SmartFusion2 MSS eNVM Driver User's Guide

Version 2.5

Table of Contents

Introduction

The SmartFusion2™ microcontroller subsystem (MSS) includes up to two embedded non-volatile memory (eNVM) blocks. Each of these eNVM blocks can be a maximum size of 256kB. This software driver provides a set of functions for accessing and controlling the MSS eNVM as part of a bare metal system where no operating system is available. The driver can be adapted for use as part of an operating system, but the implementation of the adaptation layer between the driver and the operating system's driver model is outside the scope of the driver.

Features

The MSS eNVM driver provides support for the following features:

- eNVM write (program) operations.
- eNVM page unlocking
- eNVM read page write counter

The MSS eNVM driver is provided as C source code.

Supported Hardware IP

The SmartFusion2 MSS eNVM bare metal driver can be used with the SmartFusion2 MSS version 0.0.500 or higher.

Files Provided

The files provided as part of the MSS eNVM driver fall into three main categories: documentation, driver source code and example projects. The driver is distributed via the Microsemi SoC Products Group's Firmware Catalog, which provides access to the documentation for the driver, generates the driver's source files into an application project and generates example projects that illustrate how to use the driver. The Firmware Catalog is available from: www.microsemi.com/soc/products/software/firmwarecat/default.aspx.

Documentation

The Firmware Catalog provides access to these documents for the driver:

- User's guide (this document)
- Release notes

Driver Source Code

The Firmware Catalog generates the driver's source code into a *drivers\mss_nvm* subdirectory of the selected software project directory. The files making up the driver are detailed below.

mss_nvm.h

This header file contains the public application programming interface (API) of the MSS eNVM software driver. This file should be included in any C source file that uses the MSS eNVM software driver.

mss_nvm.c

This C source file contains the implementation of the MSS eNVM software driver.

Example Code

The Firmware Catalog provides access to example projects illustrating the use of the driver. Each example project is self contained and is targeted at a specific processor and software toolchain combination. The example projects are targeted at the FPGA designs in the hardware development tutorials supplied with the SoC Products Group's development boards. The tutorial designs can be found on the Microsemi SoC Development Kit web page.

Driver Deployment

This driver is deployed from the Firmware Catalog into a software project by generating the driver's source files into the project directory. The driver uses the SmartFusion2 Cortex Microcontroller Software Interface Standard Hardware Abstraction Layer (CMSIS HAL) to access MSS hardware registers. You must ensure that the SmartFusion2 CMSIS HAL is included in the project settings of the software toolchain used to build your project and that it is generated into your project. The most up-to-date SmartFusion2 CMSIS HAL files can be obtained using the Firmware Catalog.

The following example shows the intended directory structure for a SoftConsole ARM® Cortex™-M3 project targeted at the SmartFusion2 MSS. This project uses the MSS eNVM driver. This driver relies on SmartFusion2 CMSIS HAL for accessing the hardware. The contents of the *drivers* directory result from generating the source files for the driver into the project. The contents of the *CMSIS* and *hal* directories result from generating the source files for the SmartFusion2 CMSIS HAL into the project. The contents of the *drivers_config* directory are generated by the Libero project and must be copied into the into the software project.

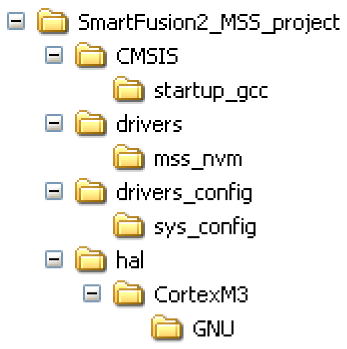

Figure 1 · SmartFusion2 MSS Project Example

Driver Configuration

The size of the MSS eNVM varies with different SmartFusion2 device types. You must only use this driver to access memory locations within the valid MSS eNVM address space for the targeted device. The size of the valid MSS eNVM address space corresponds to the size of the MSS eNVM in the device. Some pages of the MSS eNVM may be write protected by the SmartFusion2 MSS configurator as part of the hardware design flow. The driver cannot unlock or write to these protected pages.

The base address, register addresses and interrupt number assignment for the MSS eNVM blocks are defined as constants in the SmartFusion2 CMSIS HAL. You must ensure that the latest SmartFusion2 CMSIS HAL is included in the project settings of the software tool chain used to build your project and that it is generated into your project.

Application Programming Interface

This section describes the driver's API. The functions and related data structures described in this section are used by the application programmer to control the MSS eNVM peripheral.

Theory of Operation

The total amount of eNVM available in a SmartFusion2 device ranges from 128kB to 512kB, provided in one or two physical eNVM blocks. The eNVM blocks are divided into pages, with each page holding 128 bytes of data. The MSS eNVM driver treats the entire eNVM as a contiguous memory space. It provides write access to all pages that are in the valid eNVM address range for the SmartFusion2 device and that are not write-protected. The driver imposes no restrictions on writing data across eNVM block or page boundaries. The driver supports random access writes to the eNVM memory.

Writing to the eNVM

The *NVM_write()* function is used to program (write) data into the eNVM. The *NVM_unlock()* function is used to unlock the eNVM pages for a specified range of eNVM addresses. The *NVM_unlock()* function must be called before the *NVM_write()* function if the pages in the required memory range are locked. The start and end addresses of the memory range do not have to be page aligned for either function.

Types

nvm_status_t

Prototype

```
typedef enum nvm_status
{
    NVM_SUCCESS = 0,
    NVM_PROTECTION_ERROR,
    NVM_VERIFY_FAILURE,
    NVM_PAGE_LOCK_ERROR,
    NVM_PAGE_LOCK_WARNING,
    NVM_WRITE_THRESHOLD_WARNING,
    NVM_IN_USE_BY_OTHER_MASTER,
    NVM_INVALID_PARAMETER
} nvm_status_t;
```

Description

The *nvm_status_t* enumeration specifies the possible return values from the *NVM_write()* and *NVM_unlock()* functions.

Table 1 · *NVM_write()* and *NVM_unlock()* return values

Enumeration	Description
`NVM_SUCCESS`	Indicates that the programming was successful.

Enumeration	Description
`NVM_PROTECTION_ERROR`	Indicates that the operation could not be completed because of a protection error. This happens when attempting to program a page that was set as protected in the hardware flow.
`NVM_PAGE_LOCK_WARNING`	Indicates that the page write operation completed but page was not locked. This happens in following situations while writing the page with lock. - Access of M2S060 - eNVM0 block memory. - Access of M2S090/150 - eNVM1 block memory. - Access of eNVM memory pages when any protection region in the same block (eNVM0 or eNVM1 block) is write protected.
`NVM_VERIFY_FAILURE`	Indicates that one of the verify operations failed.
`NVM_PAGE_LOCK_ERROR`	Indicates that the operation could not complete because one of the pages is locked. This may happen if the page was locked during a previous call to *NVM_write()* or if the page was locked in the hardware design flow.
`NVM_WRITE_THRESHOLD_WARNING`	Indicates that the NVM maximum number of programming cycles has been reached.
`NVM_IN_USE_BY_OTHER_MASTER`	Indicates that some other MSS AHB Bus Matrix master is accessing the NVM. This could be due to the FPGA logic or the system controller programming the NVM.
`NVM_INVALID_PARAMETER`	Indicates that one of more of the function parameters has an invalid value. This is typically returned when attempting to write or unlock the eNVM for invalid address, data pointer, lock page and more eNVM than is available on the device.

Constant Values

Page locking

These constants are used to specify the *lock_page* parameter for the *NVM_write()* function.

Table 2 · Page Locking Constants

Enumeration	Description
`NVM_DO_NOT_LOCK_PAGE`	Indicates that the *NVM_write()* function should not lock the addressed pages after programming the data.
`NVM_LOCK_PAGE`	Indicates that the *NVM_write()* function should lock the addressed pages after programming the data.

Functions

NVM_write

Prototype

```
nvm_status_t NVM_write
(
    uint32_t start_addr,
    const uint8_t * pidata,
    uint32_t length,
    uint32_t lock_page
);
```

Description

The *NVM_write()* function is used to program (or write) data into the eNVM. This function treats the two eNVM blocks contiguously, so a total of 512kB of memory can be accessed linearly. The start address and end address of the memory range to be programmed do not need to be page aligned. This function supports programming of data that spans multiple pages. This function is a blocking function.

Note: The *NVM_write()* function performs a verify operation on each page programmed to ensure the eNVM is programmed with the expected data.

Parameters

start_addr

The *start_addr* parameter is the byte aligned start address in the eNVM address space, to which the data is to be programmed.

pidata

The pidata parameter is the byte aligned start address of a buffer containing the data to be programmed.

length

The *length* parameter is the number of bytes of data to be programmed.

lock_page

The *lock_page* parameter specifies whether the pages that are programmed must be locked or not once programmed. Locking the programmed pages prevents them from being overwritten by mistake. Subsequent programming of these pages will require the pages to be unlocked prior to calling *NVM_write()*. Allowed values for *lock_page* are:

- NVM_DO_NOT_LOCK_PAGE
- NVM_LOCK_PAGE

Return Value

This function returns NVM_SUCCESS or NVM_WRITE_THRESHOLD_WARNING or NVM_PAGE_LOCK_WARNING on successful execution.

It returns one of the following error codes if the programming of the eNVM fails:

- NVM_PROTECTION_ERROR
- NVM_VERIFY_FAILURE
- NVM_PAGE_LOCK_ERROR
- NVM_IN_USE_BY_OTHER_MASTER
- NVM_INVALID_PARAMETER

Example

```
    uint8_t idata[815] = {"Z"};
    status = NVM_write(0x0, idata, sizeof(idata), NVM_DO_NOT_LOCK_PAGE);
```

NVM_unlock

Prototype

```
nvm_status_t NVM_unlock
(
    uint32_t start_addr,
    uint32_t length
);
```

Description

The *NVM_unlock()* function is used to unlock the eNVM pages for a specified range of eNVM addresses in preparation for writing data into the unlocked locations. This function treats the two eNVM blocks contiguously, so a total of 512kB of memory can be accessed linearly. The start address and end address of the memory range to be unlocked do not need to be page aligned. This function supports unlocking of an eNVM address range that spans multiple pages. This function is a blocking function.

Parameters

start_addr

The *start_addr* parameter is the byte aligned start address, in the eNVM address space, of the memory range to be unlocked.

length

The *length* parameter is the size in bytes of the memory range to be unlocked.

Return Value

This function returns NVM_SUCCESS or NVM_WRITE_THRESHOLD_WARNING on successful execution.

It returns one of the following error codes if the unlocking of the eNVM fails:

- NVM_PROTECTION_ERROR
- NVM_VERIFY_FAILURE
- NVM_PAGE_LOCK_ERROR
- NVM_IN_USE_BY_OTHER_MASTER
- NVM_INVALID_PARAMETER

Example

```
int program_locked_nvm(uint32_t target_addr, uint32_t length)
{
    nvm_status_t status;
    int success = 0;
    status = NVM_unlock(target_addr, length);
    if((NVM_SUCCESS == status)||( NVM_WRITE_THRESHOLD_WARNING == status))
    {
        status = NVM_write(target_addr, buffer, length, NVM_LOCK_PAGE);
        if((NVM_SUCCESS == status) ||( NVM_WRITE_THRESHOLD_WARNING == status))
        {
            success = 1;
        }
    }
    return success;
}
```

NVM_read_page_write_count

Prototype

```
uint32_t NVM_read_page_write_count
(
    uint32_t addr,
);
```

Description

The *NVM_read_page_write_count*() function is used to read the eNVM page write counter value from eNVM auxiliary page. The value returned by *NVM_read_page_write_count*() is the number of times the eNVM page containing the eNVM location specified by the address passed as parameter has been written.

Parameters

addr

The *addr* parameter is the byte aligned address, in the eNVM address space, of the eNVM memory location for which we are requesting to read the page write counter value.

Return Value

This function returns the number of write cycles performed on the eNVM page containing the eNVM memory location specified by the *addr* function parameter. Return '0' if addr is other than eNVM memory or eNVM reserved protection area.

Example

```
#define NVM_ADDRESS     0x60000100u

uint32_t count;
count = NVM_read_page_write_count(NVM_ADDRESS);
```

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **408.643.6913**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Microsemi SoC Products Group Customer Support website for more information and support (http://www.microsemi.com/soc/support/search/default.aspx). Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

50200421-5/06.16