



**Hi3861V100 / Hi3861LV100 EFUSE**

## **使用指南**

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



**HISILICON**、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 上海海思技术有限公司

地址：            深圳市龙岗区坂田华为总部办公楼    邮编：518129

网址：            <https://www.hisilicon.com/cn/>

客户服务邮箱：  [support@hisilicon.com](mailto:support@hisilicon.com)



# 前言

## 概述

本文档主要描述了Hi3861V100/Hi3861LV100的EFUSE控制器的结构及其软件接口的使用方法。

## 产品版本

与本文档对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



## 读者对象

本文档主要适用于以下人员：



- 技术支持工程师
- 软件开发工程师

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 <b>危险</b>	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 <b>警告</b>	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

## 修改记录

文档版本	发布日期	修改说明
01	2020-04-30	第一次正式版本发布。 <ul style="list-style-type: none"><li>更新“<a href="#">表2-1</a>”。</li></ul>
00B02	2020-02-12	<ul style="list-style-type: none"><li>更新“<a href="#">表2-1</a>”。</li><li>更新“<a href="#">表3-1</a>”。</li><li>更新“<a href="#">3.1 接口描述</a>”的EFUSE参数字段ID的枚举。</li></ul>
00B01	2020-01-15	第一次临时版本发布。



# 目 录

前言..... i

1 概述..... 1

2 功能描述..... 2

    2.1 结构..... 2

    2.2 烧写..... 8

    2.3 读取..... 9

    2.4 烧写锁定..... 9

3 接口使用指导..... 11

    3.1 接口描述..... 11

    3.2 开发流程..... 16

    3.3 编程实例..... 17



# 1 概述

EFUSE是一种可编程的存储单元，由于其只可编程一次的特征，多用于芯片保存Chip ID、密钥或其他一次性存储数据。Hi3861提供了2KB（bit[2047:0]）的EFUSE空间，按照用途分割为多个参数字段，通过软件可以控制每个字段的每个bit的烧写，也可以控制锁定位实现各个EFUSE字段的烧写锁定。

## 须知

作为一次性可编程的存储单元，EFUSE的bit一旦被烧写为 1，不能再恢复为 0。

## 说明

本手册中使用到的EFUSE寄存器名称均参考自《Hi3861V100 / Hi3861LV100 / Hi3881V100 WiFi芯片 用户指南》。



# 2 功能描述

- 2.1 结构
- 2.2 烧写
- 2.3 读取
- 2.4 烧写锁定

## 2.1 结构

EFUSE一共分为94个字段，包含：

- 51个系统参数字段
- 2个客户参数字段
- 41个锁定位

各个参数字段的位宽不同。各参数字段名称与位宽等属性如表2-1所示。

表 2-1 EFUSE 参数字段描述

参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
chip_id	bit[7:0]	8	PG 0	1	0	芯片 ID，包含芯片型号、Flash厂家等信息
die_id	bit[199:8]	192	PG 1	1	0	裸片ID，包含芯片的制造厂家、生产日期等信息
pmu_fuse1	bit[209:200]	10	PG 2	1	0	内部使用
pmu_fuse2	bit[219:210]	10	PG 2	1	0	内部使用



参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
flash_encpt_cnt[7:6]	bit[221:220]	2	PG 36	1	1	固件加解密计数4, Flash加密使用, HiBurn自动更新
flash_encpt_cnt[9:8]	bit[223:222]	2	PG 37	1	1	固件加解密计数5, Flash加密使用, HiBurn自动更新
flash_encpt_cnt[11:10]	bit[225:224]	2	PG 38	1	1	固件加解密计数6, Flash加密使用, HiBurn自动更新
secure_boot	bit[233:226]	8	PG 39	1	1	安全启动标识, 0x42为非安全启动, 其他为安全启动
deep_sleep_flag	bit[234]	1	PG 40	1	1	校验失败深睡标识, 写1后FlashBoot校验失败6次, 芯片进入深睡; 默认为校验失败重启
PG36	bit[235]	1	-	1	1	PG36, 置1锁定flash_encpt_cnt[7:6]字段
PG37	bit[236]	1	-	1	1	PG37, 置1锁定flash_encpt_cnt[9:8]字段
PG38	bit[237]	1	-	1	1	PG38, 置1锁定flash_encpt_cnt[11:10]字段
PG39	bit[238]	1	-	1	1	PG39, 置1锁定secure_boot字段
PG40	bit[239]	1	-	1	1	PG40, 置1锁定deep_sleep_flag字段
root_pubkey	bit[495:240]	256	PG 3	1	1	根公钥HASH值, 安全启动使用
root_key	bit[751:496]	256	PG 4	0	1	根密钥, 固件加密或HUKS使用
customer_reserved0	bit[1007:752]	256	PG 5	1	1	用户预留
subkey_cat	bit[1039:1008]	32	PG 6	1	1	二级密钥类别, 安全启动使用
encrypt_flag	bit[1047:1040]	8	PG 7	1	1	flashboot/loaderboot代码段加密标识, 0x42为未加密, 其他为加密, 安全启动使用
rsim	bit[1071:1048]	24	PG 7	1	1	二级密钥吊销标识, 安全启动使用





