AVR1315: Accessing the XMEGA EEPROM

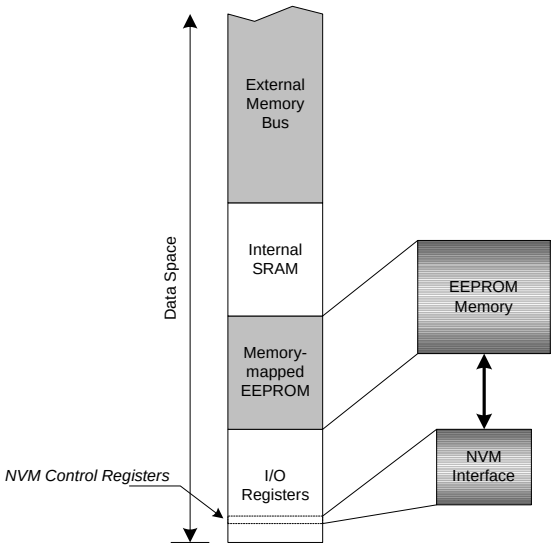
Features

- **I/O-mapped access**
- **Memory-mapped access**
- **Split erase and write operations supported**
- **Efficient page-oriented access**
- **Driver source code included**

1 Introduction

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

Figure 1-1. EEPROM Overview



8-bit AVR® Microcontrollers

Application Note

Rev. 8066A-AVR-04/08

2 EEPROM Overview

This section provides an overview of the functionality and basic configuration options of the EEPROM. Section 3 then walks you through the basic steps to get you up and running, with register descriptions and configuration details.

2.1 The Non-volatile Memory Controller

The XMEGA EEPROM is accessed through the Non-volatile Memory (NVM) controller, which is also used for access to fuse and lock bits and for updating Flash memory from software. Please refer to the application note “AVR1316: XMEGA Bootloader” or the device datasheet for more details.

As the NVM controller not only serves the EEPROM memory, it is important to check that the NVM controller is not busy with other operations (such as Flash memory update) before using it to access EEPROM. The *NVM Busy* bit (NVMBUSY) in the *NVM Status* register (STATUS) is set whenever the NVM controller is busy.

Using the NVM to access EEPROM involves using *NVM Commands*. There are commands for erasing, writing, reading etc. The procedure for issuing a command is as follows:

1. Wait for any previous NVM operations to finish.
2. Load necessary information into the *Address* registers (ADDRn) and/or the *Data* registers (DATAn). This will be described in later sections.
3. Load the command code into the *Command* register (CMD).
4. Load the *Protect IO Register* signature (byte value 0xD8) into the *Configuration Change Protection* register (CCP). This will automatically disable all interrupts for the next four CPU instruction cycles.
5. Within the next four CPU instruction cycles, set the *Command Execute* bit (CMDEX) in *NVM Control Register A* (CTRLA).
6. The operation is finished when the *NVM Busy* bit (NVMBUSY) is cleared.

The relevant command codes and their use are described in the following sections.

By setting the *EEPROM Mapping Enable* bit (EEMAPEN) in *NVM Control Register B* (CTRLB), EEPROM access is mapped to data memory space instead of using the NVM controller. However, one must still check that the NVM is not busy before accessing EEPROM, as the NVM is used internally. Memory-mapped access is covered in Section 3.2.

2.2 Erasing and Writing

There are two ways to update the EEPROM memory: *atomic write* and *split operation*. With atomic write, EEPROM locations are erased and written in one operation. With split operation, erasing and writing are separate operations.

When erasing the EEPROM locations, all bits are set to logic one. A split write can then program selected bits to logic zero. Split write operations cannot change a bit from zero to one, as opposed to an atomic write that first erase to logic one and then writes logic zeros to selected bits. This means that multiple writes with different values eventually results in all locations being logic zeros, if the locations are not

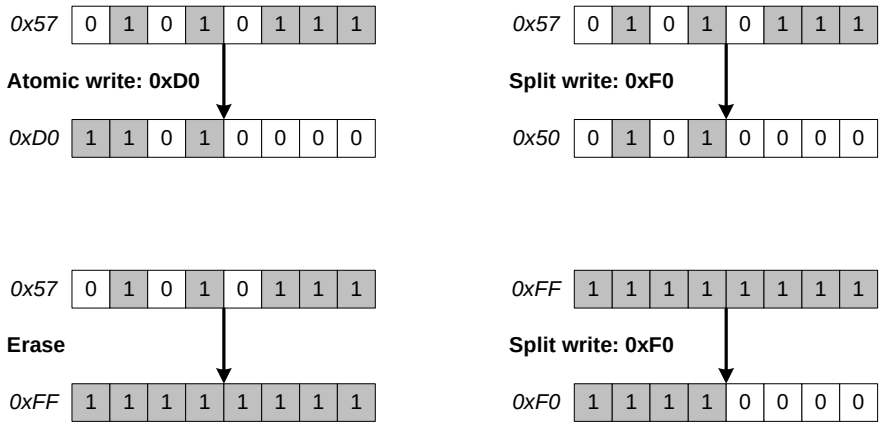
erased in between. This is similar to a logic AND operation between existing value and written value.

