Linux LRADC
开发指南

版本号: 1.2
发布日期: 2022.1.16

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.7.27	XAA0248	初始版本
1.1	2022.11.27	XAA0248	兼容 Linux-5.10 及以上内核版本
1.2	2022.1.16	XAA0248	1. 修改 bsp 仓库文件路径 2. 完善 input 信息

目　录

1 前言 **1**

1.1 文档简介 1

1.2 目标读者 1

1.3 适用范围 1

1.4 相关术语介绍 1

2 模块介绍 **2**

2.1 模块功能 2

2.2 结构框图 3

2.3 模块配置 4

2.3.1 设备树配置 4

2.3.2 menuconfig 配置 4

2.4 模块源码结构 6

3 接口设计 **7**

3.1 内部接口 7

3.1.1 evdev_open 7

3.1.2 evdev_read 7

3.1.3 evdev_write 8

3.1.4 evdev_ioctl 8

3.2 外部接口 8

3.2.1 确认 LRADC 模块的 event 节点 8

3.2.2 读取 LRADC 模块的上报数据 9

3.2.3 查看 LRADC 模块的中断次数 10

4 模块使用范例 **11**

5 FAQ **13**

插　图

图 2-1　KEY 按键电路 2
图 2-2　KEY 模块结构框图 3
图 2-3　Allwinner BSP 4
图 2-4　Device Drivers 5
图 2-5　Low Rate ADC Drivers 5
图 2-6　Keyboard Support for Allwinner SoCs 6
图 3-1　hexdump 分析 10

1 前言

1.1 文档简介

介绍 LRADC 模块的使用方法，方便开发人员使用。

1.2 目标读者

LRADC 模块的驱动开发/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.10 及以上	sunxi-lradc.c

1.4 相关术语介绍

表 1-2: 术语介绍

术语	解释说明
sunxi	指 Allwinner 的一系列 soc 硬件平台
LRADC	全志平台使用的按键模块

2 模块介绍

2.1 模块功能

LRADC 模块属于 INPUT 输入设备，一般包括 VOL+、VOL-、HOME、MENU、ENTER 等等。Sunxi LRADC 模块的实际电路如下图所示：

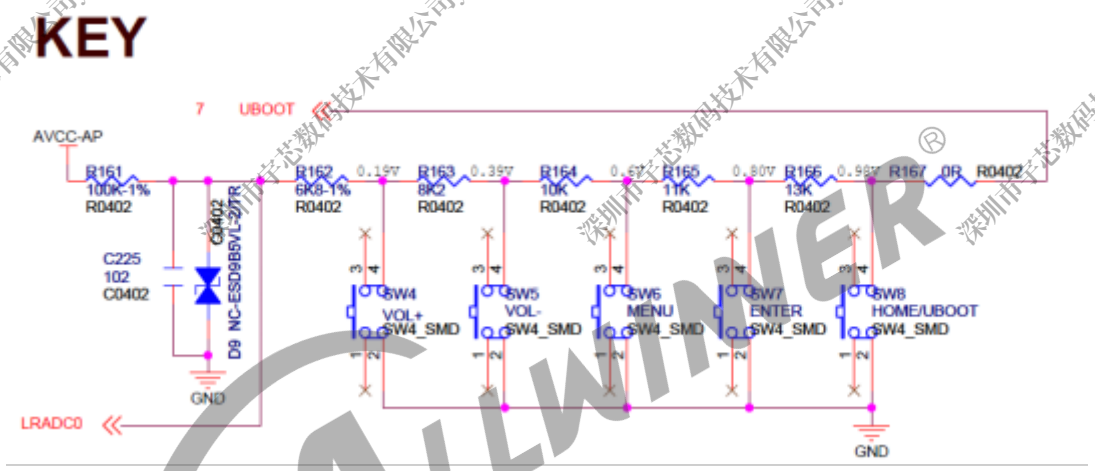

图 2-1: KEY 按键电路

AVCC-AP 为 1.8V 的供电，不同的按键按下，LRADC0 口的电压不同，CPU 通过对这个电压的采样来确定具体是哪一个按键按下。如上图，VOL+、VOL-、MENU、ENTER、HOME/UBOOT 对应的电压分别为 0.19V、0.39V、0.6V、0.80V、0.98V。

