred data.

5 DMA latency

A latency describes the delay between the request activation of the DMA data transfer and the actual completion of the request. The request is either hardware-driven from a peripheral or is software-driven when a channel is enabled.

5.1 DMA transfer timing

Four steps are required to perform a DMA data transfer. The first step is the arbitration for the bus access. When successful, address computation follows. The third step is a single data transfer itself. The fourth and final step is the acknowledge handshake.

Note: In the simplest case of no back-to-back data transfer, that is, if a block of data is reduced to a single one, the last fourth step is not present.

5.1.1 AHB peripheral and system volatile memory

For example, when storing ADC continuous conversion data in SRAM, the following steps must be followed:

- DMA request arbitration and address computation
- Reading data from the peripheral on AHB (DMA source, in our case ADC on L4x product)
- Writing loaded data in SRAM (DMA destination)

The service time per requested single data, t_S , is given by the equation below:

$$t_S = t_A + t_{RD} + t_{WR}$$

Where:

- t_A is the arbitration time, including address computation
- t_A = minimum of two AHB clock cycles, if there is no higher priority channel with a pending request and if the slave is ready. Time is longer when the AHB peripheral and slave are not ready and adds AHB wait state(s).
- t_{RD} is the peripheral read access time
- t_{RD} = minimum of two AHB clock cycles for an AHB peripheral. More cycles in case of bus sharing.
- t_{WR} is the SRAM write access time
- t_{WR} = 1 AHB clock cycle (single read/write operation) or two AHB clock cycles in case of SRAM read-after-write access on STM32F1 and STM32L1 series. Bridge wait states can be added.

5.1.2 APB peripheral and system volatile memory

The basic scheme of transfer is the same as in previous case, but this time the bus bridge is involved in:

- DMA request arbitration and address computation
- Reading data from the peripheral on the APB (DMA source)
- Writing loaded data in the SRAM (DMA destination)

The service time per channel, t_S , is given by the equation below:

$$t_S = t_A + t_{RD} + t_{WR}$$

Where:

- t_A is the arbitration time, including address computation
 t_A = minimum of two AHB clock cycles, if no higher priority transfer is pending.
- t_{RD} is the peripheral read access time
 t_{RD} = two APB clock cycles as the base minimum, one extra AHB cycle for bridge synchronization if the APB frequency is lower than AHB. See [Section 5.4](#) for additional details.
- t_{WR} is the SRAM write access time
 t_{WR} = same as in the AHB case, one or two AHB cycles.

When the DMA is idle or after the third step has been completed on one channel, the DMA arbiter compares the priorities of any pending DMA request (a request is either hardware requested or software requested). At every data transfer completion, the DMA arbiter examines if there is at least one pending request from another channel. If so, the DMA arbitration switches and selects the most priority channel with a pending request for another data transfer.

As a result, having more active DMA channels improves the bus bandwidth usage, but leads to higher latency as a consequence of the highest priority channel DMA arbitration scheme.

It can be noted that a same transfer but in the opposite direction (that is, peripheral-to-memory) would give a same latency for the DMA data transfer.

5.2 Total service time and parallelism

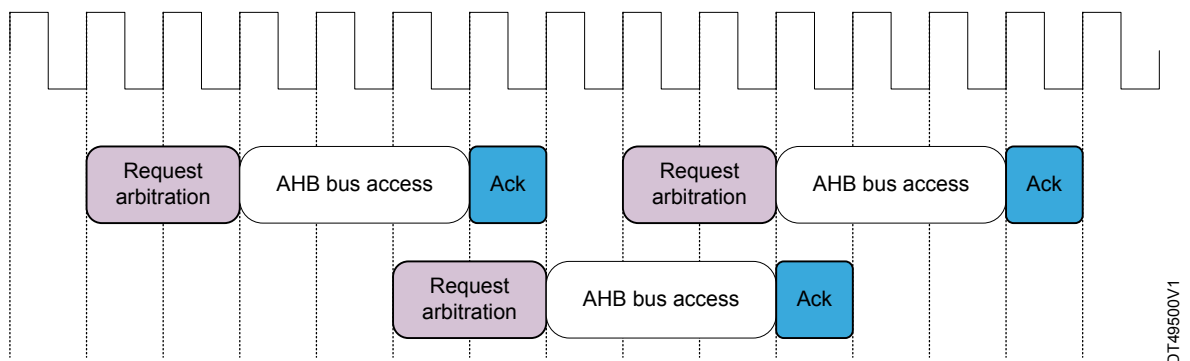
For the case where only one DMA channel is active, a new hardware back-to-back request cannot be handled by the DMA before the completion of the previous one, adding one AHB clock cycle for the final idle phase of the DMA request-acknowledge handshake protocol. For this, the total service time between every request on a same channel must be used.

