APPLICATION NOTE

Atmel AVR1304: Using the XMEGA DMA Controller

8-bit AVR Microcontrollers

Features

- Four separate DMA channels
- Flexible channel priority selection
- Trhree addressing modes: static, increment, and decrement
- Source and/or destination address reload capabilities
- Single-shot or repeated block transfers
- Double buffering (chained buffers) for continuos transfer
- Interrupts upon buffer empty and errors
- Large block sizes
- Driver source code included

Introduction

The XMEGA® Direct Memory Access Controller (DMAC) is a highly flexible four-channel DMA Controller capable of transferring data between memories and peripherals with minimal CPU intervention. While the CPU spends time in low-power sleep modes or performs other tasks, the XMEGA DMAC offloads the CPU by taking care of mere data copying from one area to another.

Flexible channel priority selection, several addressing modes, double buffering capabilities and large block sizes make the XMEGA DMAC a powerful module for all data oriented applications, such as signal processing and industrial control.

This application note describes the basic functionality of the XMEGA DMAC with code examples to get up and running quickly. A driver interface written in C is included as well.

Figure 1. DMA controller example case.

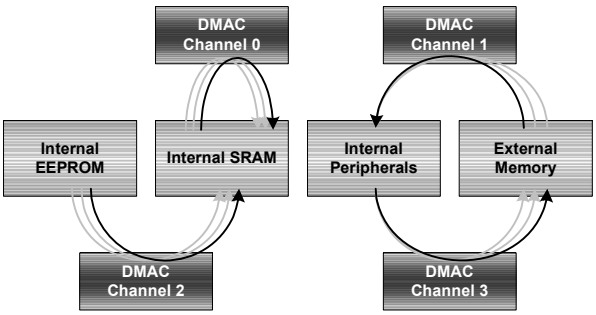


Table of Contents

1. Module overview

This section provides an overview of the basic configuration options and functionality of the DMAC. Chapter 2 then walks you through the basic steps to get up and running, with register descriptions and configuration details.

1.1 DMA channels

In addition to common registers, the DMAC has four independent channels with separate sets of control and status registers. Each channel holds the necessary control and state information for one DMA transaction, with source and destination addresses, byte count and status. When a transaction is finished, the channel can either be left unused, reconfigured for a different transaction, or triggered to repeat the previous transaction.

DMA transactions are covered in more detail in Section 1.4.

1.2 Data transfer

The concept of a *Data Transfer* in this context refers to the operation where the DMAC channel copies one, two, four, or eight bytes from the source to the destination address in one *Burst Transfer*.

When the channel is configured to transfer two, four, or eight bytes at a time, the controller holds the bus for the duration of two, four, or eight byte transfers, respectively, once the DMAC acquires the bus.

Multi-byte bursts are useful for transferring multi-byte register data from e.g. an ADC without risking that the CPU or other DMA channels access multi-byte registers, which could cause the data in the multi-byte temporary registers to be corrupted.

The transfer mode is configured with the burst length setting in the *Channel Transfer Mode* bitfield (`BURSTLEN`) in each channel's *Control* register (`CTRL`).

Figure 1-1 in Section 1.4 illustrates data transfers and burst mode.

1.3 Block transfer and repeat mode

The concept of a *Block Transfer* in this context refers to the operation of performing all data transfers necessary to transfer the number of bytes given by the *block* size. The block size is configured with the channel's 16-bit *Block Transfer Count* register (`TRFCNT`), which allows for block sizes up to 64K bytes. A value of zero means 64K bytes.

Block size do not have to be a multiple of the burst mode byte count, the data transfer stops when all bytes in the block are transferred. For instance, using four-byte burst mode with a block size of ten bytes results in two full data transfers and one final two-byte data transfer.

To extend the number of bytes transferred beyond 64K, a *repeat counter* can be used to perform the block transfer a number of times. The total amount of bytes transferred equals the block size times the repeat count. The repeat mode is enabled with the *Repeat Mode* bit (`REPEAT`) in the channel's *Control* register (`CTRL`). The repeat count itself is located in the channel's *Repeat Count* register (`REPCNT`). With an 8-bit repeat counter, the DMAC is capable of transferring 16M bytes without any CPU intervention.