2.2 结构框图

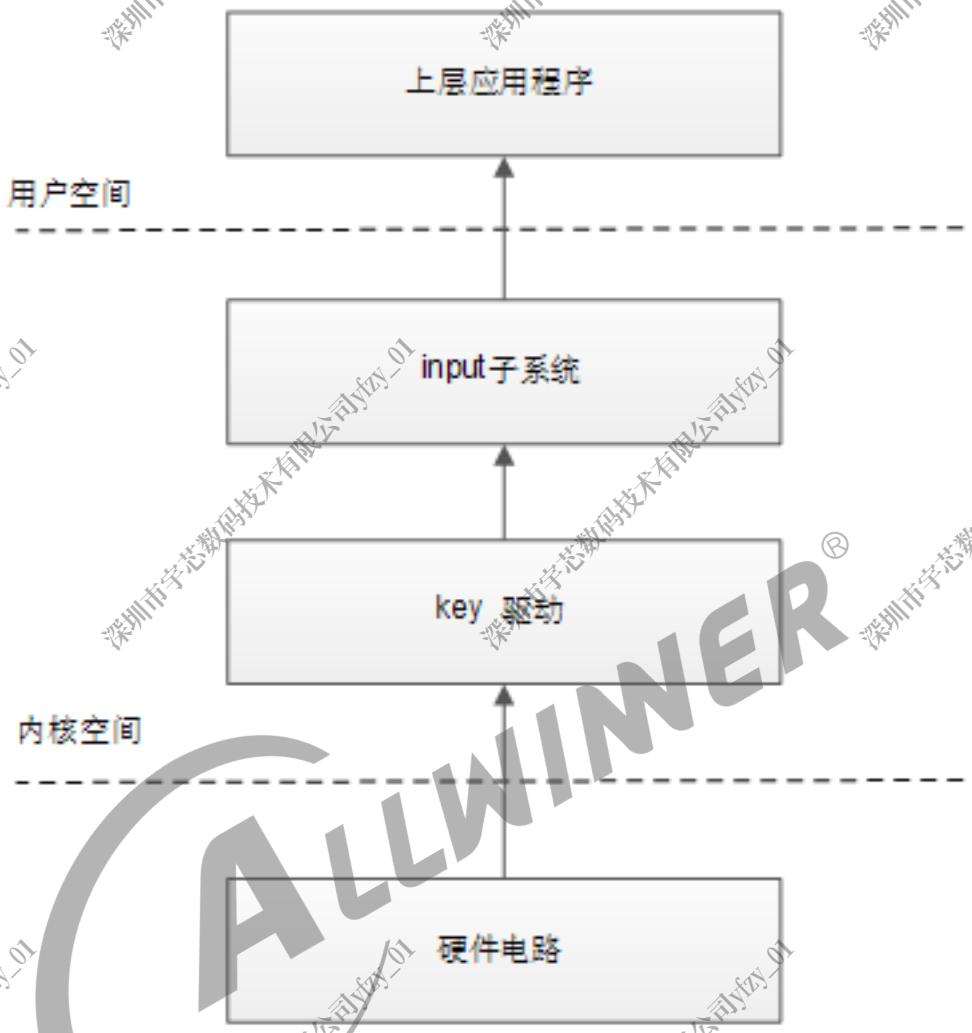

图 2-2: KEY 模块结构框图

整个系统框架流程如下所示：当用户按下按键的时候，会触发一个中断。这里的中断通过读取中断状态寄存器，判断是按键按下中断，还是数据中断，还是按键释放中断。KEY 按键驱动会进入中断，然后读取整个按键电路的电压值，然后对该电压值进行解码，然后将该事件上报给 INPUT 子系统。INPUT 子系统找到相应的事件处理程序之后，会将该按键事件上报给用户空间，等待用户程序对该按键信息的读取与处理。

2.3 模块配置

2.3.1 设备树配置

在 soc 级的 **dtsi** 文件中提炼了内存基地址、中断控制、时钟等共性信息，是该类芯片所有平台的模块配置，soc 级的 **dtsi** 文件的路径为：bsp/configs/{linux-ver}/{CHIP}.dtsi(CHIP 为研发代号，如 sun50iw10p1 等)，具体配置如下：

```
keyboard: keyboard@5070800 {
    compatible = "allwinner,keyboard_1350mv";
    reg = <0x0 0x05070800 0x0 0x400>;  /* 寄存器地址 */
    interrupts = <GIC_SPI 22 IRQ_TYPE_EDGE_RISING>; /* 中断号 + 中断类型 */
    clocks = <&ccu CLK_BUS_LRADC>;  /* clk时钟 */
    resets = <&ccu RST_BUS_LRADC>;  /* reset时钟 */
    key_cnt = <5>;    /*物理按键数量*/
    key0 = <210 115>;  /* 底下是按键电压和键值配置，左边210是电压，单位为mV，右边115为该电压对应的键值*/
    key1 = <410 114>;
    key2 = <590 139>;
    key3 = <750 28>;
    key4 = <880 102>;
  };
```