参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
start_type	bit[1072]	1	PG 2	1	0	内部使用
jtm	bit[1073]	1	PG 8	1	1	JTAG屏蔽位，置1将屏蔽JTAG功能
utm0	bit[1074]	1	PG 9	1	1	UART0屏蔽位，置1将屏蔽UART0功能
utm1	bit[1075]	1	PG 10	1	1	UART1屏蔽位，置1将屏蔽UART1功能
utm2	bit[1076]	1	PG 11	1	1	UART2屏蔽位，置1将屏蔽UART2功能
sdc	bit[1077]	1	PG 12	1	0	内部使用
rsvd0	bit[1078]	1	PG 13	1	1	预留
kdf2ecc_huk_disable	bit[1079]	1	PG 35	1	0	内部使用
SSS_corner	bit[1081:1080]	2	PG 14	1	0	内部使用
uart_halt_interval	bit[1083:1082]	2	PG 15	1	1	BOOT启动时等待HiBurn连接的时间，该时间越短，BOOT启动越快。值0~3分别对应等待时间：32ms、1ms、2ms、16ms。该时间缩短为1ms或2ms时，需要对应修改HiBurn端报文发送间隔为2ms，且一定程度上降低了HiBurn连接成功率
ts_trim	bit[1087:1084]	4	PG 2	1	0	内部使用
chip_id2	bit[1095:1088]	8	PG 16	1	0	芯片 ID 备份
ipv4_mac_addr	bit[1143:1096]	48	PG 17	1	1	IPV4 MAC地址
ipv6_mac_addr	bit[1271:1144]	128	PG 17	1	1	IPV6 MAC地址
pa2gccck0_trim0	bit[1303:1272]	32	PG 18	1	1	802.11b CORE0，功率校准系数，第1轮



参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
pa2gccka1_trim0	bit[1335:1304]	32	PG 18	1	1	802.11b CORE0, 功率校准系数, 第1轮
nvrām_pa2ga0_trim0	bit[1367:1336]	32	PG 19	1	1	802.11g 20M, 功率校准系数, 第1轮
nvrām_pa2ga1_trim0	bit[1399:1368]	32	PG 19	1	1	802.11g 20M, 功率校准系数, 第1轮
pa2gccka0_trim1	bit[1431:1400]	32	PG 20	1	1	802.11b CORE0, 功率校准系数, 第2轮
pa2gccka1_trim1	bit[1463:1432]	32	PG 20	1	1	802.11b CORE0, 功率校准系数, 第2轮
nvrām_pa2ga0_trim1	bit[1495:1464]	32	PG 21	1	1	802.11g 20M, 功率校准系数, 第2轮
nvrām_pa2ga1_trim1	bit[1527:1496]	32	PG 21	1	1	802.11g 20M, 功率校准系数, 第2轮
pa2gccka0_trim2	bit[1559:1528]	32	PG 22	1	1	802.11b CORE0, 功率校准系数, 第3轮
pa2gccka1_trim2	bit[1591:1560]	32	PG 22	1	1	802.11b CORE0, 功率校准系数, 第3轮
nvrām_pa2ga0_trim2	bit[1623:1592]	32	PG 23	1	1	802.11g 20M, 功率校准系数, 第3轮
nvrām_pa2ga1_trim2	bit[1655:1624]	32	PG 23	1	1	802.11g 20M, 功率校准系数, 第3轮
tee_boot_ver	bit[1671:1656]	16	PG 24	1	1*	TEE BOOT版本, 安全启动模式下, 烧写或升级程序时更新
tee_firmware_ver	bit[1719:1672]	48	PG 25	1	1*	TEE FIRMWARE版本, 安全启动模式下, 烧写或升级程序时更新
tee_salt	bit[1847:1720]	128	PG 26	1*	1*	TEE盐值, HUKS软件自动写入
flash_encpt_cnt[1:0]	bit[1849:1848]	2	PG 27	1	1	固件加解密计数1, Flash加密使用, HiBurn自动更新
flash_encpt_cnt[3:2]	bit[1851:1850]	2	PG 28	1	1	固件加解密计数2, Flash加密使用, HiBurn自动更新
flash_encpt_cnt[5:4]	bit[1853:1852]	2	PG 29	1	1	固件加解密计数3, Flash加密使用, HiBurn自动更新



