XR806 EFPG 中间件

开发指南

版本号：1.0

发布时间：2020-11-18

版本历史

版本	日期	责任人	版本描述
1.0	2020-11-18	AWA 1680	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 平台上 EFPG 模块的使用方法。

1.2 目标读者

XR806 SDK 用户、EFPG 模块开发使用人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示(寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

2 概述

2.1 背景说明

EFPG（eFuse Programming）模块主要用于 OEM（Original Equipment Manufacturer）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 USER AREA（用户区域）。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. 提供接口读取 eFuse 上 USER AREA 区域，且可配合 eFuse 烧写工具对 USER AREA 区域进行烧写。

2.2 规格特性

EFPG 中间件模块提供了以下功能特性。

表 2-1 EFPG 中间件的烧写功能特性

规格类型	规格描述	备注
为 eFuse 烧写工具提供功能命令	支持读取 CHIPID	
	支持烧写、读取 HOSC	
	支持烧写、读取 BOOT	
	支持烧写、读取 DCXO	
	支持烧写、读取 POUT	
	支持烧写、读取 MAC	
	支持烧写、读取 USER AREA	
为用户提供功能接口	支持读取 CHIPID	
	支持读取 HOSC	
	支持读取 BOOT	
	支持读取 DCXO	
	支持读取 POUT	
	支持读取 MAC	
	支持读取 USER AREA	

2.3 文件位置

以 SDK 包为根目录，本中间件涉及到的主要文件位置如下。

表 2-2 EFPG 中间件的文件位置

组件名	文件分类	文件位置
EFPG	源码文件	./src/efpg
	头文件	./include/efpg.h
	示例工程	./project/example/efpg/

关键文件说明如下。

表 2-3 EFPG 中间件的文件说明

文件名	文件说明
efpg.c	EFPG 模块的主函数

说明

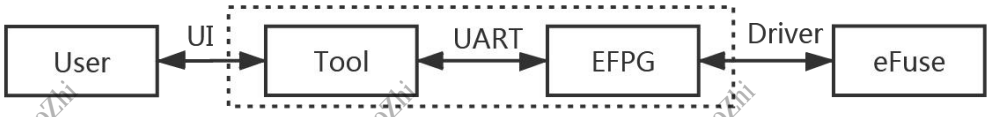
XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

3 技术说明

3.1 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 3-1 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《XR806_eFuse 工具_使用指南》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/project/common/cmd/cmd_efpg.c 文件中（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";
```

3.2 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_WLAN,      /* data buffer size: 6   bytes */
    EFPG_FIELD_DCXO,          /* data buffer size: 1   byte   */
    EFPG_FIELD_POUT_WLAN,     /* 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) : V3(954~1022
                                 bit)*/
    EFPG_FIELD_POUT_BT,       /* data buffer size: 3   bytes */
    EFPG_FIELD_MAC_BT,        /* data buffer size: 6   bytes */
    EFPG_FIELD_SECRETKEY,     /* data buffer size: 16 bytes */
    EFPG_FIELD_SECURESWD,     /* data buffer size: 1   bytes */
    EFPG_FIELD_ALL,           /* data buffer size: 128 bytes */
    EFPG_FIELD_NUM,
```

```
} efpg_field_t;
```

4 应用说明

4.1 应用简述

EFPG 中间件模块已经内嵌到 XR806 SDK，应用步骤如下：

1. 若需要使用 EFPG 模块对 eFuse 区域进行烧写，则只需使用 eFuse 烧写工具进行操作即可（烧写前开发板要正常启动平台初始化，且支持 cmd_efpg 命令，可参见《XR806_eFuse 工具_使用指南》文档）。
2. 若用户需要使用 EFPG 模块的接口，对 eFuse 区域进行读取，则需在用户应用代码中添加 EFPG 模块头文件（参见“2.3 文件位置”章节），调用 EFPG 接口进行使用（接口使用参见“4.3 接口说明”章节）。

4.2 配置说明

XR806 EFPG 模块可对 eFuse 中 DCXO_TRIM、POUT_CAL、BT_POUT_CAL、MAC_WLAN_ADDR、MAC_BT_ADDR 等区域的个数进行配置，配置说明如下表所示。

表 4-1 XR806 EFPG 模块配置列表

配置项	配置说明
DCXO_TRIM 区域个数	设置说明： 此项配置为 eFuse 中 DCXO_TRIM 区域个数（区域个数为整数，范围为[0, 2]） 设置位置： efpg 目录下的 efpg_i.h 文件，SDK/src/efpg/efpg_i.h 设置方式： 配置宏 EFPG_DCXO_TRIM_NUM 的值，如： #define EFPG_DCXO_TRIM_NUM (1) //表示 DCXO_TRIM 区域个数为 1
POUT_CAL 区域个数（WLAN）	设置说明： 此项配置为 eFuse 中 WLAN 的 POUT_CAL 区域个数（整数，范围为[0, 2]） 设置位置： efpg 目录下的 efpg_i.h 文件，SDK/src/efpg/efpg_i.h 设置方式： 配置宏 EFPG_POUT_CAL_NUM 的值，如： #define EFPG_POUT_CAL_NUM (1) //表示 WLAN 的 POUT_CAL 区域个数为 1
BT_POUT_CAL 区域个数（BT）	设置说明： 此项配置为 eFuse 中 BT 的 POUT_CAL 区域个数（整数，范围为[0, 2]）

配置项	配置说明
	设置位置： efpg 目录下的 efpg_i.h 文件，SDK/src/efpg/efpg_i.h 设置方式： 配置宏 EFPG_BT_POUT_CAL_NUM 的值，如： #define EFPG_BT_POUT_CAL_NUM (1) //表示 BT 的 POUT_CAL 区域个数为 1
MAC_WLAN_ADDR 区域个数	设置说明： 此项配置为 eFuse 中 MAC_WLAN_ADDR 区域个数（整数，范围为[0, 4]） 设置位置： efpg 目录下的 efpg_i.h 文件，SDK/src/efpg/efpg_i.h 设置方式： 配置宏 EFPG_MAC_WLAN_ADDR_NUM，如： #define EFPG_MAC_WLAN_ADDR_NUM (0) //表示 MAC_WLAN_ADDR 区域个数为 0
MAC_BT_ADDR 区域个数	设置说明： 此项配置为 eFuse 中 MAC_BT_ADDR 区域个数（整数，范围为[0, 5]） 设置位置： efpg 目录下的 efpg_i.h 文件，SDK/src/efpg/efpg_i.h 设置方式： 配置宏 EFPG_MAC_BT_ADDR_NUM，如： #define EFPG_MAC_BT_ADDR_NUM (1) //表示 MAC_BT_ADDR 区域个数为 1

说明

1. 上述区域个数配置，一般在 eFuse 烧写前统一配置好。待烧写完毕后，各区域个数配置一般不作修改，防止出现数据读取不匹配的情况。
2. 区域个数配置后，其他区域大小会被自动计算和检测，若总长度超出 eFuse 区域总大小，则编译会报错，用户进行修改调整即可。

4.3 接口说明

XR806 EFPG 中间件模块主要包括两类主要接口，一类是与 eFuse 烧写工具的通信接口，用于和 eFuse 烧写工具建立连接，方便烧写工具对 eFuse 区域进行烧写和读取；另一类是为用户提供的 eFuse 区域数据读取接口。EFPG 接口简介如下表所示。

表 4-2 EFPG 中间件模块接口简介

接口名	简要介绍
eFuse 烧写工具通信接口	包括 1 个接口函数，用于 EFPG 模块和 eFuse 烧写工具进行通信。
eFuse 区域数据读取接口	包括 9 个接口函数，为用户提供，方便用户对 eFuse 区域进行数据读取。

上述接口均于 SDK/src/efpg/efpg_efuse.c 文件中实现，XR806 EFPG 中间件模块接口的详细说明如下。

4.3.1 eFuse 烧写工具通信接口

4.3.1.1 efpg_start

表 4-3 efpg_start 接口函数说明

信息项	说明
原型	int efpg_start(uint8_t *key, uint8_t key_len, UART_ID uart_id, efpg_cb_t start_cb, efpg_cb_t stop_cb);
功能	使系统进入烧写模式，与 OEM 端的 eFuse 烧写工具建立通信。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。
参数	uint8_t *key 含义解释：烧写工具发送的 key 使用说明：N/A uint8_t key_len 含义解释：key 的长度 使用说明：N/A UART_ID uart_id 含义解释：烧写工具与 EFPG 模块通信的 UART 的 ID 使用说明：N/A efpg_cb_t start_cb 含义解释：系统进入烧写模式前，执行的回调函数 使用说明：N/A efpg_cb_t stop_cb 含义解释：系统退出烧写模式后，执行的回调函数 使用说明：N/A
返回值	0：成功

信息项	说明
	-1：失败

4.3.2 eFuse 区域数据读取接口

4.3.2.1 efpg_read_mac

表 4-4 efpg_read_mac 接口函数说明

信息项	说明
原型	uint16_t efpg_read_mac(efpg_mac_mode_t mode, uint8_t *r_data);
功能	读取 mac 地址
参数	efpg_mac_mode_t mode 含义解释：mac 地址的读取类型 使用说明：mode 为枚举类型，可参考表 4-5 efpg_mac_mode_t 枚举类型说明。 uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

表 4-5 efpg_mac_mode_t 枚举类型说明

成员项	说明
EFPG_MAC_WLAN	含义解释：表示读取 WLAN 的 mac 地址 使用说明：N/A
EFPG_MAC_BT	含义解释：表示读取 BT 的 mac 地址 使用说明：N/A

4.3.2.2 efpg_read_dcxo

表 4-6 efpg_read_dcxo 接口函数说明

信息项	说明
原型	uint16_t efpg_read_dcxo(uint8_t *r_data);
功能	读取 dcxo
参数	uint8_t *r_data 含义解释：存放读取结果的 buffer

信息项	说明
	使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.3 efpg_read_pout

表 4-7 efpg_read_pout 接口函数说明

信息项	说明
原型	uint16_t efpg_read_pout(efpg_pout_cal_mode_t mode, uint8_t *r_data);
功能	读取 pout
参数	efpg_pout_cal_mode_t mode 含义解释：pout 的读取类型 使用说明：mode 为枚举类型，可参考表 4-8 efpg_pout_cal_mode_t 枚举类型说明。 uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

表 4-8 efpg_pout_cal_mode_t 枚举类型说明

成员项	说明
EFPG_POUT_WLAN	含义解释：表示读取 WLAN 的 pout 使用说明：N/A
EFPG_POUT_BT	含义解释：表示读取 BT 的 pout 使用说明：N/A

4.3.2.4 efpg_read_chipid

表 4-9 efpg_read_chipid 接口函数说明

信息项	说明
原型	uint16_t efpg_read_chipid(uint8_t *r_data);
功能	读取 chipid

信息项	说明
参数	uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.5 efpg_read_boot

表 4-10 efpg_read_boot 接口函数说明

信息项	说明
原型	uint16_t efpg_read_boot(uint8_t *r_data);
功能	读取 boot
参数	uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.6 efpg_read_hosc

表 4-11 efpg_read_hosc 接口函数说明

信息项	说明
原型	uint16_t efpg_read_hosc(uint8_t *r_data);
功能	读取 hosc
参数	uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.7 efpg_read_user_area

表 4-12 efpg_read_user_area 接口函数说明

信息项	说明
原型	uint16_t efpg_read_user_area(uint16_t start, uint16_t num, uint8_t *r_data);

信息项	说明
功能	读取用户地址区域 user area
参数	uint16_t start 含义解释：读取的起始地址 使用说明：若从第一个比特（bit）开始读，则 start 为 0 uint16_t num 含义解释：读取的比特（bit）数 使用说明：N/A uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.8 efpg_read_all_field

表 4-13 efpg_read_all_field 接口函数说明

信息项	说明
原型	uint16_t efpg_read_all_field(uint16_t start, uint16_t num, uint8_t *data);
功能	读取 eFuse 任意区域
参数	uint16_t start 含义解释：读取的起始地址 使用说明：N/A uint16_t num 含义解释：读取的比特（bit）数 使用说明：N/A uint8_t *r_data 含义解释：存放读取结果的 buffer 使用说明：N/A
返回值	EFPG_ACK_OK：成功 EFPG_ACK_RW_ERR：读写错误 EFPG_ACK_NODATA_ERR：无有效数据