2.3.2 menuconfig 配置

在根目录中执行./build.sh menuconfig，选择 Allwinner BSP 选项进入下一级配置，如下图所示：

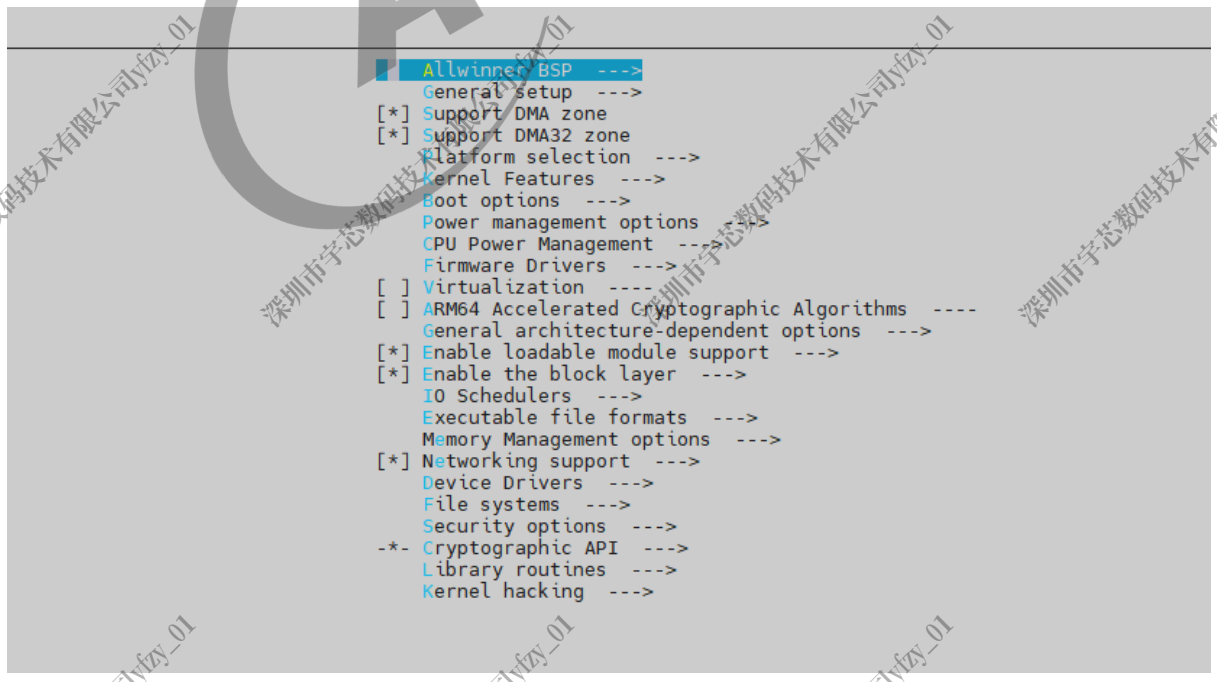

图 2-3: Allwinner BSP

选择 Device Drivers 选项进入下一级配置，如下图所示：

图 2-4: Device Drivers

选择 Low Rate ADC Drivers 选项，进入下一级配置，如下图所示：

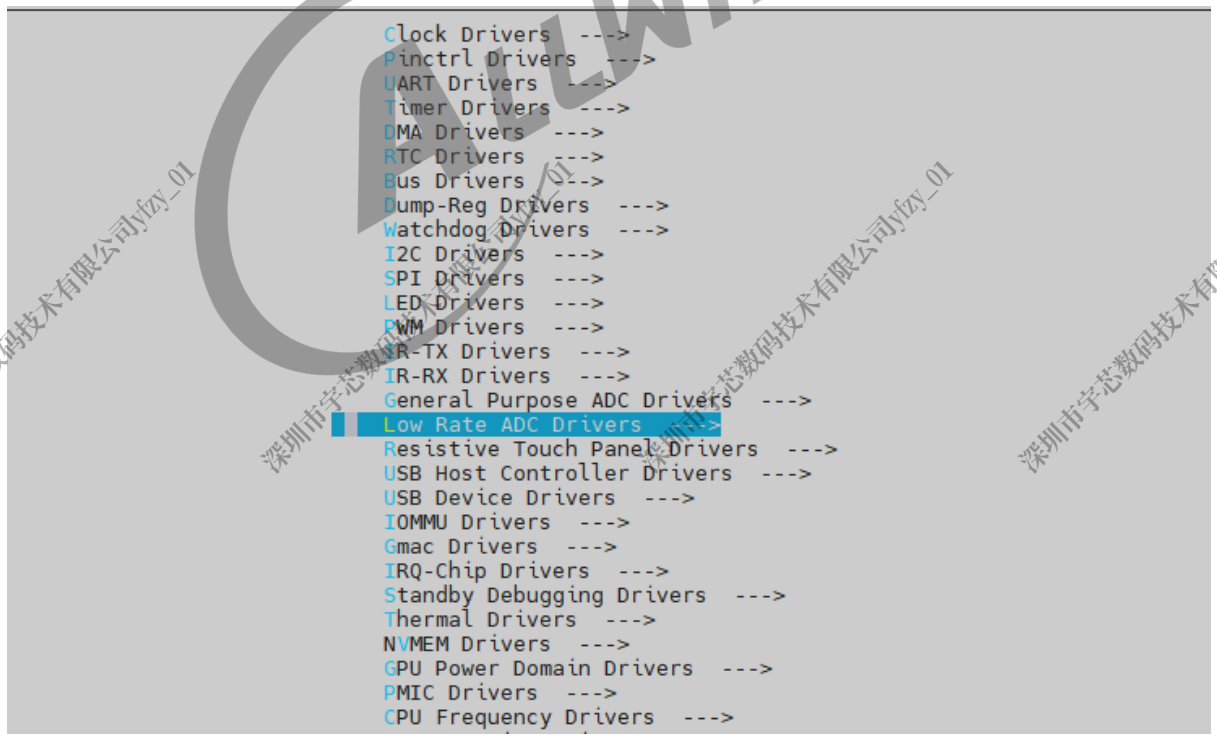

图 2-5: Low Rate ADC Drivers

选择 Keyboard Support for Allwinner SoCs 选项，可选择直接编译进内核，也可编译成模块。如下图所示：

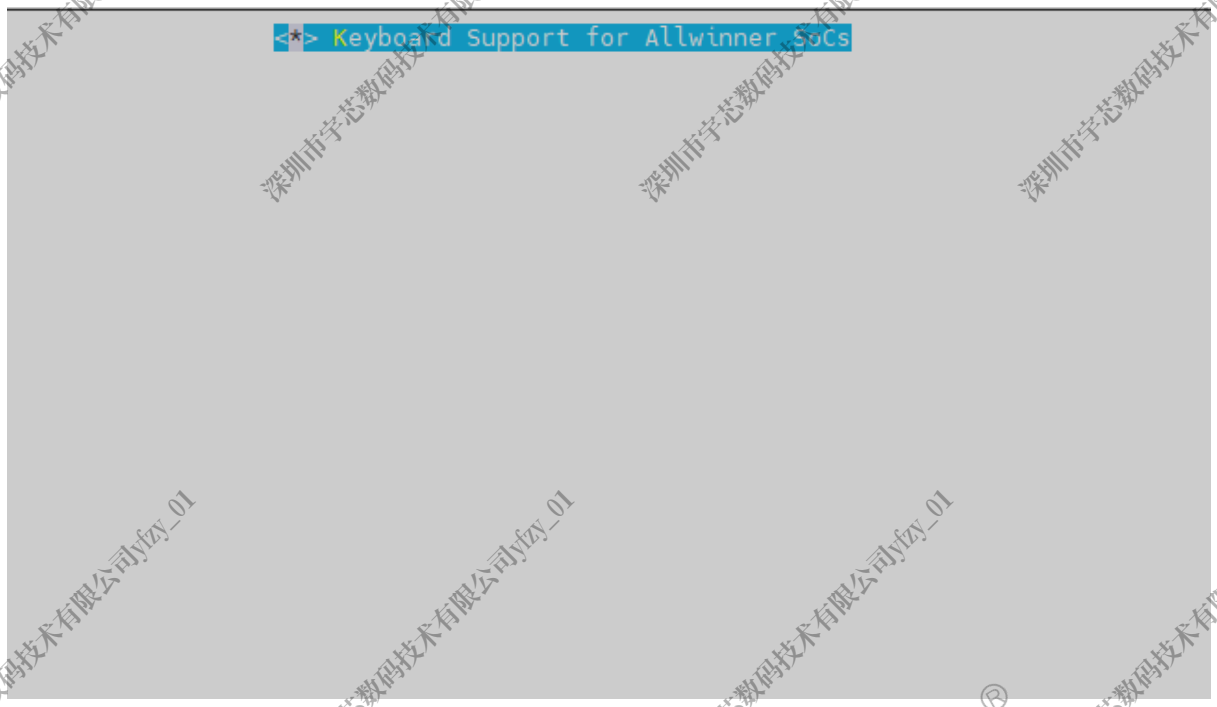

图 2-6: Keyboard Support for Allwinner SoCs

2.4 模块源码结构

KEY 模块的源码结构如下所示：

```
bsp/drivers/lradc/
├── sunxi-lradc.c
```

3 接口设计

3.1 内部接口

LRADC 模块在 Linux 内核中是作为字符设备使用，所以可以使用相关字符设备接口来对 LRADC 模块进行相应的读写和配置操作。相关定义在 evdev.c 文件里面。下面介绍几个比较有用的函数：

3.1.1 evdev_open

- 函数原型：static int evdev_open(struct inode *inode, struct file *file)。
- 功能描述：程序（C 语言等）使用 open(file) 时调用的函数。打开一个 LRADC 设备，可以像文件读写的方式往 LRADC 设备中读写数据。
- 参数说明：

（1）inode：inode 节点；

（2）file：file 结构体。

- 返回值：文件描述符。

3.1.2 evdev_read