参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
flash_encpt_cfg	bit[1854]	1	PG 30	1	1	固件加解密控制位，置1后开启Flash加密功能
flash_scramble_en	bit[1855]	1	PG 31	1	1	FLASH数据加扰使能
user_flash_ind	bit[1865:1856]	10	PG 31	1	1	FLASH数据加扰盐值
rf_pdbuffer_gcal	bit[1883:1866]	18	PG 32	1	0	内部使用
customer_rsvd1	bit[1947:1884]	64	PG 33	1	1	用户预留
die_id2	bit[2011:1948]	64	PG 34	1	0	裸片ID2
PG0	bit[2012]	1	-	1	0	PG0，置1锁定chip_id字段
PG1	bit[2013]	1	-	1	0	PG1，置1锁定die_id字段
PG2	bit[2014]	1	-	1	0	PG2，置1锁定pmu_fuse1、pmu_fuse2、start_type、ts_trim字段
PG3	bit[2015]	1	-	1	1	PG3，置1锁定root_pubkey字段
PG4	bit[2016]	1	-	1	1	PG4，置1锁定root_key字段
PG5	bit[2017]	1	-	1	1	PG5，置1锁定customer_rsvd0字段
PG6	bit[2018]	1	-	1	1	PG6，置1锁定subkey_cat字段
PG7	bit[2019]	1	-	1	1	PG7，置1锁定encrypt_flag和rsim字段
PG8	bit[2020]	1	-	1	1	PG8，置1锁定jtm字段
PG9	bit[2021]	1	-	1	1	PG9，置1锁定utm0字段
PG10	bit[2022]	1	-	1	1	PG10，置1锁定utm1字段
PG11	bit[2023]	1	-	1	1	PG11，置1锁定utm2字段
PG12	bit[2024]	1	-	1	0	PG12，置1锁定sdc字段
PG13	bit[2025]	1	-	1	1	PG13，置1锁定rsvd0字段
PG14	bit[2026]	1	-	1	0	PG14，置1锁定SSS_corner字段
PG15	bit[2027]	1	-	1	1	PG15，置1锁定uart_halt_interval字段



参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
PG16	bit[2028]	1	-	1	0	PG16, 置1锁定chip_id2字段
PG17	bit[2029]	1	-	1	1	PG17, 置1锁定ipv4_mac_addr和ipv6_mac_addr字段
PG18	bit[2030]	1	-	1	1	PG18, 置1锁定pa2gccka0_trim0和pa2gccka1_trim0字段
PG19	bit[2031]	1	-	1	1	PG19, 置1锁定nvrām_pa2ga0_trim0和nvrām_pa2ga1_trim0字段
PG20	bit[2032]	1	-	1	1	PG20, 置1锁定pa2gccka0_trim1和pa2gccka1_trim1字段
PG21	bit[2033]	1	-	1	1	PG21, 置1锁定nvrām_pa2ga0_trim1和nvrām_pa2ga1_trim1字段
PG22	bit[2034]	1	-	1	1	PG22, 置1锁定pa2gccka0_trim2和pa2gccka2_trim2字段
PG23	bit[2035]	1	-	1	1	PG23, 置1锁定nvrām_pa2ga0_trim2和nvrām_pa2ga1_trim2字段
PG24	bit[2036]	1	-	1	1*	PG24, 置1锁定tee_boot_ver字段
PG25	bit[2037]	1	-	1	1*	PG25, 置1锁定tee_firmware_ver字段
PG26	bit[2038]	1	-	1	1*	PG26, 置1锁定tee_salt字段
PG27	bit[2039]	1	-	1	1	PG27, 置1锁定flash_encpt_cnt[1:0]字段
PG28	bit[2040]	1	-	1	1	PG28, 置1锁定flash_encpt_cnt[3:2]字段
PG29	bit[2041]	1	-	1	1	PG29, 置1锁定flash_encpt_cnt[5:4]字段
PG30	bit[2042]	1	-	1	1	PG30, 置1锁定flash_encpt_cfg字段
PG31	bit[2043]	1	-	1	1	PG31, 置1锁定flash_scramble_en和user_flash_ind字段
PG32	bit[2044]	1	-	1	0	PG32, 置1锁定rf_pdbuffer_gcal字段
PG33	bit[2045]	1	-	1	1	PG33, 置1锁定customer_rsvd1字段



参数字段	bit域	位宽	锁定位	软件可读	软件可写	描述
PG34	bit[2046]	1	-	1	0	PG34，置1锁定die_id2字段
PG35	bit[2047]	1	-	1	0	PG35，置1锁定kdf2ecc_huk_disable字段

## 2.2 烧写

软件烧写通过控制EFUSE寄存器烧写EFUSE的bit[2047:0]。烧写之前，EFUSE中的数据全部为0；烧写使能后，EFUSE将编程地址对应的bit单元烧写为1。

表2-1的“软件可写”属性描述了各EFUSE字段的软件烧写权限，该列中的值：

- 0：不能通过软件烧写。
- 1：可以通过软件烧写。
- 1\*：对应的EFUSE字段只有在nmi\_int\_flag中断触发的特殊场景才能够通过软件烧写，nmi\_int\_flag中断未触发的情况下，不可通过软件对bit[1847:1656]和bit[2038:2036]进行地址编程，如果软件不可烧写时仍对这些地址启动编程操作，控制器将不执行编程，同时会报错。

软件烧写的实现流程如下：

- 步骤1** 读EFUSE\_CTRL\_ST bit[4]，如果为0，则表示EFUSE模块处于空闲状态，没有进行其他操作，此时读bit[2]，根据是否为1判断EFUSE是否已上电加载完成，如果是，则继续执行；否则，等待加载完成。
- 步骤2** 写EFUSE\_PGM\_A bit[10:0]，设置需要烧写的地址。
- 步骤3** 写EFUSE\_PGM\_EN bit[0]为1，使能编程模式。
- 步骤4** 读EFUSE\_PGM\_EN bit[0]，如果为0，则编程操作结束，继续执行；否则，等待编程操作结束。
- 步骤5** 读EFUSE\_CTRL\_ST bit[5]，判断本次烧写是否成功。如果为1，则本次烧写失败；如果为0且bit[0]为1，则烧写成功。

