XRADIO EFPG Developer Guide

Revision 1.0

Nov 11, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-11-11	Initial Version

Table 1- 1 Revision History

Contents

Tables

Figures

1 概述

EFPG（eFUSE Programming）模块主要用于 OEM 端对 eFuse 相应区域进行烧写和读取。OEM 端烧写的 eFUSE 区域主要有：HOSC（HOSC TYPE）、BOOT（SECURE BOOT）、DCXO（DCXO TRIM）、POUT（POUT CAL）、MAC 以及 RESERVED（保留区域）。EFPG 模块对上述区域的烧写和读取功能可分为三类：

1. 配合 OEM 端 eFUSE 烧写工具烧写 HOSC、BOOT、DCXO、POUT 和 MAC 数据。
2. 提供接口读取 eFUSE 上 HOSC、BOOT、DCXO、POUT、MAC 以及 CHIPID 数据。
3. 提供接口烧写和读取 eFUSE 上 RESERVED 区域。

1.1 OEM 端 eFUSE 烧写方案

使用 OEM 端 eFUSE 烧写工具可以实现对 HOSC、BOOT、DCXO、POUT 和 MAC 参数的烧写，该烧写方案的整体结构如下图所示。

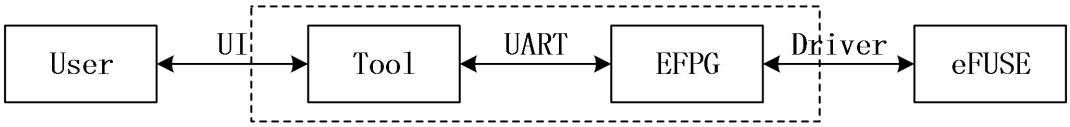

图 1-1 OEM 端 eFUSE 烧写方案结构示意图

OEM 端 eFUSE 烧写工具通过 UART 与 EFPG 模块通信，传输烧写区域和数据等信息。EFPG 模块对工具发送的信息校验通过后，将数据烧写到 eFUSE 相应的区域。

OEM 端 eFUSE 烧写工具的具体使用可参考工具的使用文档。为防止 eFUSE 相应区域被任意烧写，烧写工具中需输入与代码中相同的 Key，代码中 key 在 sdk-code/project/common/cmd/cmd_efpg.c 中配置（如下所示）。Key 的最大长度由宏 EFPG_KEY_LEN_MAX 定义，默认为 64。

```
const char *efpg_key = "efpgtest";
```

1.2 EFPG Field 定义

EFPG Field 表示通过 EFPG 模块可以直接读取的已定义区域的数据。这些已定义区域包括 OEM 端负责烧写的 HOSC、BOOT、DCXO、POUT 和 MAC，此外还包括 CHIPID。代码中对 EFPG Field 的定义为：

```
typedef enum efpg_field {
    EFPG_FIELD_HOSC = 0,    /* data buffer size: 1  byte  */
    EFPG_FIELD_BOOT,        /* data buffer size: 32 bytes */
    EFPG_FIELD_MAC,         /* data buffer size: 6  bytes */
    EFPG_FIELD_DCXO,        /* data buffer size: 1  byte  */
    EFPG_FIELD_POUT,        /* data buffer size: 3  bytes */
    EFPG_FIELD_CHIPID,      /* data buffer size: 16 bytes */
    EFPG_FIELD_UA,          /* data buffer size: V1(1447~2047) : V2(765~1023 bit)*/
    EFPG_FIELD_NUM,
} efpg_field_t;
```

2 使用说明

2.1 代码位置

相关代码位于以下目录：

sdk-code/include/efpg/

sdk-code/src/efpg/

2.2 接口说明

下面对 EFPG 模块提供的接口进行简要说明。

1. 烧写工具发送烧写命令后，会调用 efpg_start()函数，使系统进入 eFUSE 烧写模式。此接口会在使用 OEM 端 eFUSE 烧写工具时自动被调用，烧写结束后工具会通知系统退出 eFUSE 烧写模式。

