AN2548
Application note

Using the STM32F0/F1/Lx Series DMA controller

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.

Reference documents

- STM32F0x reference manuals (RM0091, RM0360)
- STM32F1x reference manuals (RM0008, RM0041)
- STM32L0x reference manuals (RM0367, RM0376, RM0377, RM0451)
- STM32L1x reference manual (RM0038)
- STM32L4x reference manuals (RM0392, RM0393, RM0394, RM0395, RM0411, RM0432)
- Using the STM32F0xx DMA controller (AN4104)

Contents

1 General information 5

2 Bus bandwidth 5

2.1 Bus architecture 5

2.2 Bus matrix 6

2.3 AHB 6

2.4 APB 6

2.5 DMAMUX 6

3 DMA controller 7

4 DMA latency 8

4.1 DMA transfer timing 8

4.1.1 AHB peripheral and system volatile memory 8

4.1.2 APB peripheral and system volatile memory 8

4.2 Total service time and parallelism 9

4.3 Sharing the bus matrix with CPU 10

4.4 Impact of APB 10

5 Insufficient resources threats 11

6 DMA overview table 12

7 Conclusion 13

Revision history 14

List of tables

Table 1. Differences in DMA implementation 12
Table 2. Document revision history 14

List of figures

Figure 1. DMA block diagram for products with 2 DMAs (DMA and DMA2) 5
Figure 2. DMA block diagram 7
Figure 3. Two DMA channels on AHB bus 9
Figure 4. Delay to serve the request 10

1 General information

The STM32F0/F1/Lx Series 32-bit microcontrollers are based on the Arm® Cortex®-M processor.

2 Bus bandwidth

Bus bandwidth is defined as 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 considered product 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 management overhead, and is presenting the bus topology.

2.1 Bus architecture

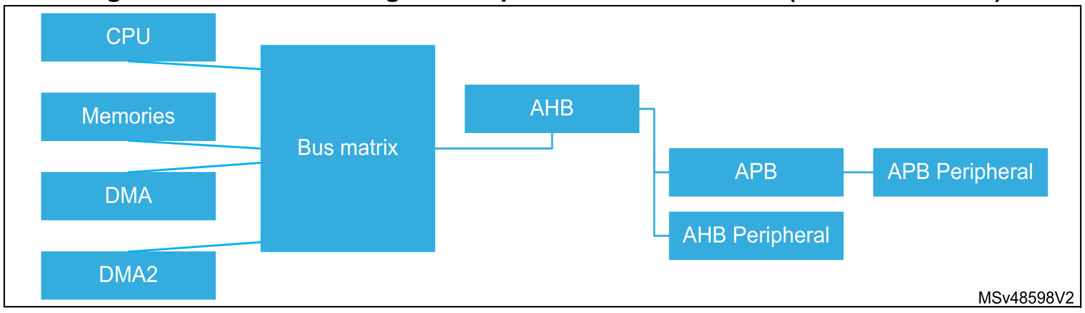
Generally the DMA transfer may span across 3 main bus components: an AHB bus matrix, the AHB bus, and the APB bus(es). A DMA controller is connected to the AHB bus matrix via one dedicated AHB master port, as the CPU and possibly another DMA controller. A peripheral served by the DMA is either connected 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 2 DMAs (DMA and DMA2)



1. DMA2 is only present on more complex products (see *Table 1* for details).

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 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 may have 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 32bit wide connection between the bus matrix and the peripherals. Most peripherals are attached to the APB, only 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 may have one or more 32bit 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. High divider ratio yields lower power consumption, but at 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, reducing also the bandwidth usable on the AHB bus.