$t_{TS} = t_A + t_{WR} + t_{RD} + t_{Ack}$, where:

- t_{Ack} is the DMA acknowledge time (closing the handshake between peripheral and DMA)
- $t_{Ack} = 1$ AHB clock cycle

When more than one channel is requesting a DMA transfer, the DMA request arbitration can be performed meanwhile the two last cycles of when the AHB bus is accessed by the DMA. Request arbitration overhead is then masked by the AHB bus transfer time.

Figure 3. Two DMA channels on AHB bus



DT49500V1

In case not only two channels, but two DMA controllers are used (in products that offer this possibility), two DMA transfers can be processed in parallel, as long as they are not conflicting within the bus matrix, not accessing the same slave device. For example, when using STM32L486, a DMA1 transfer from SRAM1 to AES can access the bus matrix simultaneously with DMA2 transfer from flash memory to any APB communication interface. No conflict and arbitration occurs.

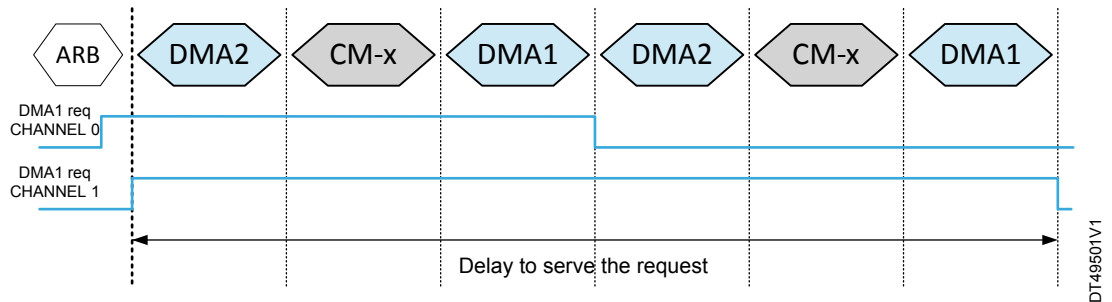
A similar case of parallelism is available in case the product features usable peripherals on separate buses. If the UART interface is available on two different APB bridges, the chances of two parallel transfers competing for system resources diminish.

5.3 Sharing the bus matrix with CPU

Both the DMA and the CPU act as masters on the bus matrix. They may need to access the same resource simultaneously, which is not possible. This is where the bus access arbitration occurs. The bus matrix gives the priority to the Cortex-M CPU versus the DMA controllers, for a CPU shorter latency. However, the round-robin policy guarantees a maximum equal bandwidth to each master port.

The worst case latency is equal among the masters and the priority is only assigned within the round, but does not provide any bandwidth privilege.

Figure 4. Delay to serve the request



The longest possible delay depends on the number of masters accessing the resource as well as the number of channels configured on that particular DMA master. For example, in a simple case of minimum latency per each bus access, the time for which the bus is occupied equals three clock cycles. If the CPU core and two DMA controllers attempt to access the same AHB resource, the arbitration mechanism provides DMA controller access to the AHB after a maximum of six clock cycles. If more channels are configured on the DMA controller, only one can be serviced in a time slot. That yields a latency of 18 clock cycles for the channel 1 of the DMA1 as illustrated in Figure 4.

5.4 Impact of APB

Most of the DMA transfers involve peripherals connected to the APB bus. The APB bus introduces a first additional latency with the AHB to APB bridge. Another APB cause of extra latency is when the APB peripheral is used with a lower APB frequency vs the AHB frequency. The bridge takes its toll to ensure the complete handshake, a precaution necessary to ensure the reliability in buses operating on different frequencies.