- 函数原型：static ssize_t evdev_read(struct file *file, char __user buffer, size_t count, loff_t ppos)。
- 功能描述：程序（C 语言等）调用 read() 时调用的函数。读取 LRADC 模块上报事件数据。
- 参数说明：

（1）file，file 结构体；

（2）buf，写数据 buf；

（3）ppos，文件偏移。

- 返回值：成功返回读取的字节数，失败返回负数。

3.1.3 evdev_write

- 函数原型：static ssize_t evdev_write(struct file *file, const char __user buffer, size_t count, loff_t ppos)。
- 功能描述：程序（C 语言等）调用 write() 时调用的函数。像 LRADC 模块里面写入上报事件。
- 参数说明：

（1）file，file 结构体；

（2）buf，读数据 buf；

（3）ppos，文件偏移。

- 返回值：成功返回 0，失败返回负数。

3.1.4 evdev_ioctl

- 函数原型：static long evdev_ioctl(struct file *file, unsigned int cmd, unsigned long arg)。
- 功能描述：程序（C 语言等）调用 ioctl() 时调用的函数。可以控制相关的 LRADC 模块。
- 参数说明：

（1）file，file 结构体；

（2）cmd，指令；

（3）arg，其他参数。

- 返回值：成功返回 0，失败返回负数。

找到 LRADC 模块对应的 eventX(如 dev/input/event0) 文件，就可以使用 C 语言的文件读写，控制函数来调用上述的接口。

3.2 外部接口

3.2.1 确认 LRADC 模块的 event 节点

在内核中，查看 /proc/bus/input/devices，确认 LRADC 的数据上报节点。

```
/ # cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="sunxi-keyboard"    //keyboard对应的name,根据此信息确认是按键对应的event
