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_pagenumber() ...................................... 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

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to get the page size for</td>
</tr>
<tr>
<td>[out]</td>
<td>size</td>
<td>Pointer to where to store the page size</td>
</tr>
</tbody>
</table>

Returns

The memory operation error code.

Table 6-2. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Page size operation successful</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Invalid memory type specified</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory</td>
</tr>
<tr>
<td>[in]</td>
<td>address</td>
<td>Byte address of the non volatile memory</td>
</tr>
<tr>
<td>[out]</td>
<td>num</td>
<td>Pointer to where to store the page number</td>
</tr>
</tbody>
</table>

Returns

The memory operation error code.

Table 6-4. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Page number operation successful</td>
</tr>
</tbody>
</table>
### 6.1.3 Function nvm_get_size()

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

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

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] mem</td>
<td></td>
<td>Type of Non-Volatile Memory to get the size for</td>
</tr>
<tr>
<td>[out] size</td>
<td></td>
<td>Pointer to where to store the size</td>
</tr>
</tbody>
</table>

#### Returns

The memory operation error code.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Size operation successful</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Invalid memory type specified</td>
</tr>
</tbody>
</table>

### 6.1.4 Function nvm_init()

*Initialize the Non-Volatile Memory specified.*

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

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] mem</td>
<td></td>
<td>Type of Non-Volatile Memory to initialize</td>
</tr>
</tbody>
</table>

#### Returns

The memory operation error code.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Initialisation successful</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Unknown memory type</td>
</tr>
<tr>
<td>ERR_NO_MEMORY</td>
<td>Memory check failed</td>
</tr>
</tbody>
</table>
6.1.5 Function nvm_page_erase()

Erase a page in the non-volatile memory.

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

Table 6-9. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to erase</td>
</tr>
<tr>
<td>[in]</td>
<td>page_number</td>
<td>Page number to erase</td>
</tr>
</tbody>
</table>

**Returns**

The memory operation error code.

Table 6-10. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Erase operation successful</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Erase failed due to an invalid parameter</td>
</tr>
</tbody>
</table>

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.

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

Table 6-11. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to read</td>
</tr>
<tr>
<td>[in]</td>
<td>address</td>
<td>Address to read</td>
</tr>
<tr>
<td>[out]</td>
<td>buffer</td>
<td>Pointer to destination buffer</td>
</tr>
<tr>
<td>[in]</td>
<td>len</td>
<td>Number of bytes to read</td>
</tr>
</tbody>
</table>

**Returns**

The memory operation error code.

Table 6-12. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Read operation successful</td>
</tr>
<tr>
<td>ERR_BAD_ADDRESS</td>
<td>Invalid memory address</td>
</tr>
</tbody>
</table>
6.1.7 Function nvm_read_char()

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

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

Table 6-13. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to read</td>
</tr>
<tr>
<td>[in]</td>
<td>address</td>
<td>Address to read</td>
</tr>
<tr>
<td>[out]</td>
<td>data</td>
<td>Pointer to where to store the read data</td>
</tr>
</tbody>
</table>

Returns

The memory operation error code.

Table 6-14. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Read operation successful</td>
</tr>
<tr>
<td>ERR_BAD_ADDRESS</td>
<td>Invalid device address</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Unknown memory type</td>
</tr>
</tbody>
</table>

6.1.8 Function nvm_set_security_bit()

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

```c
status_code_t nvm_set_security_bit(void)
```

Returns

The memory operation error code.

Table 6-15. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Security bit set operation successful</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Failed to set the device's security bit</td>
</tr>
</tbody>
</table>

6.1.9 Function nvm_write()

Write a number of bytes from from the source buffer to a destination address in the Non-Volatile Memory.
```c
def status_code_t nvm_write(
    mem_type_t mem,
    uint32_t address,
    void *buffer,
    uint32_t len)
```

### Table 6-16. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to write</td>
</tr>
<tr>
<td>[in]</td>
<td>address</td>
<td>Address to write</td>
</tr>
<tr>
<td>[in]</td>
<td>buffer</td>
<td>Pointer to source buffer</td>
</tr>
<tr>
<td>[in]</td>
<td>len</td>
<td>Number of bytes to write</td>
</tr>
</tbody>
</table>

**Returns**

The memory operation error code.

### Table 6-17. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Write operation successful</td>
</tr>
<tr>
<td>ERR_BAD_ADDRESS</td>
<td>Invalid memory address</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Write/erase failed due to an invalid parameter</td>
</tr>
</tbody>
</table>

### 6.1.10 Function nvm_write_char()

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

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

### Table 6-18. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mem</td>
<td>Type of Non-Volatile Memory to write</td>
</tr>
<tr>
<td>[in]</td>
<td>address</td>
<td>Address to write</td>
</tr>
<tr>
<td>[in]</td>
<td>data</td>
<td>Data to be written</td>
</tr>
</tbody>
</table>

**Returns**

The memory operation error code.

### Table 6-19. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Write operation successful</td>
</tr>
<tr>
<td>ERR_BAD_ADDRESS</td>
<td>Invalid device address</td>
</tr>
<tr>
<td>ERR_INVALID_ARG</td>
<td>Write/erase failed due to an invalid parameter</td>
</tr>
</tbody>
</table>
6.2 Enumeration Definitions

6.2.1 Enum mem_type_t

Table 6-20. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_FLASH</td>
<td>Internal Flash.</td>
</tr>
<tr>
<td>INT_EEPROM</td>
<td>Internal EEPROM (XMEGA only).</td>
</tr>
<tr>
<td>AT45DBX</td>
<td>External AT45DBX dataflash.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Identity</td>
</tr>
<tr>
<td>I/O</td>
<td>Input Output</td>
</tr>
<tr>
<td>QSG</td>
<td>Quick Start Guide</td>
</tr>
<tr>
<td>TWI</td>
<td>Two Wire Interface</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Changelog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial document release</td>
</tr>
</tbody>
</table>
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:

```c
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:

   ```c
   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:

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

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

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

2. Call nvm_write:

   ```c
   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:

```c
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:

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

2. Call nvm_read:

   ```c
   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:

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

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42282A</td>
<td>05/2014</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>