This additional latency does not hinder the CPU, unless the CPU is accessing an APB peripheral on the same APB bus. It is however making the AHB bus of the DMA controller port in a busy state preventing other DMA transfers from the same controller. In cases where the CPU simultaneously accesses the same APB bus as the DMA, an additional latency impact of four cycles is added.

6 Insufficient resources threats

The usage of the buses must be carefully considered when designing an application. When the danger of overloading the bus capability is underestimated, several problems may arise:

- **Overflow:** if incoming data is not read from the peripheral register before a next data arrival, the peripheral rises an overflow flag and data may be lost. This is a typical problem with serial interfaces such as UART or SPI. Refer to the peripheral documentation for more details.
- **Data lost without overflow flag:** possible, for example, in case a GPIO port is configured as a parallel communication port.
- **Pauses in transmission:** if T_x data are delayed, the communication interface is stalled for a short time. This is a typical problem of high-speed communication interfaces such as SPI.
- **ADC/DAC sampling timing problem.** This problem is less likely due to typically lower sampling speeds, but not totally impossible.

It is recommended to carefully examine which of the required peripherals are hooked up on which bus, and choose bus frequencies to match the projected bandwidth plus some safety margin.

Caution: The reasonable and recommended safety margin for this occasion is 50%, leaving one-third of the total bus capacity in reserve.

The needed bus bandwidth is to be computed based on the DMA transfer data rate and a fixed 32-bit bus width, independently from the programmed DMA data width of the transfer from the source and to the destination. For example, for a 2 Mbaud 8-bit USART reception a 250 K transfer bandwidth is necessary, because while the internal bus is 32-bit wide, only 8 bits are used at a time.

Note: *The priority scheme of the DMA arbiter always looks for pending request from another channel at data transfer completion. Then it switches to the channel with the pending request of the highest priority. There is no round robin here and transfers with lower priority may never be served if the DMA is kept busy.*

It is still better to assign priorities to minimize latency within the round for selected channels.

Caution: Reserve highest priority for high-speed input, slave mode peripherals, gradually decreasing the priority level down to least priority for low-speed output communication or those in master mode.

7 DMA overview

Table 2 summarizes differences related to the DMA implementation and usage in the products addressed by this document.

Common notable features:

- Circular buffer support
- Programmable transfer size data up to 65535
- Configurable event and interrupt flags for full transfer, half transfer, and error
- Internal data size conversion, allowing 8-bit communication interface data transferred to/from 32-bit memory in packed format.

Table 2. Differences in DMA implementation

Product	F0	F09x	C0	G0	G0Bx	U0	L0	F1	F3	G4	L1	L4	L4+	L5	
DMA1	7(5) ⁽¹⁾	7	3	7(5) ⁽¹⁾	7	7	7(5) ⁽¹⁾	7	7	8	7	7	7	8	
DMA2	-	5	-	-	5	5 ⁽²⁾	-	5	5 ⁽²⁾	8	5	7	7	8	
DMAMUX channels inputs	-	-	4 channels 21 inputs		4 channels 23 inputs		-	-	-	4 channels 21 inputs	-	-	4 channels 26 inputs	4 channels 23 inputs	
Architecture	Von Neumann							Harvard							
Flash memory interface connection	32 bits		64 bits				32 bits	32/64 bits		32 + 32 bits	32/64 bits		32 + 32 bits	64 bits	
CPU core	M0		M0+					M3	M4	M4	M3	M4		M33	
AHB:APB bridge clock	2							3		2	3	2			

1. Only five channels in STM32L0 cat1, STM32G03x, and STM32F0x0.

2. Only some F3 and U0 products feature the DMA2. These are STM32F303xB/C/D/E, STM32F35x, STM32F37x, STM32F39x, STM32U073xx, and STM32U083xx. Others have only DMA1.

8 Conclusion

When correctly used, the DMA controller provides the capability to increase the product efficiency by supporting the application to handle data transfer tasks that would otherwise require a more powerful CPU. This document provides different common DMA features, performances, and guidelines, which are related to the usage of such a DMA controller.

