
AT07336: Common Non-Volatile Memory (NVM) driver

ASF PROGRAMMERS MANUAL

Common Non-Volatile Memory (NVM) driver

This driver provides an interface for the configuration and management of Non-Volatile Memories within the device. It can be used for the partitioning, erasing, reading, and writing of data.

The following peripherals are used by this module:

- **NVM (Non-Volatile Memory)**

The outline of this documentation is as follows:

- [Prerequisites](#)
- [Module Overview](#)
- [Special Considerations](#)
- [Extra Information](#)
- [Examples](#)
- [API Overview](#)

Table of Contents

Common Non-Volatile Memory (NVM) driver	1
Software License	3
1. Prerequisites	4
2. Module Overview	5
3. Special Considerations	6
3.1. Page Erasure	6
3.2. Clocks	6
3.3. Security Bit	6
4. Extra Information	7
5. Examples	8
6. API Overview	9
6.1. Function Definitions	9
6.1.1. Function nvm_get_page_size()	9
6.1.2. Function nvm_get_pagenum()	9
6.1.3. Function nvm_get_size()	10
6.1.4. Function nvm_init()	10
6.1.5. Function nvm_page_erase()	11
6.1.6. Function nvm_read()	11
6.1.7. Function nvm_read_char()	12
6.1.8. Function nvm_set_security_bit()	12
6.1.9. Function nvm_write()	12
6.1.10. Function nvm_write_char()	13
6.2. Enumeration Definitions	14
6.2.1. Enum mem_type_t	14
7. Extra Information for Non-Volatile Memory Driver	15
7.1. Acronyms	15
7.2. Dependencies	15
7.3. Errata	15
7.4. Module History	15
8. Examples for Non-Volatile Memory Driver	16
8.1. Quick Start Guide for Common NVM driver	16
8.1.1. Basic Use Case	16
8.1.2. Setup Steps	16
8.1.3. Usage Steps	16
Index	20
Document Revision History	21

Software License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.
4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

The Non-Volatile Memory (NVM) module provides an interface to the device's Non-Volatile Memory controller, so that memory pages can be written, read, erased and reconfigured in a standardized manner.

The device specific flash driver can be used to program fuses, read the device's unique ID (if supported) or lock/unlock regions of memory.

Typically the NVM driver is used to store static parameters e.g. external device calibration data, factory calibration data, user application specific data, unique IDs etc.

3. Special Considerations

3.1 Page Erasure

The granularity of an erase is for a certain number of pages (dependent on the specific device family), while the granularity of a write is per page. Thus, if the user application is modifying only part of a page, the original data must be read, updated, and then the page(s) erased and programmed.

3.2 Clocks

The user must ensure that the driver is configured with the correct number of wait states when the CPU is running at high frequencies.

3.3 Security Bit

When the security bit is set external access to the flash using JTAG/SWD (or Fast Programming) is forbidden. This ensures the confidentiality of the code/data programmed into the Flash.

The security bit can only be cleared by performing a chip erase.

4. Extra Information

For extra information, see [Extra Information for Non-Volatile Memory Driver](#). This includes:

- [Acronyms](#)
- [Dependencies](#)
- [Errata](#)
- [Module History](#)

5. Examples

For a list of examples related to this driver, see [Examples for Non-Volatile Memory Driver](#).

6. API Overview

6.1 Function Definitions

6.1.1 Function `nvm_get_page_size()`

Get the page size (in bytes) for the Non-Volatile Memory specified.

```
status_code_t nvm_get_page_size(  
    mem_type_t mem,  
    uint32_t * size)
```

Table 6-1. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to get the page size for
[out]	size	Pointer to where to store the page size

Returns The memory operation error code.

Table 6-2. Return Values

Return value	Description
STATUS_OK	Page size operation successful
ERR_INVALID_ARG	Invalid memory type specified

6.1.2 Function `nvm_get_pagenumber()`

Get the page number from the byte address in the Non-Volatile Memory specified.

```
status_code_t nvm_get_pagenumber(  
    mem_type_t mem,  
    uint32_t address,  
    uint32_t * num)
```

Table 6-3. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory
[in]	address	Byte address of the non volatile memory
[out]	num	Pointer to where to store the page number

Returns The memory operation error code.

Table 6-4. Return Values

Return value	Description
STATUS_OK	Page number operation successful

Return value	Description
ERR_INVALID_ARG	Invalid memory type specified

6.1.3 Function `nvm_get_size()`

Get the size (in bytes) for the whole Non-Volatile Memory specified.

```
status_code_t nvm_get_size(
    mem_type_t mem,
    uint32_t * size)
```

Table 6-5. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to get the size for
[out]	size	Pointer to where to store the size

Returns

The memory operation error code.