2.5 DMAMUX

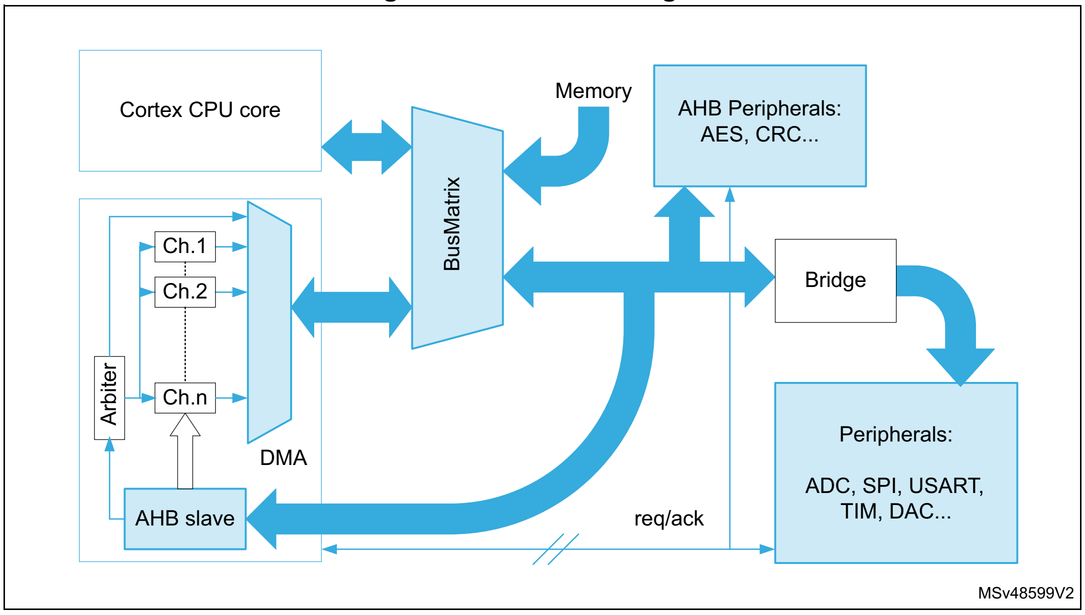
In STM32L4Rx and STM32L4Sx products the DMA capabilities are enhanced by a DMA request multiplexer (DMAMUX). DMAMUX adds a fully configurable routing of any DMA request from a given peripheral in DMA mode to any DMA channel of the DMA controller(s). DMAMUX does not add any clock cycle between the DMA 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 own trigger inputs or by software.

More detailed information about multiplexer can be found in the respective reference manual (RM0432).

3 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 may be assigned to peripherals freely. Typically the channels are preconfigured for different types of block-based data transfers used in the application and then 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



There are 2 distinct types of transfer available:

1. 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.

2. 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. 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.

4 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 channel is enabled.

4.1 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, i.e. if a block of data is reduced to a single one, the last fourth step is not present.

4.1.1 AHB peripheral and system volatile memory

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

1. DMA request arbitration & address computation
2. Reading data from the peripheral on AHB (DMA source, in our case ADC on L4x product)
3. 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 2 AHB clock cycles, if there is no higher priority channel with a pending request and if the slave is ready. Time may be longer when the AHB peripheral and slave is not ready and adds AHB wait state(s).
- t_{RD} is the peripheral read access time
  t_{RD} = minimum 2 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 2 AHB clock cycles in case of SRAM read-after-write access on F1 and L1 series. Bridge wait states may be added.

4.1.2 APB peripheral and system volatile memory

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

1. DMA request arbitration & address computation
2. Reading data from the peripheral on APB (DMA source)
3. Writing loaded data in 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 2 AHB clock cycles, if no higher priority transfer is pending.

- t_{RD} is the peripheral read access time

t_{RD} = 2 APB clock cycles as the base minimum, one extra AHB cycle for bridge synchronization if APB frequency is lower than AHB. See chapter 3.4 for additional details.

- t_{WR} is the SRAM write access time

t_{WR} = same as in AHB case, 1 or 2 AHB cycles.

When the DMA is idle or after the third step has completed on one channel, the DMA arbiter compares the priorities of any pending DMA request (a request may be 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 may lead 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 (i.e. peripheral-to-memory) would give a same latency for the DMA data transfer.

4.2 Total service time and parallelism

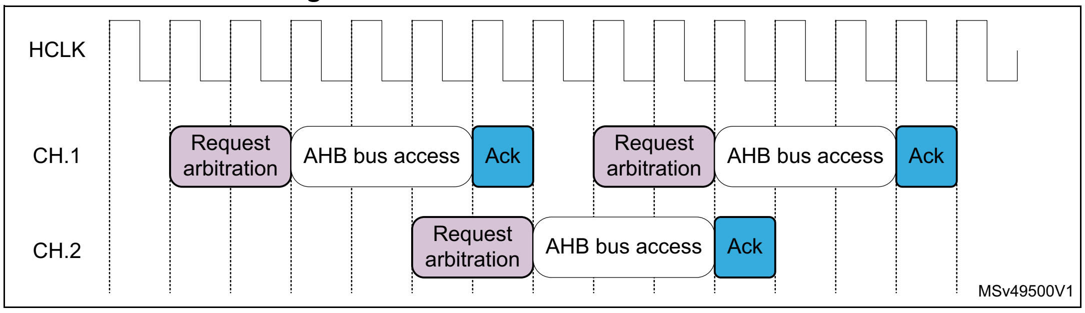
For the case where only one DMA channel is active, a new hardware back-to-back request can not be handled by the DMA before the completion of the previous one, adding one AHB clock cycle for the final idle phase of the DMA request-acknowledge handshake protocol. For this, the total service time, between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



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 to any APB communication interface. No conflict and arbitration occurs.

