



# Linux SID 开发指南

版本号: 2.8  
发布日期: 2025.03.26

## 版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.7.19	AWA1835	初始版本
2.0	2020.11.11	AWA1538	适配 linux-5.4 平台。
2.1	2022.03.04	AWA1832	更新函数说明。
2.1	2022.03.04	AWA1832	此文档今后只支持内核版本为 Linux-5.4 及之后的平台。
2.2	2022.03.08	AWA1768	修改 sid 的源码路径。
2.3	2023.11.25	XAA0248	修改格式问题。
2.4	2024.07.15	XAA0175	修改部分格式问题，优化有歧义的内容，增加章节“动态调试接口”。
2.5	2024.10.18	XAA0190	1. 增加第 6 章节“FAQ”
2.6	2024.12.09	XAA0190	1. 修改第 6 章节“FAQ”格式
2.7	2025.02.11	XAA0190	1. 修改 2.3 章节，模块 device tree 配置内容 2. 修改第 3 章节，关键数据结构内容 3. 修改第 4 章节内部函数接口
2.8	2025.03.26	XAA0190	1. 修改 2.3 章节，模块 device tree 配置内容

# 目 录

<b>1 前言</b>	<b>1</b>
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
1.4 术语、定义、缩略语	1
<b>2 模块描述</b>	<b>2</b>
2.1 模块功能	2
2.1.1 Chip ID 功能	2
2.1.2 SoC Version 功能	2
2.1.3 Efuse 功能	2
2.1.4 一些状态位	3
2.2 模块位置	3
2.3 模块 device tree 配置说明	3
2.4 模块源码结构	4
2.5 内核配置	5
2.5.1 SID 驱动	5
2.5.2 sysinfo 驱动	5
<b>3 模块设计</b>	<b>6</b>
3.1 结构框图	6
3.2 关键数据定义	6
3.2.1 常量及宏定义	6
3.2.1.1 key 的名称定义	6
3.2.2 关键数据结构	7
3.2.2.1 soc_ver_map	7
3.2.2.2 soc_ver_reg	8
3.2.3 全局变量	8
3.3 模块流程设计	8
3.3.1 SoC 信息读取流程	8
3.3.2 Efuse Key 读取流程	9
<b>4 接口设计</b>	<b>10</b>
4.1 接口函数	10
4.1.1 s32 sunxi_get_platform(s8 *buf, s32 size)	10
4.1.2 int sunxi_get_soc_chipid(u8 *chipid)	10
4.1.3 int sunxi_get_serial(u8 *serial)	10
4.1.4 sunxi_get_soc_chipid_str(char *serial)	11
4.1.5 int sunxi_get_soc_ft_zone_str(char *serial)	11

4.1.6	int sunxi_get_soc_rotpk_status_str(char *status) . . . . .	11
4.1.7	int sunxi_soc_is_secure(void) . . . . .	11
4.1.8	unsigned int sunxi_get_soc_bin(void) . . . . .	12
4.1.9	unsigned int sunxi_get_soc_ver(void) . . . . .	12
4.1.10	s32 sunxi_efuse_readn(void *key_name, void *buf, u32 n) . . . . .	12
4.2	内部函数 . . . . .	13
4.2.1	static s32 sunxi_sid_get_base(struct device_node **pnode, void __iomem **base, s8 *compatible, u32 sec) . . . . .	13
4.2.2	static void sunxi_sid_put_base(struct device_node *pnode, void __iomem *base, u32 sec) . . . . .	13
4.2.3	static u32 sid_rd_bits(s8 *name, u32 offset, u32 shift, u32 mask, u32 sec) . . . . .	13
<b>5</b>	<b>可测试性</b>	<b>15</b>
5.1	sysfs 调试接口 . . . . .	15
5.2	动态调试接口 . . . . .	15
5.3	应用层调用示例 . . . . .	15
<b>6</b>	<b>FAQ</b>	<b>17</b>