----结束

### 须知

上述步骤结束后，判断当前烧写地址EFUSE\_PGM\_A bit[10:0]。如果为0x73E和0x73F，表示当前非锁定参数字段为flash\_scramble\_en和user\_flash\_ind，则烧写即刻生效；如果为其他非锁定参数字段，则需要下电，重新上电后生效。



## 2.3 读取

表2-2的“软件可读”属性描述了各EFUSE字段是否能够通过软件来读取，该列中的值：

- 0：不能通过软件读取。
- 1：可以通过软件读取。
- 1\*：对应的EFUSE字段只有在nmi\_int\_flag中断触发的特殊场景才能够通过软件读取，nmi\_int\_flag中断未触发的情况下，不能通过软件读取。

须知

EFUSE参数字段如果不可读，对这些地址启动读取操作，读取数据为全零。

软件读取数据的实现流程如下：

- 步骤1** 读EFUSE\_CTRL\_ST bit[4]，如果为0，则表示EFUSE模块处于空闲状态，没有进行其他操作，此时读bit[2]，根据是否为1判断EFUSE是否已上电加载完成，如果是，则继续执行；否则，等待加载完成。
- 步骤2** 给读数据地址寄存器EFUSE\_RD\_A bit[7:0]赋值，每个值对应于EFUSE的8bit数据，对应关系如表2-2所示。

表 2-2 读地址与 EFUSE 数据对应关系

EFUSE_RD_A bit[7:0]	0x0	0x1	0x2	……	0xFF
EFUSE数据	bit[7:0]	bit[15:8]	bit[23:16]	……	bit[2047:2040]

- 步骤3** 写EFUSE\_RD\_EN bit[0]为1，进入读数据模式，每次读出8bit数据。
- 步骤4** 读EFUSE\_RD\_EN bit[0]，根据是否为0判断读模式是否已结束。如果为0，表示读模式已结束，同时如果EFUSE\_CTRL\_ST bit[1]为1，表示读取完成。
- 步骤5** 读EFUSE\_RDATA bit[7:0]，获取所需的8bit数据。
- 结束

## 2.4 烧写锁定

软件通过烧写锁定位为1，锁定对应的非锁定参数字段，对应关系请参见表2-1的“锁定位”列。EFUSE字段锁定后不可再进行地址编程，对这些地址启动编程操作，控制器将不执行编程，同时会报错。例如：flash\_encpt\_cnt[7:6]字段对应的锁定位为PG36，一旦烧写PG36锁定位为1，则flash\_encpt\_cnt[7:6]字段不可再烧写，如果flash\_encpt\_cnt[7:6]字段的某个bit在锁定之后值是0，则该bit的值将永远为0，不可更改。



#### 须知

锁定位和非锁定参数字段并不都是一对一的，部分锁定位可以锁定多个非锁定参数字段（如表2-1所示）。

烧写锁定的软件实现流程如下：

- 步骤1** 读EFUSE\_CTRL\_ST bit[4]，如果为0，则表示EFUSE模块处于空闲状态，没有进行其他操作，此时读EFUSE\_CTRL\_ST bit[2]，根据是否为1判断EFUSE是否已上电加载完成，如果是，则继续执行；否则，等待加载完成。
- 步骤2** 写EFUSE\_PGM\_A bit[10:0]，设置对应地址（锁定位地址范围为0xEB~0xEF和0x7F0~0x7FF）。
- 步骤3** 写EFUSE\_PGM\_EN bit[0]为1，使能编程模式。
- 步骤4** 读EFUSE\_PGM\_EN bit[0]，如果为0，则编程操作结束，继续执行；否则，等待编程操作结束。
- 步骤5** 读EFUSE\_CTRL\_ST bit[5]，判断本次软件烧写动作是否成功。如果为1，则本次烧写失败；如果为0且EFUSE\_CTRL\_ST bit[0]为1，则烧写成功。

----结束

#### 须知

上述步骤结束后，判断当前烧写地址EFUSE\_PGM\_A bit[10:0]。如果为0x7FB，表示当前烧写锁定位为PG31，则锁定即刻生效；否则，当前烧写其他锁定位，则需要下电，重新上电后生效。



# 3 接口使用指导

## 3.1 接口描述

## 3.2 开发流程

## 3.3 编程实例

## 3.1 接口描述

依据上述流程，EFUSE模块提供了两套软件接口：

- ID号索引方式：将非锁参数字段和锁参数字段分别赋予ID号（软件非锁参数字段ID和锁参数字段ID与EFUSE字段的对应关系如表3-1所示），通过ID号索引到对应的EFUSE字段方式实现的接口，包括以下功能接口：
  - hi\_efuse\_get\_id\_size：获取ID号对应的EFUSE字段的长度（单位：bit）。
  - hi\_efuse\_read：从ID号对应的EFUSE字段中读取数据。
  - hi\_efuse\_write：写数据到ID号对应的EFUSE字段。
  - hi\_efuse\_lock：通过锁ID加锁EFUSE中的某个区域，加锁后该区域无法再写入。
  - hi\_efuse\_get\_lockstat：获取EFUSE的锁状态，查询哪些区域已锁定。
- 起始地址方式：指定EFUSE起始地址的读写方式，主要针对用户预留区使用，包括以下功能接口：
  - hi\_efuse\_usr\_read：从指定的起始地址读EFUSE。
  - hi\_efuse\_usr\_write：从指定的起始地址写EFUSE。

表 3-1 EFUSE 参数字段关系表

参数字段	非锁参数字段ID	锁定位ID
chip_id	0	-
die_id	1	-
pmu_fuse1	2	-
pmu_fuse2	3	-





参数字段	非锁参数字段ID	锁定位ID
flash_encpt_cnt[7:6]	4	-
flash_encpt_cnt[9:8]	5	-
flash_encpt_cnt[11:10]	6	-
secure_boot	52	-
deep_sleep_flag	7	-
PG36	-	36
PG37	-	37
PG38	-	38
PG39	-	39
PG40	-	40
root_pubkey	8	-
root_key	9	-
rsvd0	10	-
subkey_cat	11	-
encrypt_flag	12	-
rsim	13	-
start_type	14	-
jtm	15	-
utm0	16	-
utm1	17	-
utm2	18	-
sdc	19	-
rsvd1	20	-
kdf2ecc_huk_disable	21	-
SSS_corner	22	-
uart_halt_interval	23	-
ts_trim	24	-
chip_id2	25	-
ipv4_mac_addr	26	-
ipv6_mac_addr	27	-



参数字段	非锁参数字段ID	锁定位ID
pa2gccka0_trim0	28	-
pa2gccka1_trim0	29	-
nvrnram_pa2ga0_trim0	30	-
nvrnram_pa2ga1_trim0	31	-
pa2gccka0_trim1	32	-
pa2gccka1_trim1	33	-
nvrnram_pa2ga0_trim1	34	-
nvrnram_pa2ga1_trim1	35	-
pa2gccka0_trim2	36	-
pa2gccka1_trim2	37	-
nvrnram_pa2ga0_trim2	38	-
nvrnram_pa2ga1_trim2	39	-
tee_boot_ver	40	-
tee_firmware_ver	41	-
tee_salt	42	-
flash_encpt_cnt[1:0]	43	-
flash_encpt_cnt[3:2]	44	-
flash_encpt_cnt[5:4]	45	-
flash_encpt_cfg	46	-
flash_scramble_en	47	-
user_flash_ind	48	-
rf_pdbuffer_gcal	49	-
customer_rsvd0	50	-
customer_rsvd1	51	-
PG0	-	0
PG1	-	1
PG2	-	2
PG3	-	3
PG4	-	4
PG5	-	5



参数字段	非锁参数字段ID	锁定位ID
PG6	-	6
PG7	-	7
PG8	-	8
PG9	-	9
PG10	-	10
PG11	-	11
PG12	-	12
PG13	-	13
PG14	-	14
PG15	-	15
PG16	-	16
PG17	-	17
PG18	-	18
PG19	-	19
PG20	-	20
PG21	-	21
PG22	-	22
PG23	-	23
PG24	-	24
PG25	-	25
PG26	-	26
PG27	-	27
PG28	-	28
PG29	-	29
PG30	-	30
PG31	-	31
PG32	-	32
PG33	-	33
PG34	-	34
PG35	-	35



EFUSE参数字段ID的枚举：

```
typedef enum {  
    HI_EFUSE_CHIP_RW_ID = 0,  
    HI_EFUSE_DIE_RW_ID = 1,  
    HI_EFUSE_PMU_FUSE1_RW_ID = 2,  
    HI_EFUSE_PMU_FUSE2_RW_ID = 3,  
    HI_EFUSE_FLASH_ENCPY_CNT3_RW_ID = 4,  
    HI_EFUSE_FLASH_ENCPY_CNT4_RW_ID = 5,  
    HI_EFUSE_FLASH_ENCPY_CNT5_RW_ID = 6,  
    HI_EFUSE_DSLEEP_FLAG_RW_ID = 7,  
    HI_EFUSE_ROOT_PUBKEY_RW_ID = 8,  
    HI_EFUSE_ROOT_KEY_WO_ID = 9,  
    HI_EFUSE_CUSTOMER_RSVD0_RW_ID = 10,  
    HI_EFUSE_SUBKEY_CAT_RW_ID = 11,  
    HI_EFUSE_ENCRYPT_FLAG_RW_ID = 12,  
    HI_EFUSE_SUBKEY_RSIM_RW_ID = 13,  
    HI_EFUSE_START_TYPE_RW_ID = 14,  
    HI_EFUSE_JTM_RW_ID = 15,  
    HI_EFUSE_UTM0_RW_ID = 16,  
    HI_EFUSE_UTM1_RW_ID = 17,  
    HI_EFUSE_UTM2_RW_ID = 18,  
    HI_EFUSE_SDC_RW_ID = 19,  
    HI_EFUSE_RSVD0_RW_ID = 20,  
    HI_EFUSE_KDF2ECC_HUK_DISABLE_RW_ID = 21,  
    HI_EFUSE_SSS_CORNER_RW_ID = 22,  
    HI_EFUSE_UART_HALT_INTERVAL_RW_ID = 23,  
    HI_EFUSE_TSENSOR_RIM_RW_ID = 24,  
    HI_EFUSE_CHIP_BK_RW_ID = 25,  
    HI_EFUSE_IPV4_MAC_ADDR_RW_ID = 26,  
    HI_EFUSE_IPV6_MAC_ADDR_RW_ID = 27,  
    HI_EFUSE_PG2GCCA0_TRIM0_RW_ID = 28,  
    HI_EFUSE_PG2GCCA1_TRIM0_RW_ID = 29,  
    HI_EFUSE_NVRAM_PA2GA0_TRIM0_RW_ID = 30,  
    HI_EFUSE_NVRAM_PA2GA1_TRIM0_RW_ID = 31,  
    HI_EFUSE_PG2GCCA0_TRIM1_RW_ID = 32,  
    HI_EFUSE_PG2GCCA1_TRIM1_RW_ID = 33,  
    HI_EFUSE_NVRAM_PA2GA0_TRIM1_RW_ID = 34,  
    HI_EFUSE_NVRAM_PA2GA1_TRIM1_RW_ID = 35,  
    HI_EFUSE_PG2GCCA0_TRIM2_RW_ID = 36,  
    HI_EFUSE_PG2GCCA1_TRIM2_RW_ID = 37,  
    HI_EFUSE_NVRAM_PA2GA0_TRIM2_RW_ID = 38,  
    HI_EFUSE_NVRAM_PA2GA1_TRIM2_RW_ID = 39,  
    HI_EFUSE_TEE_BOOT_VER_RW_ID = 40,  
    HI_EFUSE_TEE_FIRMWARE_VER_RW_ID = 41,  
    HI_EFUSE_TEE_SALT_RW_ID = 42,  
    HI_EFUSE_FLASH_ENCPY_CNT0_RW_ID = 43,  
    HI_EFUSE_FLASH_ENCPY_CNT1_RW_ID = 44,  
    HI_EFUSE_FLASH_ENCPY_CNT2_RW_ID = 45,  
    HI_EFUSE_FLASH_ENCPY_CFG_RW_ID = 46,  
    HI_EFUSE_FLASH_SCRAMBLE_EN_RW_ID = 47,  
    HI_EFUSE_USER_FLASH_IND_RW_ID = 48,  
    HI_EFUSE_RF_PDBUFFER_GCAL_RW_ID = 49,  
    HI_EFUSE_CUSTOMER_RSVD1_RW_ID = 50,  
    HI_EFUSE_DIE_2_RW_ID = 51,  
    HI_EFUSE_SEC_BOOT_RW_ID = 52,  
    HI_EFUSE_IDX_MAX,  
} hi_efuse_idx;
```

锁定位ID的枚举：



```
typedef enum {  
    HI_EFUSE_LOCK_CHIP_ID = 0,  
    HI_EFUSE_LOCK_DIE_ID = 1,  
    HI_EFUSE_LOCK_PMU_FUSE1_FUSE2_START_TYPE_TSENSOR_ID = 2,  
    HI_EFUSE_LOCK_ROOT_PUBKEY_ID = 3,  
    HI_EFUSE_LOCK_ROOT_KEY_ID = 4,  
    HI_EFUSE_LOCK_CUSTOMER_RSVD0_ID = 5,  
    HI_EFUSE_LOCK_SUBKEY_CAT_ID = 6,  
    HI_EFUSE_LOCK_ENCRYPT_RSIM_ID = 7,  
    HI_EFUSE_LOCK_JTM_ID = 8,  
    HI_EFUSE_LOCK_UTM0_ID = 9,  
    HI_EFUSE_LOCK_UTM1_ID = 10,  
    HI_EFUSE_LOCK_UTM2_ID = 11,  
    HI_EFUSE_LOCK_SDC_ID = 12,  
    HI_EFUSE_LOCK_RSVD0_ID = 13,  
    HI_EFUSE_LOCK_SSS_CORNER_ID = 14,  
    HI_EFUSE_LOCK_UART_HALT_INTERVAL_ID = 15,  
    HI_EFUSE_LOCK_CHIP_BK_ID = 16,  
    HI_EFUSE_LOCK_IPV4_IPV6_MAC_ADDR_ID = 17,  
    HI_EFUSE_LOCK_PG2GCCKA0_PG2GCCKA1_TRIM0_ID = 18,  
    HI_EFUSE_LOCK_NVRAM_PA2GA0_PA2GA1_TRIM0_ID = 19,  
    HI_EFUSE_LOCK_PG2GCCKA0_PG2GCCKA1_TRIM1_ID = 20,  
    HI_EFUSE_LOCK_NVRAM_PA2GA0_PA2GA1_TRIM1_ID = 21,  
    HI_EFUSE_LOCK_PG2GCCKA0_PG2GCCKA1_TRIM2_ID = 22,  
    HI_EFUSE_LOCK_NVRAM_PA2GA0_PA2GA1_TRIM2_ID = 23,  
    HI_EFUSE_LOCK_TEE_BOOT_VER_ID = 24,  
    HI_EFUSE_LOCK_TEE_FIRMWARE_VER_ID = 25,  
    HI_EFUSE_LOCK_TEE_SALT_ID = 26,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT0_ID = 27,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT1_ID = 28,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT2_ID = 29,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CFG_ID = 30,  
    HI_EFUSE_LOCK_FLASH_SCRAMBLE_EN_FLASH_IND_ID = 31,  
    HI_EFUSE_LOCK_RF_PDBUFFER_GCAL_ID = 32,  
    HI_EFUSE_LOCK_CUSTOMER_RSVD1_ID = 33,  
    HI_EFUSE_LOCK_DIE_2_ID = 34,  
    HI_EFUSE_LOCK_KDF2ECC_HUK_DISABLE_ID = 35,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT3_ID = 36,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT4_ID = 37,  
    HI_EFUSE_LOCK_FLASH_ENCPY_CNT5_ID = 38,  
    HI_EFUSE_LOCK_SEC_BOOT_ID = 39,  
    HI_EFUSE_LOCK_DSLEEP_FLAG_ID = 40,  
    HI_EFUSE_LOCK_MAX,  
} hi_efuse_lock_id;
```

## 3.2 开发流程

以customer\_rsvd0参数字段为例，ID号索引方式软件接口的使用流程如下：

**步骤1** 调用hi\_efuse\_write接口，写数据到EFUSE中。

**步骤2** 调用hi\_efuse\_read接口，从EFUSE中读取数据。

### 须知

读取EFUSE时，须确保用于存储读取的数据的空间不能小于要读取数据的长度。



**步骤3** 调用hi\_efuse\_lock接口，加锁EFUSE中的某个区域，加锁后该区域无法再写入数据。

----结束

以customer\_rsvd0参数字段为例，起始地址方式软件接口的使用流程如下：

**步骤1** 调用hi\_efuse\_usr\_write接口，写数据到EFUSE中。

**步骤2** 调用hi\_efuse\_usr\_read接口，从EFUSE中读取数据。

#### 须知

- 由于直接通过地址读取EFUSE一次只能读取8bit数据（如表2-2所示），所以调用hi\_efuse\_usr\_read接口读取EFUSE时，如果起始地址或读取长度不是8bit对齐，则需要对起始地址和读取长度进行8bit对齐，读取到的数据还需要进行相应的移位处理。例如：输入起始地址8bit对齐，输入读取位数为10，则读取到的数据右移6bit后才是真正要读取的数据。
- 读取EFUSE时，须确保用于存储读取的数据的空间不能小于要读取数据的长度。

**步骤3** 调用hi\_efuse\_usr\_write接口，烧写锁定位为1，加锁EFUSE中的某个区域，加锁后该区域无法再写入数据。

----结束

## 3.3 编程实例

示例1：customer\_rsvd0字段通过ID号方式的读、写、锁操作。

```
#define EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN 2 /* EFUSE customer_rsvd0字段的长度为64bit, 读写数据需要2个32bit空间来存储 */
hi_void efuse_get_lock_stat(hi_void)
{
    hi_u64 lock_data;

    hi_efuse_get_lockstat(&lock_data);
    printf("lock_stat = 0x%08X ", (hi_u32)((lock_data >> 32) & 0xFFFFFFFF)); /* right shift 32bits
*/
    printf("%08X\n", (hi_u32)(lock_data & 0xFFFFFFFF));
}

hi_u32 efuse_id_read(hi_void)
{
    hi_u32 ret;
    hi_u32 read_data[EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN] = {0};
    hi_efuse_idx efuse_id = HI_EFUSE_CUSTOMER_RSVD0_RW_ID;

    ret = hi_efuse_read(efuse_id, (hi_u8 *)read_data, (hi_u8)sizeof(read_data));
    if (ret != HI_ERR_SUCCESS) {
        printf("Failed to read EFUSE at line%d! Err code = %X\n", __LINE__, ret);
        return ret;
    }
    printf("id_data = 0x%08X %08X\n", read_data[0], read_data[1]);

    return HI_ERR_SUCCESS;
}
```



```
hi_u32 efuse_id_write(hi_void)
{
    hi_u32 ret;
    hi_u32 write_data[EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN] = {
        0x1,
        0x0,
    };
    hi_efuse_idx efuse_id = HI_EFUSE_CUSTOMER_RSVD0_RW_ID;

    ret = efuse_id_read();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }

    ret = hi_efuse_write(efuse_id, (hi_u8 *)write_data);
    if (ret != HI_ERR_SUCCESS) {
        printf("Failed to write EFUSE!\n");
        return ret;
    }

    return HI_ERR_SUCCESS;
}

hi_u32 efuse_id_lock(hi_void)
{
    hi_u32 ret;
    hi_efuse_lock_id lock_id = HI_EFUSE_LOCK_CUSTOMER_RSVD0_ID;
    efuse_get_lock_stat();

    ret = hi_efuse_lock(lock_id);
    if (ret != HI_ERR_SUCCESS) {
        printf("Failed to lock EFUSE!\n");
        return ret;
    }

    efuse_get_lock_stat();

    return HI_ERR_SUCCESS;
}

hi_u32 sample_id_efuse(hi_void)
{
    hi_u32 ret;

#ifdef EFUSE_WRITE_ENABLE
    ret = efuse_id_write();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }
#endif

    ret = efuse_id_read();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }

#ifdef EFUSE_LOCK_ENABLE
    ret = efuse_id_lock();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }
#endif
}
```



```
    return HI_ERR_SUCCESS;  
}
```