An atomic write takes approximately twice the time of one erase or one write. Therefore, split operation can be used to save time by erasing EEPROM locations in advance, e.g. during initialization. For instance, if the application needs to store vital data when a power drop is detected, a split write will take less time than an atomic write.

Another useful application for the split operation is to increase EEPROM endurance, especially for applications that update EEPROM locations frequently. By implementing a scheme where EEPROM locations are not erased unless they have to, the split operation feature can be used to increase EEPROM endurance. For more details, refer to the application note “AVR101: High Endurance EEPROM Storage”.

Different erase and write operations, together with byte values before and after the operation, are illustrated in Figure 2-1 below.

Figure 2-1. Atomic Write, Split Write and Erase Operations



2.3 The Temporary Page Buffer

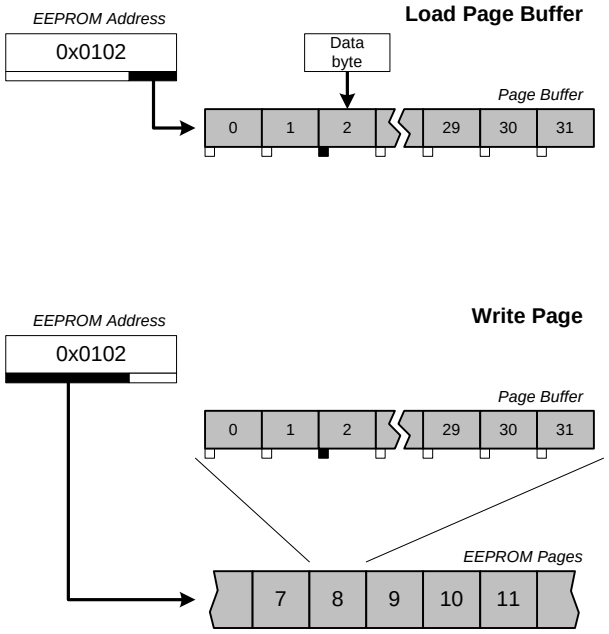
The EEPROM memory is organized in pages, and both erase and write operations operate on pages. The page size depends on memory size and is given in the device datasheet. Erase and write operations use a temporary page buffer, both to keep track of which bytes in a page should be accessed and, for write operations the data itself.

A write operation actually consists of two operations: (1) loading the page buffer with data and (2) writing data to an EEPROM page. When loading the page buffer, the lower part of the EEPROM address is used to select a byte in the page buffer, while the upper part is ignored. When writing or erasing a page, the upper part of the address selects the page, while the lower part is ignored.

Figure 2-2 below shows an example where one byte is loaded into the page buffer and the buffer is written to an EEPROM page afterwards. The figure shows that the buffer location that was loaded gets tagged so that the page buffer knows which byte locations need to be written.

For a page erase operation, only the bytes that are tagged in the buffer will be erased in EEPROM. Unless you plan to perform a write afterwards, the actual values in the buffer do not matter.

Figure 2-2. Example Buffer Load and Page Write with a 32-byte Buffer



Once a byte has been loaded into the buffer, the *EEPROM Page Buffer Active Loading* bit (EELOAD) in the *NVM Status* register (STATUS) will be set. The bit remains set until either the buffer is flushed or a page write is performed (atomic or split write). To flush the page buffer, issue the *EEPROM Flush Page Buffer* command (byte value 0x36) to the NVM controller. Erase and write commands are covered in Section 3.1.

Note that if multiple bytes are loaded to the same buffer location, the resulting value is a logical AND between existing data and the new data, similar to what happens during page write operations.

2.4 DMA Considerations

When using DMA and EEPROM in the same application, it should be considered that the DMA controller cannot access the EEPROM when the CPU is in any sleep mode. An ongoing DMA transaction will be aborted if an access is attempted during sleep. Please refer to the application note “AVR1304: Using the XMEGA DMA Controller” for more information.

2.5 Interrupt Considerations

When using interrupts and EEPROM in the same application, the following should be considered:

- Take care not to corrupt the EEPROM page buffer contents. When loading the page buffer, make sure that no interrupt handlers access the page buffer. Or, if an interrupt handler is used to access the page buffer, make sure that other parts of the application do not access it at the same time.
- Instead of continuously polling the *NVM Busy* flag (NVMBUSY) to detect when an NVM operation is finished, it is possible to enable the EEPROM Interrupt by setting an appropriate interrupt level with the *EEPROM Interrupt Level* bitfield

(EELVL) in the *Interrupt Control* register (INTCTRL). The corresponding interrupt handler will be called whenever the *NVM Busy* flag (NVMBUSY) is not set. For instance, this can be used to implement an interrupt-controlled EEPROM update. More details on interrupts can be found in application note “AVR1305: XMEGA Interrupts and the Programmable Multi-level Interrupt Controller”.