4.3.2.9 efpg_layout

表 4-14 efpg_layout 接口函数说明

信息项	说明
原型	void efpg_layout(void);
功能	打印 eFuse 区域布局
参数	无
返回值	无

5 示例说明

EFPG 模块接口主要用于读取指定的 eFuse 区域信息，EFPG 示例工程展示了在 XR806 SDK 中进行 eFuse 区域数据读取的代码实现方法。

说明

eFuse 区域只能烧写一次，对 eFuse 区域的烧写需配合 OEM 端 eFuse 工具进行操作，可参见《XR806_eFuse 工具_使用指南》文档。

5.1 示例简介

本工程示例会读取 eFuse 里的各个区域的字段信息，包括 hosc 信息、boot 信息、mac 地址信息、chipid 信息以及用户区域数据。读取成功，则打印此信息；读取失败，则报错。

5.1.1 获取方式

EFPG 示例有示例工程代码，位于 XR806 SDK 的/project/example/efpg 目录，以下此示例工程简称为 EFPG 示例工程。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

5.1.2 准备工作

EFPG 示例工程的硬件准备有如下。

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 UART0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。

EFPG 示例工程的软件准备，包括烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》。

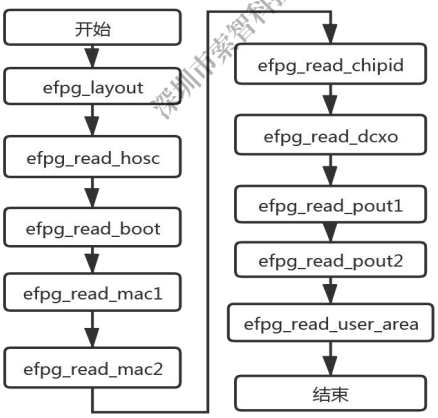
5.1.3 操作步骤

EFPG 示例工程无需控制命令，完成烧写后，复位即可，示例代码自动运行。

5.2 关键实现

系统平台的初始化在 XR806 SDK 的 platform_init()函数中完成，具体请阅读此函数。本文重点描述 EFPG 模块对 eFuse 各区域数据读取的操作过程，示例代码流程如下。

图 5-1 EFPG 示例流程图



具体操作见下面的示例代码。

```
#define EFPG_USER_AREA_LEN 8

static void efpg_log(uint8_t *buf, uint8_t size)
{
    int i;

    for (i = 0; i < size; i++)
        printf("[%02d] 0x%02x\n", i, buf[i]);
    printf("\n");
}

static void efpg_ua_log(uint8_t *buf, int len_bits)
{
    int len_bytes;

extern void print_hex_dump_bytes(const void *addr, size_t len);

    len_bytes = len_bits / 8;
    print_hex_dump_bytes(buf, len_bytes);
}

static int efpg_read_example()
{
    uint16_t ret;
    int len_bits;
    uint8_t buf[32];
    uint8_t ua_buf[128];
    efpg_layout();

    ret = efpg_read_hosc(buf);
```

```
if (ret == EFPG_ACK_OK) {
        printf("efpg read hosc success.\n");
        efpg_log(buf, 1);
} else {
        printf("efpg read hosc fail.ret:%u\n", ret);
}

ret = efpg_read_boot(buf);
if (ret == EFPG_ACK_OK) {
        printf("efpg read boot success.\n");
        efpg_log(buf, 32);
} else {
        printf("efpg read boot fail.ret:%u\n", ret);
}

ret = efpg_read_mac(EFPG_MAC_WLAN, buf);
if (ret == EFPG_ACK_OK) {
        printf("efpg read wlan mac success.\n");
        efpg_log(buf, 6);
} else {
        printf("efpg read wlan mac fail.ret:%u\n", ret);
}

ret = efpg_read_mac(EFPG_MAC_BT, buf);
if (ret == EFPG_ACK_OK) {
        printf("efpg read bt mac success.\n");
        efpg_log(buf, 6);
} else {
        printf("efpg read bt mac fail.ret:%u\n", ret);
}

ret = efpg_read_chipid(buf);
if (ret == EFPG_ACK_OK) {
        printf("efpg read chipid success.\n");
        efpg_log(buf, 16);
} else {
        printf("efpg read chipid fail.ret:%u\n", ret);
}

ret = efpg_read_dcxo(buf);
if (ret == EFPG_ACK_OK) {
        printf("efpg read dcxo success.\n");
        efpg_log(buf, 1);
} else {
```

```
        printf("efpg read dcxo fail.ret:%u\n", ret);
    }

    ret = efpg_read_pout(EFPG_POUT_WLAN, buf);
    if (ret == EFPG_ACK_OK) {
        printf("efpg read wlan pout success.\n");
        efpg_log(buf, 3);
    } else {
        printf("efpg read wlan pout fail.ret:%u\n", ret);
    }

    ret = efpg_read_pout(EFPG_POUT_BT, buf);
    if (ret == EFPG_ACK_OK) {
        printf("efpg read bt pout success.\n");
        efpg_log(buf, 3);
    } else {
        printf("efpg read bt pout fail.ret:%u\n", ret);
    }

    len_bits = EFPG_USER_AREA_LEN;
    ret = efpg_read_user_area(0, len_bits, ua_buf);
    if (ret == EFPG_ACK_OK) {
        printf("efpg read user area success.\n");
        efpg_ua_log(ua_buf, len_bits);
    } else {
        printf("efpg read user area fail.ret:%u\n", ret);
    }

    return 0;
}
```