Beyond this document, the user must read the reference manual of the specific product, to understand its DMA specific aspects, especially the system architecture and the memory mapping, the list and mapping of the DMA requests from the peripherals to the DMA channels. Based on this information, the user must accordingly distribute such tasks over the product resources, such as DMA channels, AHB/APB buses, peripherals/memories, and possibly (two) DMA controllers.

Revision history

Table 3. Document revision history

Date	Version	Changes
29-Jun-2007	1	Initial release.
10-Dec-2007	2	<p>Minor text modifications in Section 1.1: Main features.</p> <p>Updated DMA/CPU clock cycle information with bus matrix arbitration and APB bridge data in Section 2.3: DMA latency and Section 2.4: Databus bandwidth limitation.</p> <p>Updated relation between internal data bandwidth and bus type in Section 2.5.2: Internal data bandwidth.</p> <p>Updated Section 3.1: Example of ADC continuous data acquisition with SPI transfer.</p> <p>Changed DMA channel 4 into DMA channel 6, Timer 1 into Timer 3 and 8-bit data into 16-bit data in Section 3.3: GPIO fast data transfer with DMA.</p>
30-Apr-2009	3	<p>Document updated to cover the case where the device has two DMA controllers (Table 2: Peripherals served by DMA2 and channel allocation added, Figure 1: Bus system and peripherals supporting DMA updated).</p> <p>Updated DMA/CPU clock cycle information with latency vs. total service time in Section 2.3: DMA latency and Section 2.4: Databus bandwidth limitation.</p> <p>Small text changes.</p>
28-Jun-2013	4	Added STM32L1 series and related information.
17-Jan-2018	5	<p>Document scope extended to bus bandwidth, lack of bus resources and DMA sequences:</p> <ul style="list-style-type: none"> • Updated: Introduction • Added: Bus bandwidth, DMA controller, DMA latency, Insufficient resources threats, DMA overview table, Conclusion and new figures Figure 1 to Figure 4 • Removed: DMA controller description, Performance considerations, DMA programming examples and old figures
9-May-2019	6	<p>List of supported products extended to G0 and F3:</p> <ul style="list-style-type: none"> • Updated title, Reference documents, General information, Product specific features, DMA controller and Table 1
28-May-2020	7	<ul style="list-style-type: none"> • Updated document title. • Added Table 1: Applicable products. • Updated Table 2: Differences in DMA implementation. • Updated Section 3.1: DMAMUX. • Added Section 3.2: Presence of TrustZone® security features.
12-Dec-2022	8	<p>Added STM32C0 series and related information.</p> <p>Updated Table 2. Differences in DMA implementation.</p>
29-Feb-2024	9	<p>Added STM32U0 series and related information.</p> <p>Updated document title and Table 2. Differences in DMA implementation.</p>

Contents

1	General information	2
2	Bus bandwidth	3
2.1	Bus architecture	3
2.2	Bus matrix	3
2.3	AHB	3
2.4	APB	4
3	Product specific features	5
3.1	DMAMUX	5
3.2	Presence of TrustZone® security features	5
4	DMA controller	6
5	DMA latency	7
5.1	DMA transfer timing	7
5.1.1	AHB peripheral and system volatile memory	7
5.1.2	APB peripheral and system volatile memory	7
5.2	Total service time and parallelism	8
5.3	Sharing the bus matrix with CPU	9
5.4	Impact of APB	9
6	Insufficient resources threats	10
7	DMA overview	11
8	Conclusion	12
	Revision history	13

List of tables

Table 1.	Applicable products	1
Table 2.	Differences in DMA implementation	11
Table 3.	Document revision history	13

List of figures

Figure 1.	DMA block diagram for products with DMA and DMA2	3
Figure 2.	DMA block diagram	6
Figure 3.	Two DMA channels on AHB bus	8
Figure 4.	Delay to serve the request	9

IMPORTANT NOTICE – READ CAREFULLY

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

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

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

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

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

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

© 2024 STMicroelectronics – All rights reserved