# 1 前言

## 1.1 编写目的

介绍 Linux 内核中基于 Sunxi 硬件平台的 SID 模块驱动的详细设计，为软件编码和维护提供基础。

## 1.2 适用范围

内核版本为 Linux-5.4 及之后的平台。

## 1.3 相关人员

SID 驱动、Efuse 驱动、Sysinfo 驱动的维护、应用开发人员等。

## 1.4 术语、定义、缩略语

表 1-1: 术语表

术语或缩略	描述
Efuse	Electronic Fuse，电子熔丝
SID	Security ID，特指 AW SoC 中的 SID 模块
Sysinfo	一个字符型设备驱动，用于方便用户空间获取、调试 SID 信息
sunxi	指 Allwinner 的一系列 SOC 硬件平台。

## 2 模块描述

### 2.1 模块功能

SID 提供的功能可以分为四大部分：ChipID、SoC Version、Efuse 功能，以及一些状态位。

#### 2.1.1 Chip ID 功能

对于全志的 SoC 来说，ChipID 用于该 SoC 的唯一标识，如 A83 的 ChipID 用于标识任一颗 A83 SoC 在所有的 A83 中是唯一的（目前仅保证同一型号 SoC 中的 ChipID 唯一）。

ChipID 由 4 个 word（16 个 byte）组成，共 128bit，通常放在 Efuse（见 2.1.3 节）的起始 4 个 word。

#### 2.1.2 SoC Version 功能

严格讲 SoC Version 包含两部分信息：

1. Bonding ID，表示不同封装。
2. Version，表示改版编号。

##### 📖 说明

这两个信息所在的寄存器不一定都在 SID 模块内部，且各平台位置不一，但软件上为了统一管理，都归属为 SID 模块。

BSP 会返回这两个信息的组合值，由应用去判断和做出相应的处理。

#### 2.1.3 Efuse 功能

对软件来说，Efuse 中提供了一个可编程的永久存储空间，特点是每一位只能写一次（从 0 写为 1）。

Efuse 接口方式：Efuse 容量大于 512bit 采用 SRAM 方式。带有 SRAM 的硬件结构示意图如下：

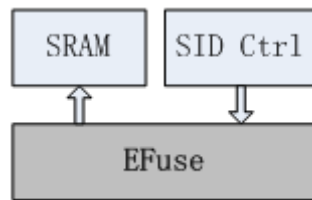


图 2-1: 带有 SRAM 的 SID 模块硬件结构

## 2.1.4 一些状态位

Secure Enable 标志位。

1. 标明当前系统的 Security 属性是否打开，即是否运行了 SecureBoot 和 SecureOS。
2. 芯片 SecureEnable 状态位通常保存在 SID 模块的 0xa0 寄存器（Secure Mode Status Register）。

## 2.2 模块位置

SID 是一个比较独立的模块，在 Linux 内核中没有依赖其他子系统，在 Sunxi 平台存放在 `{SDK}/bsp/drivers/sid/` 目录中。SID 为其他模块提供 API 的调用方式。关系如下图：

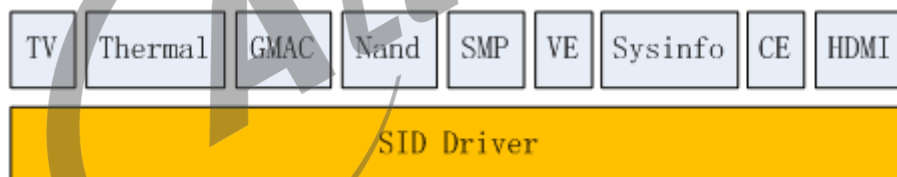


图 2-2: SID 与其他模块的关系

- TV、Thermal、GMAC 的校准参数保存在 SID 中
- Nand、SMP、VE 需要读取 SoC Version
- CE 和 HDMI 会用到 SID 中的一些 Key
- Sysinfo 比较特殊。它是为了方便用户空间获取、调试 SID 信息，而专门设计的一个字符型设备驱动