Note that unlimited repeat count can be achieved by enabling repeat mode and setting the repeat count to zero.

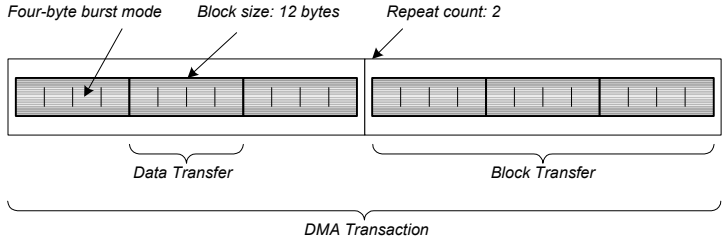
Figure 1-1 in Section 1.4 illustrates block transfers in relation to data transfers and DMA transactions.

1.4 DMA transaction

The concept of a *DMA Transaction* in this context refers to the whole operation with all data transfers and repeated block transfers. A DMA transaction starts with the first DMA request after preparing the channel and ends when all block transfers are finished and the repeat counter is zero.

Figure 1-1 shows the relation between data transfers, block transfers, and DMA transactions.

Figure 1-1. Data transfer, block transfer, and DMA transactions.



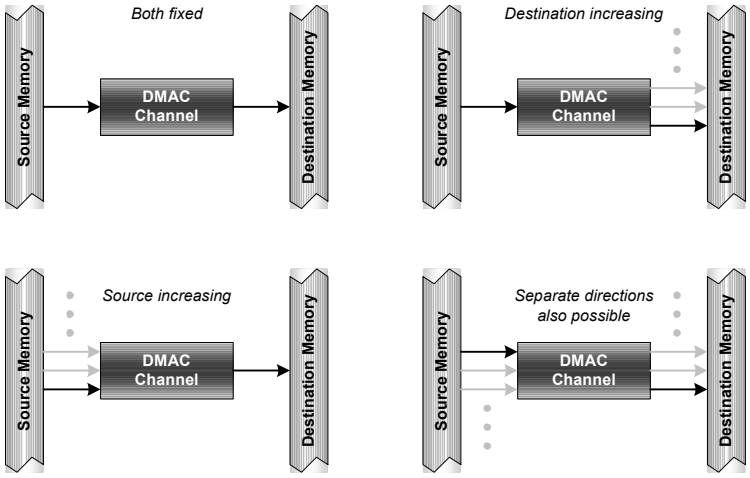
1.5 Addressing modes

By default the source and destination addresses are fixed during all transfers, which means each data transfer for the channel copies data to and from the same memory locations. This is useful for e.g. piping data from one peripheral to another. The burst length decides the number of bytes to copy during each data transfer, but the source and destination address are still fixed for each access. For instance, four-byte burst mode would copy the same byte four times during each data transfer.

For applications where data is to be stored in an array, the DMAC can be configured to increase or decrease the *destination* address after each byte access. Similarly, when data is to be *retrieved* from an array, the DMAC can be configured to increase or decrease the *source* address after each byte access. Finally, for copying data from one array to another, both source and destination address can be increased or decreased after each byte access.

Addressing modes are configured with the *Source* and *Destination Address Mode* bitfields (`SRCDIR` and `DESTDIR`) in each channel's *Address Control* register (`ADDRCTRL`).

Figure 1-2. Addressing mode examples.



1.6 Address reload

When address increment or decrement is used, the default behavior is to update the address after each access. However, many applications would reset the source or destination pointers to the start of the array at some point. During a DMA transaction the original source and destination address are stored by the DMAC, and the addresses can be individually configured to reload at the following points in time:

- End of each data transfer
- End of each block transfer
- End of transaction
- Never reload

For instance, reload after each data transfer can be used in conjunction with eight-byte burst mode to repeatedly read all four results of an XMEGA ADC and write the data into an array. The eight-byte burst mode makes sure all four ADC results are read without CPU access in between and the source address reload positions the source pointer back to the first ADC result register after the data transfer.