Table 6-6. Return Values

Return value	Description
STATUS_OK	Size operation successful
ERR_INVALID_ARG	Invalid memory type specified

6.1.4 Function `nvm_init()`

Initialize the Non-Volatile Memory specified.

```
status_code_t nvm_init(
    mem_type_t mem)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to initialize

Returns

The memory operation error code.

Table 6-8. Return Values

Return value	Description
STATUS_OK	Initialisation successful
ERR_INVALID_ARG	Unknown memory type
ERR_NO_MEMORY	Memory check failed

6.1.5 Function `nvm_page_erase()`

Erase a page in the non-volatile memory.

```
status_code_t nvm_page_erase(  
    mem_type_t mem,  
    uint32_t page_number)
```

Table 6-9. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to erase
[in]	page_number	Page number to erase

Returns

The memory operation error code.

Table 6-10. Return Values

Return value	Description
STATUS_OK	Erase operation successful
ERR_INVALID_ARG	Erase failed due to an invalid parameter

6.1.6 Function `nvm_read()`

Read a number of bytes from the source address in the Non-Volatile Memory and store them in a destination buffer.

```
status_code_t nvm_read(  
    mem_type_t mem,  
    uint32_t address,  
    void * buffer,  
    uint32_t len)
```

Table 6-11. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to read
[in]	address	Address to read
[out]	buffer	Pointer to destination buffer
[in]	len	Number of bytes to read

Returns

The memory operation error code.

Table 6-12. Return Values

Return value	Description
STATUS_OK	Read operation successful
ERR_BAD_ADDRESS	Invalid memory address

Return value	Description
ERR_INVALID_ARG	Read failed due to an invalid parameter

6.1.7 Function `nvm_read_char()`

Read a single byte of data from an address in the Non-Volatile Memory.

```
status_code_t nvm_read_char(
    mem_type_t mem,
    uint32_t address,
    uint8_t * data)
```

Table 6-13. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to read
[in]	address	Address to read
[out]	data	Pointer to where to store the read data

Returns The memory operation error code.

Table 6-14. Return Values

Return value	Description
STATUS_OK	Read operation successful
ERR_BAD_ADDRESS	Invalid device address
ERR_INVALID_ARG	Unknown memory type

6.1.8 Function `nvm_set_security_bit()`

Enable the security bit which blocks external read and write access to the device.

```
status_code_t nvm_set_security_bit(void)
```

Returns The memory operation error code.

Table 6-15. Return Values

Return value	Description
STATUS_OK	Security bit set operation successful
ERR_INVALID_ARG	Failed to set the device's security bit

6.1.9 Function `nvm_write()`

Write a number of bytes from the source buffer to a destination address in the Non-Volatile Memory.

```
status_code_t nvm_write(
    mem_type_t mem,
    uint32_t address,
    void * buffer,
    uint32_t len)
```

Table 6-16. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to write
[in]	address	Address to write
[in]	buffer	Pointer to source buffer
[in]	len	Number of bytes to write

Returns The memory operation error code.

Table 6-17. Return Values

Return value	Description
STATUS_OK	Write operation successful
ERR_BAD_ADDRESS	Invalid memory address
ERR_INVALID_ARG	Write failed due to an invalid parameter

6.1.10 Function nvm_write_char()

Write a single byte of data to an address in the Non-Volatile Memory.

```
status_code_t nvm_write_char(
    mem_type_t mem,
    uint32_t address,
    uint8_t data)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[in]	mem	Type of Non-Volatile Memory to write
[in]	address	Address to write
[in]	data	Data to be written

Returns The memory operation error code.

Table 6-19. Return Values

Return value	Description
STATUS_OK	Write operation successful
ERR_BAD_ADDRESS	Invalid device address
ERR_INVALID_ARG	Write/erase failed due to an invalid parameter

6.2 Enumeration Definitions

6.2.1 Enum mem_type_t

Table 6-20. Members

Enum value	Description
INT_FLASH	Internal Flash.
INT_USERPAGE	Userpage/User signature (SAM4/UC3/XMEGA).
INT_EEPROM	Internal EEPROM (XMEGA only).
AT45DBX	External AT45DBX dataflash.

7. Extra Information for Non-Volatile Memory Driver

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
ID	Identity
I/O	Input Output
QSG	Quick Start Guide
TWI	Two Wire Interface

7.2 Dependencies

This driver has the following dependencies:

- Enhanced Embedded Flash Controller (EEFC) - SAM4C/SAM4E/SAM4N/SAM4S devices
- Flash Controller (FLASHCALW) - SAM4L devices only
- FLASH_EFC - SAM4C/SAM4E/SAM4N/SAM4S devices

7.3 Errata

There are no errata related to this driver.

7.4 Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial document release

8. Examples for Non-Volatile Memory Driver

This is a list of the available Quick Start Guides (QSGs) and example applications for the [Common Non-Volatile Memory \(NVM\) driver](#). QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that QSGs can be compiled as a standalone application or be added to the user application.

- [Quick Start Guide for Common NVM driver](#)

8.1 Quick Start Guide for Common NVM driver

This is the quick start guide for the [Common Non-Volatile Memory \(NVM\) driver](#), with step-by-step instructions on how to configure and use the driver in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, for example, the main application function.