3 Getting Started

This section walks you through the basic steps for getting up and running with the XMEGA EEPROM. The necessary registers are described along with relevant bit settings.

3.1 I/O-mapped Access

I/O-mapped access means that all EEPROM access is done using I/O registers in the NVM controller and issuing NVM commands.

3.1.1 Reading EEPROM

The steps to read an EEPROM location are as follows:

1. Wait for any previous NVM operations to finish.
2. Load the *NVM Address* registers (ADDRn) with the desired EEPROM byte address.
3. Issue the *EEPROM Read* command (byte value 0x06) to the NVM controller.
4. The CPU will be halted two clock cycles and then data is available in *NVM Data Register 0* (DATA0). There is no need to wait for the *NVM Busy* bit to clear.

This operation is not supported if memory-mapped EEPROM access is enabled.

3.1.2 Loading EEPROM Page Buffer

The steps to load a byte into the temporary page buffer are as follows:

1. Wait for any previous NVM operations to finish.
2. Load the *EEPROM Load Page Buffer* command (byte value 0x33) into the *NVM Command* register (CMD).
3. Load the *NVM Address registers* (ADDRn) with the desired EEPROM byte address.
4. Load *NVM Data Register 0* (DATA0) with the data byte. This automatically triggers the buffer load operation. This is a command that does not require the *Command Execute* bit (CMDEX) in *NVM Control Register A* (CTRLA) to be set. The CPU will be halted for one clock cycle during this operation.

It is important that the write to the data register is done last, as this triggers the operation itself. The order of the other steps is not important.

This operation is not supported if memory-mapped EEPROM access is enabled.

3.1.3 Atomic Write (Erase and Write) EEPROM Page

The steps to erase a page and write prepared page buffer data to the page is as follows:

1. Wait for any previous NVM operations to finish.
2. Load the *NVM Address registers* (ADDRn) with an EEPROM address within the page to be updated.
3. Issue the *EEPROM Atomic Write* command (byte value 0x35) to the NVM controller.
4. The operation is finished when the *NVM Busy* bit is cleared.

Only the buffer locations that have been loaded will be updated in the EEPROM page.

3.1.4 Erase EEPROM Page

The steps to erase a page without writing anything are as follows:

1. Wait for any previous NVM operations to finish.
2. Load the *NVM Address registers* (ADDRn) with an EEPROM address within the page to be erased.
3. Issue the *EEPROM Erase Page* command (byte value 0x32) to the NVM controller.
4. The operation is finished when the *NVM Busy* bit is cleared.

Only the buffer locations that have been loaded will be erased in the EEPROM page. Therefore, dummy bytes should be loaded into every buffer location corresponding to the bytes you want to erase from the EEPROM page.

3.1.5 Split Write (Write Only) EEPROM Page

The steps to write prepared page buffer data to an already erased page are as follows:

1. Wait for any previous NVM operations to finish.
2. Load the *NVM Address registers* (ADDRn) with an EEPROM address within the page to be updated.
3. Issue the *EEPROM Split Write* command (byte value 0x34) to the NVM controller.
4. The operation is finished when the *NVM Busy* bit is cleared.

Only the buffer locations that have been loaded will be updated in the EEPROM page.

3.2 Memory-mapped Access

Memory-mapped access means that EEPROM read and page buffer load operations are mapped into data space. This means that EEPROM data can be read simply by reading from a location in data memory. For the XMEGA A1 family, memory-mapped EEPROM starts at address 0x1000.

Page buffer loading is also simply a matter of writing to data memory. However, flushing the buffer and erasing and writing pages must still be done through the NVM controller as for I/O-mapped access. Also, the NVM controller must not be busy when accessing EEPROM.

The necessary steps to perform an atomic write using memory-mapped access are as follows:

1. Wait for any pervious NVM operations to finish.
2. Load page buffer by writing directly to data space, while staying inside one EEPROM page.
3. Load the *NVM Address registers* (ADDRn) with an EEPROM address within the page to be updated.
4. Issue the *EEPROM Atomic Write* command (byte value 0x35) to the NVM controller.
5. The operation is finished when the *NVM Busy* bit is cleared.

The procedure is similar for erase and split write operations. In short, reading data and loading the page buffer is replaced by memory-mapped access. The rest is similar to I/O-mapped access. Please refer to the code examples of this application note for more details.

4 Driver Implementation

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

Note that this EEPROM driver is not intended for use with high-performance code. It is designed as a library to get started with the EEPROM. For timing and code space critical application development, you should access the EEPROM (NVM) 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:

- *eeprom_driver.c* – EEPROM driver source file
- *eeprom_driver.h* – EEPROM driver header file
- *eeprom_example.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.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature