MT793X IoT SDK for eFuse User Guide

Version: 1.0

Release date: 2021-08-02

Version History

Version	Date	Author	Description
1.0	2021-08-02	Ryan Wu	Initial release

Table of Contents

1 Getting Started

This chapter introduces the eFuse feature an gives you an idea of what you need to prepare to get started.

1.1 Overview

The eFuse module is a one-time programmable EEPROM (Electrically Erasable Programmable Read-Only Memory) hereinafter called eFuse. eFuse is separated into triplicates, respectively known as group1, group2, and group3. We can write eFuse value on a specific address by API for blowing eFuse. In group1, the writing address is what location you want to blow on. In group2 and group3, writing address is what mapping table uses and the mapping table loads value of this address to specify the eFuse location. Location is considered as logical address and writing address is considered as physical address.

1.2 Code Layout

driver/chip/mt7933/src_core/inc/hal_efuse.h
driver/chip/mt7933/src_core/src/hal_efuse.c

1.3 eFuse API

hal_efuse_status_t hal_efuse_physical_write(uint32_t group, uint32_t addr, uint32_t *buf);

```
/**
 * @brief Write data into eFuse. This API only provides for GRP1.
 *
 * Write a block size of data in buffer buf into eFuse at the
 * physical address addr
 *
 * @param group efuse group shoud be 0 for GRP1.
 * @param addr eeprom block address to write. example : eeprom address 0x0b0 under GRP1 is equal to
 *         0x0b0 logical address of GRP1.
 * @param buf efuse block size of data to write from.
 * @return HAL_EFUSE_OK if write succeeded.
 * @return HAL_EFUSE_INVALID_PARAMETER group is not supported.
 * @return HAL_EFUSE_INVALID_ACCESS efuse already be blown, address is not supported.
 */
```

hal_efuse_status_t hal_efuse_physical_read(uint32_t group, uint32_t addr, uint32_t *buf);

```
/**
 * Read data from eFuse physical addr.
 *
 * Read a block size of data from physical address addr in eFuse
```

```
 * into buffer buf.
 *
 * @param group efuse group shoud be in range from 0 to 2
 * @param addr eeprom physical address to read from.
 * @param buf efuse block size of data to read to.
 * @return HAL_EFUSE_OK if read succeeded.
 * @return HAL_EFUSE_INVALID_ACCESS address is not supported
 * @return HAL_EFUSE_INVALID_PARAMETER group is not supported.
 */
```

hal_efuse_status_t hal_efuse_logical_read(uint32_t group, uint32_t addr, uint32_t *buf);

```
/**
 * Read data from eFuse logical addr.
 *
 * Read a block size of data from logical address addr in eFuse
 * into buffer buf.
 *
 * @param group efuse group shoud be in range from 1 to 2.
 *          1 for GRP2, 2 for GRP3.
 * @param addr logical block address to read from.
 * @param buf efuse block size of data to read to.
 * @return HAL_EFUSE_INVALID_ACCESS the efuse block address doesn't exist in the mapping table(un-blown).
 * @return HAL_EFUSE_INVALID_PARAMETER group is not supported.
 */
```

int hal_cal_mapping_table(int addr);

```
/**
 * Calculate mapping value
 *
 * Calculate mapping value includes parity.
 *
 * @param addr  logical traget address
 * @return mapping value
 */
```

hal_efuse_status_t hal_efuse_read(uint32_t addr, uint32_t *buf);

```
/**
 * This function provides for ACS and is not used on hal.
 * Read data from eFuse physical addr.
 * Please note, eeprom physical memory of GRP2/GRP3 needs to map on the logical address.
 *
 * @param addr eeprom physical block address to read from
 * @param buf efuse block size of data to read to.
 * @return HAL_EFUSE_OK if read succeeded.
 * @return HAL_EFUSE_INVALID_PARAMETER address is not supported.
 */
```

hal_efuse_status_t hal_efuse_write(uint32_t magic, uint32_t addr, uint32_t *buf);

```
/**
 * This function provides for ACS and is not used on hal.
 * Write a block size of data into eFuse group 1.
 *
 * @param magic check if a match for hal driver
 * @param addr eeprom block address to read from.
 * @param buf efuse block size of data to read to.
 * @return HAL_EFUSE_OK if write succeeded.
 * @return HAL_EFUSE_INVALID_PARAMETER buf is incorrect, address is not supported,
 *       or length is not supported.
 * @return HAL_EFUS
 */
```

hal_efuse_status_t hal_efuse_logical_read_group(uint32_t group);

```
/**
 * Read data from eFuse logical group and print out the data.
 *
 * @param group efuse group shoud be in range from 1 to 2.
 *          1 for GRP2, 2 for GRP3.
 * @return HAL_EFUSE_OK if read succeeded.
 * @return HAL_EFUSE_INVALID_PARAMETER group is incorrect.
 */
```

hal_efuse_status_t hal_efuse_physical_read_group(uint32_t group);

```
/**
 * Read data from eFuse physical group and print out data.
 *
 * @param group efuse group shoud be in range from 0 to 2
 * @return HAL_EFUSE_OK if read succeeded.
 * @return HAL_EFUSE_INVALID_PARAMETER group is out of range.
 */
```

2 eFuse Sample Use Case

2.1 Blowing Data

Write a block of data on 0x10 of physical address of group1.

```
int ret = 0;
uint32_t addr = 0x10; // block address of group1
uint32_t buf[4] = {0xabcd1234, 0xbcde2345, 0xcdef3456, 0x12345678};

ret = hal_efuse_physical_write(0, addr, buf);

if(ret){
        /* error handling */
        return ret;
}

/* writing success */
```

2.2 Reading Data

Read a block of data from 0x10 of physical address of group1.

```
int ret = 0;
uint32_t addr = 0x10; // block address
uint32_t buf[4] = {0};

ret = hal_efuse_physical_read(0, addr, buf);

if(ret){
        /* error handling */
        return ret;
}

/* reading success */
```

Read a block of data from 0x10 of logical address of group2.

```
int ret = 0;
uint32_t addr = 0x10; // block address
uint32_t buf[4] = {0};

ret = hal_efuse_logical_read (1, addr, buf);

if(ret){
```

```
        /* error handling */
        return ret;
    }

    /* reading success */
```

Read data from eFuse logical group2 and print out data.

```
    int ret = 0;
    ret = hal_efuse_logical_read_group (1);

    if(ret){
        /* error handling */
        return ret;
    }

    /* reading success */
```

Read data from eFuse physical group1 and print out data.

```
    int ret = 0;
    ret = hal_efuse_physical_read_group (0);

    if(ret){
        /* error handling */
        return ret;
    }

    /* reading success */
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.