## 2.3 模块 device tree 配置说明

在 soc 级的 `dtsi` 文件中配置了寄存器基地址等共性信息。soc 级的 `dtsi` 文件的路径为：`{SDK}/bsp/configs/{linux-ver}/{CHIP}.dtsi`。其中 `{CHIP}` 为研发代号，如 `sun50iw10p1`。

以sun50iw10p1为例，需要在sun50iw10p1.dtsi中包含如下配置：

```
sid@3006000 {
    compatible = "allwinner,sun50iw10p1-sid", "allwinner,sunxi-sid";
    reg = <0x0 0x03006000 0 0x1000>;
    #address-cells = <1>;
    #size-cells = <1>;
    non-secure-maxoffset = <0x80>;
    non-secure-maxlen = <0x20>;

    secure_status {
        reg = <0x0 0>;
        offset = <0xa0>;
        size = <0x4>;
    };
    chipid {
        reg = <0x0 0>;
        offset = <0x200>;
        size = <0x10>;
    };
    rotpk {
        reg = <0x0 0>;
        offset = <0x140>;
        size = <0x20>;
    };
};

sram_ctrl: sram_ctrl@3200000 {
    compatible = "allwinner,sram_ctrl";
    reg = <0x0 0x03200000 0 0x184>;
    soc_ver {
        offset = <0x24>;
        mask = <0x7>;
        shift = <0>;
    };
};
```

在 dts 的 sid 节点下增加子节点secure\_status, chipid, rotpk，就可以用 sysfs 节点key\_info来访问：

```
console:/ # echo secure_status > /sys/class/sunxi_info/key_info; cat /sys/class/sunxi_info/key_info
console:/ # 1
```

sram\_ctrl 节点用于 soc\_version 功能的配置，直接使用即可，无需更改

## 2.4 模块源码结构

SID 驱动的源码如下：

```
$(SDK)/bsp/
├── include/
│   └── sunxi-sid.h
├── drivers/smc/
│   └── sunxi-sysinfo.c
├── drivers/sid/
│   ├── Kconfig
│   └── Makefile
```

```
└─ sunxi-sid.c
```

## 2.5 内核配置

### 2.5.1 SID 驱动

SID 驱动的内核配置项为CONFIG\_AW\_SID。此配置项一般默认打开，不需要重新配置。

若要配置，可在 SDK 根目录执行./build.sh menuconfig：

```
Allwinner BSP
├─> Device Drivers
│   └─> SID Drivers
│       └─> Allwinner SID Support
```

### 2.5.2 sysinfo 驱动

若要在用户空间通过 sysfs 访问 efuse 里的值，则需要打开 sysinfo 驱动。对应的配置项为CONFIG\_AW\_SYS\_INFO，驱动代码为drivers/smc/sunxi-sysinfo.co

若要配置，可在 SDK 根目录执行./build.sh menuconfig：

```
Allwinner BSP
├─> Device Drivers
│   └─> SMC Drivers
│       └─> Allwinner System Info Driver
```

## 3 模块设计

### 3.1 结构框图

SID 驱动内部的功能划分如下图所示：



图 3-1: SID 驱动的内部结构

总体上，SID 驱动内部可以分为两大部分：

1. SID Register RW，封装了对寄存器按位读取的接口，以及获取指定 compatible 的模块基地址等。
2. SID Api，以 API 的方式提供一些功能接口：获取 Key、获取 SoC Version、获取 SecureEnable、获取 ChipID 等。