8.1.1 Basic Use Case

In this basic use case the NVM driver is configured to use Internal Flash.

8.1.2 Setup Steps

8.1.2.1 Example Code

Add the following to your application's C-file:

```
if (nvm_init(INT_FLASH) == STATUS_OK) {  
    do_something();  
}
```

8.1.2.2 Workflow

1. Ensure that `board_init()` has configured selected I/Os for TWI function when using external AT45DBX dataflash.

Note

This step is only applicable to XMEGA® and Atmel® AVR® 32-bit Microcontrollers.

2. Ensure that the header file `conf_nvm.h` is present for the driver.

Note

This file is only for the driver and should not be included by the user.

3. Call `nvm_init`:

```
nvm_init(INT_FLASH);
```

and optionally check its return code.

8.1.3 Usage Steps

8.1.3.1 Example Code: Writing to Non-Volatile Memory

Use the following in your application's C-file:

```
uint8_t buffer[] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE};
```

```

if (nvm_write(INT_FLASH, test_address, (void *)buffer, sizeof(buffer))
    == STATUS_OK) {
    do_something();
}

```

8.1.3.2 Workflow

1. Prepare the data that you want to send to the Non-Volatile Memory:

```
uint8_t buffer[] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE};
```

2. Call `nvm_write`:

```
nvm_write(INT_FLASH, test_address, (void *)buffer, sizeof(buffer))
```

and optionally check its return value for `STATUS_OK`.

8.1.3.3 Example Code: Reading From Non-Volatile Memory

Use the following in your application's C-file:

```

uint8_t data_read[8];

if (nvm_read(INT_FLASH, test_address, (void *)data_read, sizeof(data_read))
    == STATUS_OK) {
    // Check read content
    if (data_read[0] == 0xAA) {
        do_something();
    }
}

```

8.1.3.4 Workflow

1. Prepare a data buffer that will contain the data read from the Non-Volatile Memory:

```
uint8_t data_read[8];
```

2. Call `nvm_read`:

```
nvm_read(INT_FLASH, test_address, (void *)data_read, sizeof(data_read));
```

and optionally check its return value for `STATUS_OK`. The data read from the Non-Volatile Memory can be found in the `data_read` buffer.

8.1.3.5 Example Code: Erasing a Page of Non-Volatile Memory

Use the following in your application's C-file:

```

if (nvm_page_erase(INT_FLASH, test_page) == STATUS_OK) {
    do_something();
}

```

8.1.3.6 Workflow

- Call `nvm_page_erase`:

```
nvm_page_erase(INT_FLASH, test_page)
```

and optionally check its return value for STATUS_OK.

8.1.3.7 Example Code: Reading Configuration of Non-Volatile Memory

Use the following in your application's C-file:

```
uint8_t mem_size, page_size, page_num;  
  
nvm_get_size(INT_FLASH, &mem_size);  
nvm_get_page_size(INT_FLASH, &page_size);  
nvm_get_pagenumber(INT_FLASH, test_address, &page_num);
```

8.1.3.8 Workflow

1. Define the variables in which to store the configuration of the Non-Volatile Memory:

```
uint32_t mem_size, page_size, page_num;
```

2. Call `nvm_get_size`:

```
nvm_get_size(INT_FLASH, &mem_size);
```

and optionally check its return value for STATUS_OK. The memory size of the Non-Volatile Memory is in `mem_size`.

3. Call `nvm_get_page_size`:

```
nvm_get_page_size(INT_FLASH, &page_size);
```

and optionally check its return value for STATUS_OK. The page size of the Non-Volatile Memory is in `page_size`.

4. Call `nvm_get_pagenumber`:

```
nvm_get_page_number(INT_FLASH, test_address, &page_num);
```

and optionally check its return value for STATUS_OK. The page number of `test_address` in the Non-Volatile Memory is in `page_num`.

8.1.3.9 Example Code: Enabling Security Bit

Use the following in your application's C-file:

```
if (nvm_set_security_bit() == STATUS_OK) {  
    do_something();  
}
```

8.1.3.10 Workflow

- Call `nvm_set_security_bit`:

```
nvm_set_security_bit();
```

and optionally check the return value for STATUS_OK.

Index

E

Enumeration Definitions

mem_type_t, [14](#)

F

Function Definitions

nvm_get_pagenumber, [9](#)

nvm_get_page_size, [9](#)

nvm_get_size, [10](#)

nvm_init, [10](#)

nvm_page_erase, [11](#)

nvm_read, [11](#)

nvm_read_char, [12](#)

nvm_set_security_bit, [12](#)

nvm_write, [12](#)

nvm_write_char, [13](#)

Document Revision History

Doc. Rev.	Date	Comments
42282A	05/2014	Initial document release



Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA T: (+1)(408) 441.0311 F: (+1)(408) 436.4200 | www.atmel.com

© 2014 Atmel Corporation. All rights reserved. / Rev.: 42282A-MCU-05/2014

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, AVR® 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.