Note that when using reload after a burst and the total byte count is not a multiple of the burst size; the last burst transfer will stop when the total byte count is reached, and the reload action occurs. For example, using 4-byte bursts with a 17-byte block, the reload will occur after the 4^{th}, 8^{th}, 16^{th} and 17^{th} byte.

Address reload is configured with the *Source* and *Destination Address Reload* bitfields (`SRCRELOAD` and `DESTRELOAD`) in each channel's *Address Control* register (`ADDRCTRL`).

1.7 Transfer triggers and single-shot mode

The available trigger sources vary from device to device, and the user should consult the datasheet of each device for a complete list of DMA transfer trigger sources.

In addition to the external transfer triggers, it is possible to manually trigger a transfer for a channel by setting the *Transfer Request* bit (`TRFREQ`) in the channel's *Control* register (`CTRL`).

By default, a transfer trigger starts a Block Transfer operation. The transaction continues until one block is transferred. When the block is transferred, the channel waits for the next trigger to arrive.

For some applications it might not be desired to finish one complete block transfer for every trigger. By setting the *Single Shot* bit (`SINGLE`) in a channel's *Control* register (`CTRL`), each trigger will complete one data transfer instead of a block transfer. The rest of a channel's settings and operating modes are not affected by enabling the single-shot mode.

Single-shot mode is often used in conjunction with accessing multi-byte peripheral registers where addresses are reloaded after each data transfer. See the example in Section 2.3.

1.8 Interrupt handling

Each DMA channel can be configured to request interrupts after the following events:

- Transfer Complete
- Transfer Error

By default, the *Transfer Complete* interrupt is requested after each block transfer. When repeat mode is enabled, the interrupt is not requested until all block transfers are completed and the repeat counter has reached zero. For the special case of unlimited repeating, an interrupt is requested after every block transfer.

The *Transfer Error* interrupt is requested if a transfer operation is aborted for some reason. This happens if for instance a channel or the whole DMAC module is disabled while transactions are still in progress. The transfer is also aborted by write attempts to EEPROM space and also by read attempts to EEPROM space if the CPU is in any sleep mode.

Note: The two interrupts share one common interrupt vector (one vector per DMA channel). Therefore, the interrupt handler must check the flags, take appropriate action and clear the flags manually.

1.9 Accessing multi-byte DMAC registers

Several registers in the DMAC module are two- or three-byte registers. To ensure that all bytes of a multi-byte register are read and written simultaneously, always read and write the lowest byte register first.

During read operations, the complete register is latched when the first byte is read and the next one or two bytes are stored in temporary locations until they are read.

During write operations, the first one or two bytes are stored in temporary locations until the final byte is written, which triggers a write operation for the whole register.

It is important to note that the temporary storage registers are shared among *all* the multi-byte registers in the DMAC module. Therefore it is imperative to complete a read or write operation for such a register before accessing other multi-byte registers, or else the data will be corrupted. For multithreaded applications or applications that access the DMAC from interrupt handlers, the user must implement software mechanisms to ensure mutually exclusive access to the multi-byte registers of the DMAC module.

The registers in question are:

- Block Transfer Count
- Source Address
- Destination Address

2. Getting started

This section walks you through the basic steps for getting up and running with simple transfers and experimenting with different operating modes. The necessary registers are described along with relevant bit settings.

2.1 Memory block copy

Task: Copy 1K bytes from source to destination array using DMA channel 0 in eight-byte burst mode and manual transfer trigger.