### 3.2 关键数据定义

#### 3.2.1 常量及宏定义

##### 3.2.1.1 key 的名称定义

在获取 Key 的时候，调用者需要知道 Key 的名称，以此作为索引的依据。Key 名称详见 `sunxi-sid.h`：

```
#define EFUSE_CHIPID_NAME    "chipid"
#define EFUSE_BROM_CONF_NAME "brom_conf"
#define EFUSE_BROM_TRY_NAME "brom_try"
#define EFUSE_THM_SENSOR_NAME "thermal_sensor"
#define EFUSE_FT_ZONE_NAME  "ft_zone"
#define EFUSE_TV_OUT_NAME   "tvout"
#define EFUSE_OEM_NAME      "oem"
```

```
#define EFUSE_WR_PROTECT_NAME    "write_protect"
#define EFUSE_RD_PROTECT_NAME    "read_protect"
#define EFUSE_IN_NAME            "in"
#define EFUSE_ID_NAME            "id"
#define EFUSE_ROTPK_NAME         "rotpk"
#define EFUSE_SSK_NAME           "ssk"
#define EFUSE_RSSK_NAME          "rssk"
#define EFUSE_HDCP_HASH_NAME     "hdcp_hash"
#define EFUSE_HDCP_PKF_NAME      "hdcp_pkf"
#define EFUSE_HDCP_DUK_NAME      "hdcp_duk"
#define EFUSE_EK_HASH_NAME       "ek_hash"
#define EFUSE_SN_NAME            "sn"
#define EFUSE_NV1_NAME           "nv1"
#define EFUSE_NV2_NAME           "nv2"
#define EFUSE_BACKUP_KEY_NAME    "backup_key"
#define EFUSE_RSAKEY_HASH_NAME   "rsakey_hash"
#define EFUSE_RENEW_NAME         "renewability"
#define EFUSE_OPT_ID_NAME        "operator_id"
#define EFUSE_LIFE_CYCLE_NAME    "life_cycle"
#define EFUSE_JTAG_SECU_NAME     "jtag_security"
#define EFUSE_JTAG_ATTR_NAME     "jtag_attr"
#define EFUSE_CHIP_CONF_NAME     "chip_config"
#define EFUSE_RESERVED_NAME     "reserved"
#define EFUSE_RESERVED2_NAME    "reserved2"
/* For KeyLadder */
#define EFUSE_KL_SCK0_NAME       "keyladder_sck0"
#define EFUSE_KL_KEY0_NAME       "keyladder_master_key0"
#define EFUSE_KL_SCK1_NAME       "keyladder_sck1"
#define EFUSE_KL_KEY1_NAME       "keyladder_master_key1"
```

#### 说明

并非sunxi-sid.h里的所有 key 都能访问。一般可以访问的已经在 dts 定义。

## 3.2.2 关键数据结构

### 3.2.2.1 soc\_ver\_map

用于管理多个 SoC 的 Version 信息，方使用查表的方式实现 SoC Version API。其中有两个分量：

1. id，即 BondingID；
2. rev[]，用于保存 BondingID 和 Version 的各种组合值。

```
#define SUNXI_VER_MAX_NUM 8
struct soc_ver_map {
    u32 id;
    u32 rev[SUNXI_VER_MAX_NUM];
};
```

对于一个 SoC 定义一个 soc\_ver\_map 结构数组，使用 id 和不同 Version 在 rev[] 中查找对应的组合值。

### 3.2.2.2 soc\_ver\_reg

SoC Version、BondingID、SecureEnable 的存储位置因 SoC 而异，所以定义了一个结构来记录这类信息的位置，包括属于那个模块（基地址）、偏移、掩码、位移等。定义见sunxi-sid.c：

```
#define SUNXI_SOC_ID_INDEX 1
#define SUNXI_SECURITY_ENABLE_INDEX 2
struct soc_ver_reg {
    u32 offset;
    u32 mask;
    u32 shift;
    struct soc_ver_map ver_map;
};
```

每个 SoC 会定义一个 soc\_ver\_reg 数组，目前各元素的定义如下：

1. SoC Version 信息在寄存器中的位置。
2. BondingID 信息在寄存器中的位置。
3. SecureEnable 信息在寄存器中的位置。

### 3.2.3 全局变量

定义几个 static 全局变量，用于保存解析后的 ChipID、SoC\_Ver 等信息。

```
static unsigned int sunxi_soc_chipid[4];
static unsigned int sunxi_serial[4];
static int sunxi_soc_secure;
static unsigned int sunxi_soc_bin;
static unsigned int sunxi_soc_ver;
```

## 3.3 模块流程设计

### 3.3.1 SoC 信息读取流程

本节中，这里把 SoC Ver、ChipID、SecureEnable 信息统称为“SoC 信息”，因为他们的读取过程非常相似。都是遵循以下流程：

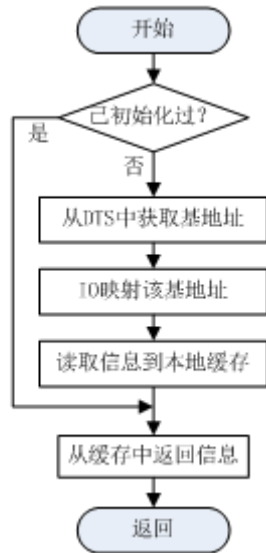


图 3-2: SoC 信息读取流程

### 3.3.2 Efuse Key 读取流程

在读取 Efuse 中 Key 的时候，需要判断是否存在、以及访问权限，过程稍微复杂些，用以下流程图进行简单说明。

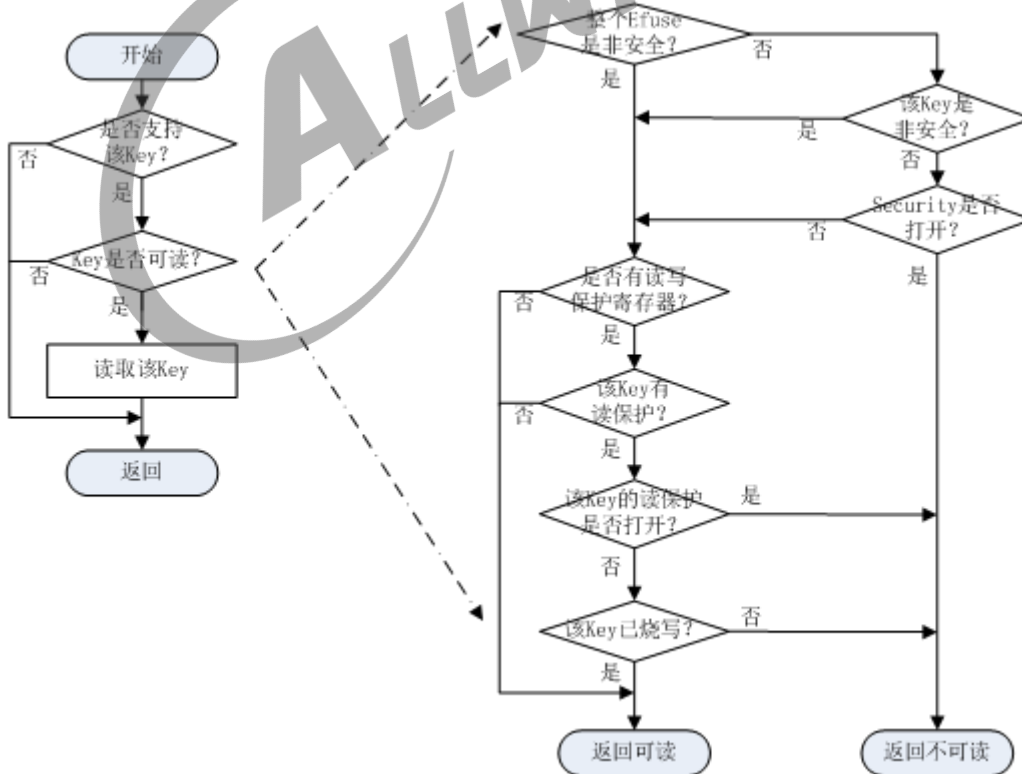


图 3-3: Efuse Key 的读取流程

## 4 接口设计

### 4.1 接口函数

#### 4.1.1 s32 sunxi\_get\_platform(s8 \*buf, s32 size)

- 作用：获取 SoC 平台的名称，实际上是一个 BSP 研发代号，如sun8iw11。
- 参数：
  - buf: 用于保存平台名称的缓冲区。
  - size: buf的大小。
- 返回：
  - 返回buf中平台名称的实际拷贝长度（如果size/小于名称长度，返回size）。