表 2-1 烧写模式使用 api 说明

function	detail
efpg_start();	声 明： int efpg_start(uint8_t *key, uint8_t key_len, UART_ID uart_id, efpg_cb_t start_cb, efpg_cb_t stop_cb); 目的：使系统进入烧写模式 参数：key：烧写工具发送的 key key_len：key 的长度 uart_id：烧写工具与 EFPG 模块通信的 UART 的 ID start_cb：系统进入烧写模式前，执行的回调函数 stop_cb：系统退出烧写模式后，执行的回调函数 返回值：返回状态（0：成功；-1：失败）

2. 读取指定 EFPG Field 的数据。

表 2-2 读取 EFPG 区域 api 说明

function	detail
efpg_read_mac();	声明：uint16_t efpg_read_mac(uint8_t *r_data); 目的：读取 mac 地址 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）

efpg_read_dcxo();	声明：uint16_t efpg_read_dcxo(uint8_t *r_data); 目的：读取 dcxo 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）
efpg_read_pout();	声明：uint16_t efpg_read_pout(uint8_t *r_data); 目的：读取 pout 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）
efpg_read_chipid();	声明：uint16_t efpg_read_chipid(uint8_t *r_data); 目的：读取 chipid 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）
efpg_read_boot();	声明：uint16_t efpg_read_boot(uint8_t *r_data); 目的：读取 boot 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）
efpg_read_hosc();	声明：uint16_t efpg_read_hosc(uint8_t *r_data); 目的：读取 hosc 参数：r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）
efpg_read_user_area();	声明：uint16_t efpg_read_user_area(uint16_t start, uint16_t num, uint8_t *r_data); 目的：读取用户地址区域 RESERVED（保留区域） 参数：start：读取的起始地址（从第一个比特开始读则为 0） num：读取的比特数 r_data：存放结果的 buffer 返回值：返回状态（0：成功；-1：失败）

3. 烧写 eFUSE 上 OEM RESERVED 区域。

表 2-3 烧写 EFPG 区域 api 说明

function	detail
efpg_write_user_area();	声明：uint16_t efpg_write_user_area(uint16_t start, uint16_t num, uint8_t *data);

	目的：烧写 efuse 上 OEM RESERVED 区域 参数：start：开始烧写的起始地址（从第一个比特开始写则为 0） num：写的比特数 r_data：存放待烧写数据的 buffer 返回值：返回状态（0：成功；-1：失败）

2.3 使用示例

1. 在文件 sdk-code/project/common/cmd/cmd_efpg.c 的函数 cmd_efpg_start()中对 efpg_start()函数的调用示例如下。烧写工具发送烧写命令后将自动调用该函数。此处传入的回调函数 cmd_efpg_start_cb 和 cmd_efpg_stop_cb 目的在于进入烧写模式前关闭控制台，结束烧写模式后开启控制台。

```
static void cmd_efpg_start_cb(void)
{
    console_disable();
    stdout_enable(0);
}

static void cmd_efpg_stop_cb(void)
{
    console_enable();
    stdout_enable(1);
}

static enum cmd_status cmd_efpg_start(void)
{
    const char *efpg_key = "efpgtest";

    cmd_write_respond(CMD_STATUS_OK, "OK");

    UART_ID uart_id = console_get_uart_id();
    if (uart_id >= UART_NUM) {
        CMD_ERR("get uart id failed\n");
        return CMD_STATUS_ACKED;
    }

    if (efpg_start((uint8_t *)efpg_key, strlen(efpg_key), uart_id,
        cmd_efpg_start_cb, cmd_efpg_stop_cb) < 0) {
        CMD_ERR("efpg program failed\n");
        return CMD_STATUS_ACKED;
    }

    return CMD_STATUS_ACKED;
}
```

2. 以读 eFUSE 上的 MAC 为例，调用 efpg_read()函数示例如下。

```
uint8_t data[6];
```

```
efpg_read_mac(data);
```

3. 以烧写和读取 eFUSE 上 OEM RESERVED 区域的前 10 个比特为例，调用 efpg_write_user_area()和 efpg_read_user_area()函数示例如下。

```
uint8_t wr_data[2];
uint8_t rd_data[2];
……
efpg_write_user_area(0, 10, wr_data);
efpg_read_user_area(0, 10, rd_data);
```