- Set the Enable bit (`ENABLE`) in the *DMA Control* register (`CTRL`) to enable the DMAC with default settings for priority
- Set the *Transfer Mode* bitfield (`BURSTLEN`) in *Channel 0 Control Register* (`CTRL`) equal to 0x03 to select eight-byte burst mode
- Set the Source Address Mode bitfield (`SRCDIR`) in *Channel 0 Address Control* register (`ADDRCTRL`) equal to 0x01 to make the source address increment after each byte access
- Set the Destination Address Mode bitfield (`DESTDIR`) in *Channel 0 Address Control* register (`ADDRCTRL`) equal to 0x01 to make the destination address increment after each byte access
- Set the *Channel 0* Trigger Source register (`TRIGSRC`) equal to 0x00 to select manual trigger source
- Set the *Channel 0* Block Transfer Count register (`TRFCNT`) equal to 0x0400 to set the block size to 1K bytes
- Set the *Channel 0* Source Address register (`SRCADDR`) to point to the start of the source array
- Set the *Channel 0* Destination Address register (`DESTADDR`) to point to the start of the destination array
- Set the *Enable* bit (`ENABLE`) in *Channel 0 Control* register (`CTRL`) to enable DMA channel 0
- Set the *Transfer Request* bit (`TRFREQ`) in *Channel 0 Control* register (`CTRL`) to start the transaction

- Wait for the *Transaction Complete Interrupt Flag* bit for channel 0 (CH0TRNIF) in the *Transfer Interrupt Status* register (INTFLAGS) to be set, indicating that the transaction is finished. Clear the flag afterwards by writing logic one to it

2.2 Copy single-byte data from SRAM array to peripheral

Task: Copy 2K bytes from an array to a peripheral I/O register using DMA channel 2 in cycle stealing mode and manual transfer trigger for every byte.

- Set the *Enable* bit (ENABLE) in the *DMA Control* register (CTRL) to enable the DMAC with default settings for priority
- Set the *Single Shot* bit (SINGLE) in *Channel 2 Control* register (CTRL) to select single-shot mode
- Set the *Transfer Mode* bitfield (BURSTLEN) in *Channel 2 Control* register (CTRL) equal to 0x00 to select cycle stealing mode
- Set the *Source Address Mode* bitfield (SRCDIR) in *Channel 2 Address Control* register (ADDRCTRL) equal to 0x01 to make the source address increment after each byte access
- Set the *Destination Address Mode* bitfield (DESTDIR) in *Channel 2 Address Control* register (ADDRCTRL) equal to 0x00 to leave the destination address between byte accesses
- Set the *Channel 2 Trigger Source* register (TRIGSRC) equal to 0x00 to select manual trigger source
- Set the *Channel 2 Block Transfer Count* register (TRFCNT) equal to 0x0800 to set the block size to 2K bytes
- Set the *Channel 2 Source Address* register (SRCADDR) to point to the start of the source array
- Set the *Channel 2 Destination Address* register (DESTADDR) to point to I/O register of the peripheral
- Set the *Enable* bit (ENABLE) in *Channel 2 Control* register (CTRL) to enable DMA channel 2
- Set the *Transfer Request* bit (TRFREQ) in *Channel 2 Control* register (CTRL) to transfer one byte from the array to the peripheral. Repeat until the *Transaction Complete Interrupt Flag* bit for channel 2 (CH2TRNIF) in the *Transfer Interrupt Status* register (INTFLAGS) is set, indicating that the transaction is finished. Clear the flag afterwards by writing logic one to it

2.3 Copy four-byte results from peripheral to SRAM array

Task: Copy 4K bytes from a peripheral 4-byte result register to an array using DMA channel 1 in four-byte burst mode and transfer triggered by the peripheral.