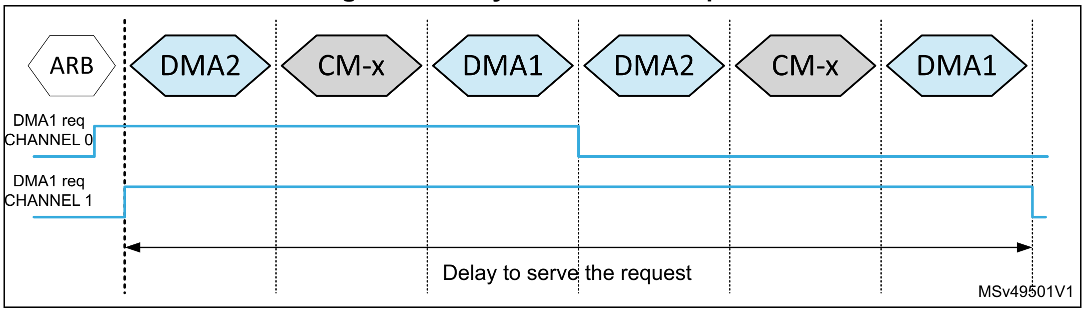
Similar case of parallelism is available in case the product features usable peripherals on separate buses. If UART interface is available on two different APB bridges, the chances of two parallel transfers competing for system resources diminish.

4.3 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 controller(s), 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 simple case of minimum latency per each bus access, the time for which the bus is occupied equals 3 clock cycles. If the CPU core and 2 DMA controllers all attempt to access the same AHB resource, the arbitration mechanism will provide DMA controller access to the AHB after a maximum of 6 clock cycles. If more channels are configured on the DMA controller, only one can be serviced in a time slot. That yields latency of 18 clock cycles for the channel1 of the DMA1 as illustrated in *Figure 4: Delay to serve the request*

4.4 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 guarantee 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 would simultaneously access the same APB bus as the DMA, additional latency impact of 4 cycles is added.

5 Insufficient resources threats

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:

1. Overrun – If incoming data is not read from the peripheral register before next data arrival, the peripheral may rise an overrun 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.
2. Data lost without overrun flag – possible for example in case a GPIO port is configured as a parallel communication port.
3. Pauses in transmission – if T_X data are delayed, the communication interface may be stalled for a short time. This is a typical problem of high speed communication interface such as SPI
4. ADC/DAC sampling timing problem. This problem is less likely due to typically lower sampling speeds, but not totally impossible.

It is advised 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: **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 2Mbaud 8-bit USART reception a 250ktransfers bandwidth is necessary, because while the internal bus is 32bit wide, only 8 bits are used at a time.

Note that the priority scheme of the DMA arbiter will always look for pending request from another channel at data transfer completion. Then it will switch to the channel with 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.**

6 DMA overview table

Table 1 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 1. Differences in DMA implementation

Product / Interfaces	**F0xx**	**F0x0**	**F09x**	**L0(cat1)**	**L0**	**F1**	**L1**	**L4, except L4R and L4S**	**L4Rx and L4Sx**
DMA1	7	5	7	5	7	7	7	7	7
DMA2	-	-	5	-	-	5	5	7	7
DMAMUX	-	-	-	-	-	-	-	-	4 channels, 26 inputs
architecture	Von Neumann					Harvard			
flash interface	32 bits	32 bits	32 bits	32 bits	32 bits	32 / 64 bits	32 / 64 bits	32 + 32 bits	32 + 32 bits
CPU	M0	M0	M0	M0+	M0+	M3	M3	M4	M4
AHB:APB bridge clock	2	2	2	2	2	3	3	2	2

7 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 shall read the reference manual of the specific product, in order 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 should accordingly distribute such tasks over the product resources, resources being DMA channels, AHB/APB buses, peripherals/memories, and possibly (two) DMA controllers.

Revision history

Table 2. Document revision history

Document revision history		
Date	**Revision**	**Changes**
29-June-2007	1	Initial release.
10-Dec-2007	2	Minor text modifications in *Section 1.1: Main features*. 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*. Updated relation between internal data bandwidth and bus type in *Section 2.5.2: Internal data bandwidth*. Updated *Section 3.1: Example of ADC continuous data acquisition with SPI transfer*. 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*.
30-Apr-2009	3	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). 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*. Small text changes.
28-Jun-2013	4	Added STM32L1 Series and related information.
17-Jan-2018	5	Document scope extended to bus bandwidth, lack of bus resources and DMA sequences: – 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