5.3 效果展示

在评估板中运行 FDCM 示例程序后，在控制台中打印输出如下。

```
---------- efuse layout ----------
********* HOSC type *********
HOSC_TYPE_START:385 bit
HOSC_TYPE_TOTAL_LEN:4 bit
********* SECURE BOOT *********
SECURE_BOOT_START:389 bit
SECURE_BOOT_TOTAL_LEN:276 bit
********* MAC address *********
```

```
MAC_ADDR_START:665 bit
MAC_ADDR_NUM:1
MAC_ADDR_TOTAL_LEN:49 bit
********* WLAN MAC address *********
MAC_WLAN_ADDR_START:714 bit
MAC_WLAN_ADDR_NUM:0
MAC_WLAN_ADDR_TOTAL_LEN:0 bit
********* BT MAC address *********
MAC_BT_ADDR_START:714 bit
MAC_BT_ADDR_NUM:1
MAC_BT_ADDR_TOTAL_LEN:49 bit
********* DCXO TRIM *********
DCXO_TRIM_START:763 bit
DCXO_TRIM_NUM:1
DCXO_TRIM_TOTAL_LEN:9 bit
********* POUT CAL *********
POUT_CAL_START:772 bit
POUT_CAL_NUM:1
POUT_CAL_TOTAL_LEN:22 bit
********* BT POUT CAL *********
POUT_BT_CAL_START:794 bit
POUT_BT_CAL_NUM:1
POUT_BT_CAL_TOTAL_LEN:22 bit
********* USER area *********
USER_AREA_START:816 bit
USER_AREA_LEN:208 bit
---------------------------------------
efpg read hosc success.
[00] 0x00

efpg read boot fail.ret:405
efpg read wlan mac success.
[00] 0x00
[01] 0x67
[02] 0xc4
[03] 0x6a
[04] 0xf3
[05] 0x1d

efpg read bt mac success.
[00] 0x28
[01] 0x41
[02] 0x63
[03] 0x85
```

```
[04] 0x07
[05] 0xb9

efpg read chipid success.
[00] 0x00
[01] 0x00
[02] 0x00
[03] 0x00
[04] 0x00
[05] 0x00
[06] 0x00
[07] 0xf0
[08] 0x00
[09] 0x00
[10] 0x01
[11] 0x23
[12] 0x45
[13] 0x67
[14] 0x89
[15] 0x00

efpg read dcxo success.
[00] 0x8d

efpg read wlan pout success.
[00] 0xb4
[01] 0xa3
[02] 0x12

efpg read bt pout success.
[00] 0x06
[01] 0x42
[02] 0x17

efpg read user area success.

[0x205790]: 99
```

可以看到，EFPG 模块成功读取到 eFuse 的各区域信息。其中，若 eFuse 某些区域未被烧写，则会出现读取失败提示，如示例效果展示中出现“read boot fail”是因为示例板没有烧写过 eFuse 的 boot 区域。

附录 A： 术语表

表 A-1 术语表

B		
BT	Bluetooth	蓝牙
O		
OEM	Original Equipment Manufacturer	原始设备制造商
U		
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发传输器
W		
WLAN	Wireless Local Area Network	无线局域网

著作权声明

商标声明

免责声明