- Set the *Enable* bit (ENABLE) in the *DMA Control* register (CTRL) to enable the DMAC with default settings for priority
- Set the *Single Shot* bit (SINGLE) in *Channel 1 Control* register (CTRL) to select single-shot mode
- Set the *Transfer Mode* bitfield (BURSTLEN) in *Channel 1 Control* register (CTRL) equal to 0x02 to select four-byte burst mode
- Set the *Source Address Reload* bitfield (SRCRELOAD) in *Channel 1 Address Control* register (ADDRCTRL) equal to 0x02 to make the source address reload after each four-byte data transfer
- Set the *Source Address Mode* bitfield (SRCDIR) in *Channel 1 Address Control* register (ADDRCTRL) equal to 0x01 to make the source address increment after each byte access
- Set the *Destination Address Mode* bitfield (DESTDIR) in *Channel 1 Address Control* register (ADDRCTRL) equal to 0x01 to make the destination address increment after each byte access
- Set the *Channel 1 Trigger Source* register (TRIGSRC) to a suitable trigger source from the peripheral
- Set the *Channel 1 Block Transfer Count* register (TRFCNT) equal to 0x1000 to set the block size to 4K bytes
- Set the *Channel 1 Source Address* register (SRCADDR) to point to the first result register of the peripheral
- Set the *Channel 1 Destination Address* register (DESTADDR) to point to the start of the destination array

- Set the *Enable* bit (ENABLE) in *Channel 1 Control* register (CTRL) to enable DMA channel 1
- Wait for the *Transaction Complete Interrupt Flag* bit for channel 1 (CH1TRNIF) in the *Transfer Interrupt Status* register (INTFLAGS) to be set, indicating that the transaction is finished. Clear the flag afterwards by writing logic one to it

3. Advanced features

This section introduces more advanced features and possibilities with the DMAC.

3.1 Double buffering

To allow for continuous transfer, two channels can be interlinked so that the second takes over the transfer when the first is finished and vice versa. This leaves time for the application to process the data transferred by the first channel, prepare fresh data buffers and set up the channel registers again while the second channel is working. This is referred to as *Double Buffering or Chained Transfers*.

When double buffering is enabled for a channel pair, it is important that the two channels are configured with the same repeat count. The block sizes need not be equal, but for most applications they should be, along with the rest of the channel's operation mode settings.

Note that the double buffering channel pairs are limited to channel 0 and 1 as the first pair and channel 2 and 3 as the second pair. However, it is possible to have one pair operate in double buffered mode while the other is left unused or operating independently.

3.2 Channel priority

If more than one channel has pending transactions when the DMAC acquires the data bus, some kind of priority scheme must be used to ensure correct operation.

The XMEGA DMAC provides the following priority schemes:

- Round robin scheduling for all channels
- Channel 0 highest, with round robin for the rest
- Channel 0 highest, then channel 1, and finally round robin for the rest
- All channels prioritized with channel 0 having the highest priority

For channels with round robin scheduling, the channels take turns with their data transfers every time the DMAC acquires the data bus. Channels with fixed priority always get to complete a pending data transfer before the channels with round robin scheduling. However, no data transfer can interrupt another once it has started. Note that only one data transfer for any channel is allowed each time the DMAC acquires the bus, and not one data transfer for each channel.

For timing sensitive applications that use the DMAC heavily, it is important to plan and schedule the channel assignments and priority scheme carefully to achieve sufficient timing accuracy.

4. Driver implementation

This application note includes a source code package with a basic DMAC driver implemented in C. It is written for the IAR Embedded Workbench® compiler.

Note that this DMAC driver is not intended for use with high-performance code. It is designed as a library to get started with the DMAC. For timing and code space critical application development, you should access the DMAC registers directly. Please refer to the driver source code and device datasheet for more details.

4.1 Files

The source code package consists of three files:

- *dma_driver.c* – DMAC driver source file
- *dma_driver.h* – DMAC driver header file
- *main.c* – Example code using the driver

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation.

4.2 Doxygen documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the readme.html file in the source code folder.

5. Revision History

Doc. Rev.	Date	Comments
8046D	05/2013	The document title is corrected and the Revision History is added
8046C	10/2012	New template
8046B	07/2009	Several bugs fixed
8046A	02/2008	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370

© 2013 Atmel Corporation. All rights reserved. / Rev.: 8046D–AVR–05/2013

Atmel®, Atmel logo and combinations thereof, AVR®, Enabling Unlimited Possibilities®, XMEGA®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.