#### 4.1.2 int sunxi\_get\_soc\_chipid(u8 \*chipid)

- 作用：获取 SoC 的 ChipID（从 Efuse 中读到的原始内容，包括数据内容和顺序）。
- 参数：
  - chipid: 用于保存 ChipID 的缓冲区。
- 返回：
  - 会返回0，无实际意义。

#### 4.1.3 int sunxi\_get\_serial(u8 \*serial)

- 作用：获取 SoC 的序列号（由 ChipID 加工而来）。
- 参数：
  - serial: 用于保存序列号的缓冲区。
- 返回：
  - 会返回0，无实际意义。

#### 4.1.4 sunxi\_get\_soc\_chipid\_str(char \*serial)

- 作用：获取 SoC 的 ChipID 的第一个字节，要求转换为字符串格式。
- 参数：
  - serial：用于打印 ChipID 第一个字节的缓冲区。
- 返回：
  - 只会返回8（4个字节的十六进制打印长度），无实际意义。

#### 4.1.5 int sunxi\_get\_soc\_ft\_zone\_str(char \*serial)

- 作用：获取 FZ ZONE 的最后一个字节，要求转换为字符串格式。
- 参数：
  - serial：用于打印 ChipID 第一个字节的缓冲区。
- 返回：
  - 只会返回8（4个字节的十六进制打印长度），无实际意义。

#### 4.1.6 int sunxi\_get\_soc\_rotpk\_status\_str(char \*status)

- 作用：获取 rotpk 的状态，是否烧码。
- 参数：
  - status：用于记录是否烧码的缓冲区；0，未烧；1，已烧。
- 返回：
  - %d的长度，无实际意义。

#### 4.1.7 int sunxi\_soc\_is\_secure(void)

- 作用：获取整个系统的 Secure 状态，即安全系统是否启用。
- 参数：
  - 无。
- 返回：
  - 0，未启用安全系统；1，启用。

### 4.1.8 unsigned int sunxi\_get\_soc\_bin(void)

- 作用：用于芯片分 bin，部分 SoC 平台才支持。
- 参数：
  - 无。
- 返回：
  - 0: fail。
  - 1: normal。
  - 2: faster。
  - 3: fastest。

### 4.1.9 unsigned int sunxi\_get\_soc\_ver(void)

- 作用：获取 SoC 的版本信息。
- 参数：
  - 无。
- 返回：
  - 返回一个十六进制的编号，需要调用者去判断版本号然后做出相应的处理。详情参看 dts 的 sid 节点。

### 4.1.10 s32 sunxi\_efuse\_readn(void \*key\_name, void \*buf, u32 n)

- 作用：读取 Efuse 中的一个 key 信息。
- 参数：
  - key\_name - Key 的名称，定义详见 sunxi-sid.h。
  - buf - 用于保存 Key 值的缓冲区。
  - size - buf 的大小。
- 返回：
  - 0: success。
  - other: fail。

## 4.2 内部函数

### 4.2.1 static s32 sunxi\_sid\_get\_base(struct device\_node \*\*pnode, void \_\_iomem \*\*base, s8 \*compatible, u32 sec)

- 作用：从 dts 中获取指定模块的寄存器基地址。
- 参数：
  - pnode - 用于保存获取到的模块 node 信息。
  - base - 用于保存获取到的寄存器基地址。
  - compatible - 模块名称，用于匹配 dts 中的模块。
- 返回：
  - 0: success。
  - other: fail。

### 4.2.2 static void sunxi\_sid\_put\_base(struct device\_node \*pnode, void \_\_iomem \*base, u32 sec)

- 作用：释放一个模块的基地址。
- 参数：
  - pnode - 保存模块 node 信息。
  - base - 该模块的寄存器基地址。
- 返回：
  - 无。