示例2: customer\_rsvd0字段通过起始地址的读、写、锁操作。

```
#define EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN 2 /* EFUSE customer_rsvd0字段的长度为  
64bit, 读写数据需要2个32bit空间来存储 */  
hi_void efuse_get_lock_stat(hi_void)  
{  
    hi_u64 lock_data;  
  
    hi_efuse_get_lockstat(&lock_data);  
    printf("lock_stat = 0x%08X ", (hi_u32)((lock_data >> 32) & 0xFFFFFFFF)); /* right shift 32bits  
*/  
    printf("%08X\n", (hi_u32)(lock_data & 0xFFFFFFFF));  
}  
  
hi_u32 efuse_usr_read(hi_void)  
{  
    hi_u32 ret;  
    hi_u32 read_data[EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN] = {0};  
    hi_u16 start_bit = 0x75C; /* customer_rsvd0的偏移地址为0x75C */  
    hi_u16 rw_bits = 64; /* customer_rsvd0的长度为64bit */  
    hi_u16 align_size;  
    hi_u8 diff_head_read = 0;  
    hi_u8 tmp_data[9] = {0}; /* customer_rsvd0地址和长度8bit对齐后读取的的长度为9byte  
( 72bit ) */  
    hi_u64 first_u64;  
    hi_u8 second_u8;  
  
    if ((start_bit & 0x7) != 0x0) {  
        diff_head_read = start_bit % 8; /* 起始地址8bit对齐读取 */  
        start_bit = start_bit - diff_head_read;  
        align_size = rw_bits + diff_head_read;  
    }  
  
    if ((align_size & 0x7) != 0x0) {  
        align_size = ((align_size >> 3) + 1) << 3; /* 3bit移位: 读取长度以8bit为单位 */  
    }  
  
    ret = hi_efuse_usr_read(start_bit, align_size, (hi_u8 *)tmp_data);  
    if (ret != HI_ERR_SUCCESS) {  
        printf("Failed to read EFUSE at line%d! Err code = %X\n", __LINE__, ret);  
        return ret;  
    }  
  
    first_u64 = *(hi_u64 *)&tmp_data[0];  
    second_u8 = *(hi_u8 *)&tmp_data[8]; /* the last u8 bit */  
    /* 丢弃第一个u64多读的低位(diff_head_read位) */  
    first_u64 = first_u64 >> diff_head_read;  
    /* 取第二个char的低位, 作为第一个u64的高位(diff_head_read位) */  
    first_u64 = first_u64 | ((hi_u64)second_u8 << (64 - diff_head_read)); /* 左移(64 -  
diff_head_read)bits */  
    *(hi_u64 *)read_data = first_u64;  
  
    printf("usr_data = 0x%08X %08X\n", read_data[0], read_data[1]);  
  
    return HI_ERR_SUCCESS;  
}  
  
hi_u32 efuse_usr_write(hi_void)
```





```
{
    hi_u32 ret;
    hi_u32 write_data[EFUSE_USR_RW_SAMPLE_BUFF_MAX_LEN] = {
        0x0,
        0x1,
    };
    hi_u16 start_bit = 0x75C; /* customer_rsvd0的偏移地址为0x75C */
    hi_u16 rw_bits = 64; /* customer_rsvd0的长度为64bit */

    ret = efuse_usr_read();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }

    ret = hi_efuse_usr_write(start_bit, rw_bits, (hi_u8 *)write_data);
    if (ret != HI_ERR_SUCCESS) {
        printf("Failed to write EFUSE!\n");
        return ret;
    }

    return HI_ERR_SUCCESS;
}

hi_u32 efuse_usr_lock(hi_void)
{
    hi_u32 ret;
    hi_u8 lock_data = 0x1;
    hi_u16 lock_start_bit = 0x7FD; /* customer_rsvd0锁的偏移地址为0x7FD */
    hi_u16 lock_bits = 1; /* customer_rsvd0锁的长度为1bit */

    efuse_get_lock_stat();

    ret = hi_efuse_usr_write(lock_start_bit, lock_bits, &lock_data);
    if (ret != HI_ERR_SUCCESS) {
        printf("Failed to lock EFUSE!\n");
        return ret;
    }

    efuse_get_lock_stat();

    return HI_ERR_SUCCESS;
}

hi_u32 sample_usr_efuse(hi_void)
{
    hi_u32 ret;

#ifdef EFUSE_WRITE_ENABLE
    ret = efuse_usr_write();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }
#endif

    ret = efuse_usr_read();
    if (ret != HI_ERR_SUCCESS) {
        return ret;
    }

#ifdef EFUSE_LOCK_ENABLE
    ret = efuse_usr_lock();
    if (ret != HI_ERR_SUCCESS) {
```



```
        return ret;
    }
#endif

    return HI_ERR_SUCCESS;
}
```