### 4.2.3 static u32 sid\_rd\_bits(s8 \*name, u32 offset, u32 shift, u32 mask, u32 sec)

- 作用：从一个模块的寄存器中，读取指定位置的 bit 信息。
- 参数：
  - name - 模块名称，用于匹配 DTS 中的模块。
  - offset - 寄存器相当于基地址的偏移。
  - shift - 该 bit 在寄存器中的位移。
  - mask - 该 bit 的掩码值。
- 返回：

- 0: fail。
- other: 获取到的实际 bit 信息。



## 5 可测试性

### 5.1 sysfs 调试接口

1. `/sys/class/sunxi_info/sys_info` 此节点文件可以打印出一些 SoC 信息，包括版本信息、ChipID 等。

```
# cat /sys/class/sunxi_info/sys_info
sunxi_platform : sun50iw10p1           // 当前硬件平台信息
sunxi_secure   : secure                // 当前系统是否安全固件
sunxi_chipid   : 00000000000000000000000000000000 // 当前板卡对应的chipid
```

2. `/sys/class/sunxi_info/key_info` 此节点用于获取指定名称的 Key 信息。方法是先写入一个 Key 名称，然后就可以读取到 Key 的内容。

```
# echo chipid > /sys/class/sunxi_info/key_info
# cat /sys/class/sunxi_info/key_info
0xf1c1b200: 0x00000400
0xf1c1b204: 0x00000000
0xf1c1b208: 0x00000000
0xf1c1b20c: 0x00000000
```

### 5.2 动态调试接口

SID 驱动使用动态调试接口（比如 `pr_debug()`）来打印部分调试信息。若要查看这部分打印信息，请按如下步骤执行：1. 打开内核配置中的 `CONFIG_DYNAMIC_DEBUG` 宏。2. 在小机端，使能指定的 debug 信息：

```
# 挂载 debugfs:
mount -t debugfs none /sys/kernel/debug/

# 打开指定文件的动态 debug (+p 表示打开, -p 表示关闭) :
echo "file sunxi-sid.c +p" > /sys/kernel/debug/dynamic_debug/control

# 打开指定行的动态 debug:
echo "file sunxi-sid.c line 154 +p" > /sys/kernel/debug/dynamic_debug/control
```

### 5.3 应用层调用示例

应用层可通过设备节点 `/dev/sunxi_soc_info` 读取 efuse 信息，具体使用可参考下述 demo

表 5-1: ioctl 命令含义表

cmd	含义
CHECK_SOC_SECURE_ATTR	read soc secure attr informations
CHECK_SOC_VERSION	read soc version informations
CHECK_SOC_BONDING	read soc bonding informations
CHECK_SOC_CHIPID	read chipid informations
CHECK_SOC_CHIPID_ORIGIN	read origin chipid informations
CHECK_SOC_FT_ZONE	read ft_zone informations

```
enum {  
    CHECK_SOC_SECURE_ATTR,  
    CHECK_SOC_VERSION,  
    CHECK_SOC_BONDING = 3,  
    CHECK_SOC_CHIPID,  
    CHECK_SOC_FT_ZONE,  
    CHECK_SOC_CHIPID_ORIGIN = 8,  
};
```

```
//打开设备节点
```

```
fd = open("/dev/sunxi_soc_info",O_RDONLY);  
if (fd < 0){  
    printf("open /dev/sunxi_soc_info failed!");  
    return -1;  
}
```

```
//通过ioctl读取efuse信息  
ret = ioctl(fd,cmd,buf);
```

```
if (ret < 0) {  
    printf("ioctl err!\n");  
    close(fd);  
    return ret;  
}
```

## 6 FAQ

1. 若系统无法访问 efuse 信息, 应如何处理?

答案: 针对部分产品, 当启用安全系统后, Non-Secure 空间将无法访问大部分的 Efuse 信息, 这个时候需要通过 SMC 指令来读取这些 Key 信息。此时不能再使用普通的寄存器读接口, 而需要调用 SMC 接口: `int sunxi_smc_readl(phys_addr_t addr)`。目前, `sunxi_smc_readl()`的实现在`bsp/drivers/smc/sunxi-smc.c`。






## 著作权声明

版权所有 ©2025 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、、**全志科技**、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的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所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。