P: Phys=sunxikbd/input0
S: Sysfs=/devices/virtual/input/input0
U: Uniq=
H: Handlers=kbd event0  //生成的event节点
B: PROP=0
B: EV=3
B: KEY=800 c0040 0 0 10000000
```

3.2.2 读取 LRADC 模块的上报数据

直接在内核中 hexdump 相应的 event 节点，当按下按键后 LRADC 模块采集到数据的时候，可以看到 event 节点上报的数据。

```
root@TinaLinux:/# hexdump /dev/input/event0
[  487.638954] key down
0000000 7b86 000a 0000 0000 e0d6 0006 0000 0000
0000010 0001 0073 0001 0000 7b86 000a 0000 0000
0000020 e0d6 0006 0000 0000 0000 0000 0000 0000
[  487.792679] key up
0000030 7b86 000a 0000 0000 0b84 0009 0000 0000
0000040 0001 0073 0000 0000 7b86 000a 0000 0000
0000050 0b84 0009 0000 0000 0000 0000 0000 0000
```

当按下按键时，触发按键中断，先后调用 input_report_key 和 input_sync 上报 EV_KEY 和 EV_SYN 类型的事件，并将按键对应的设备树的键值 115（0x73）发送到 event0 节点；松开按键时再次调用 input_report_key 和 input_sync，这两个函数都会调用 input_event，其原型为：

```
void input_event(struct input_dev *dev, unsigned int type, unsigned int code, int value);
```

因此，有如下分析：

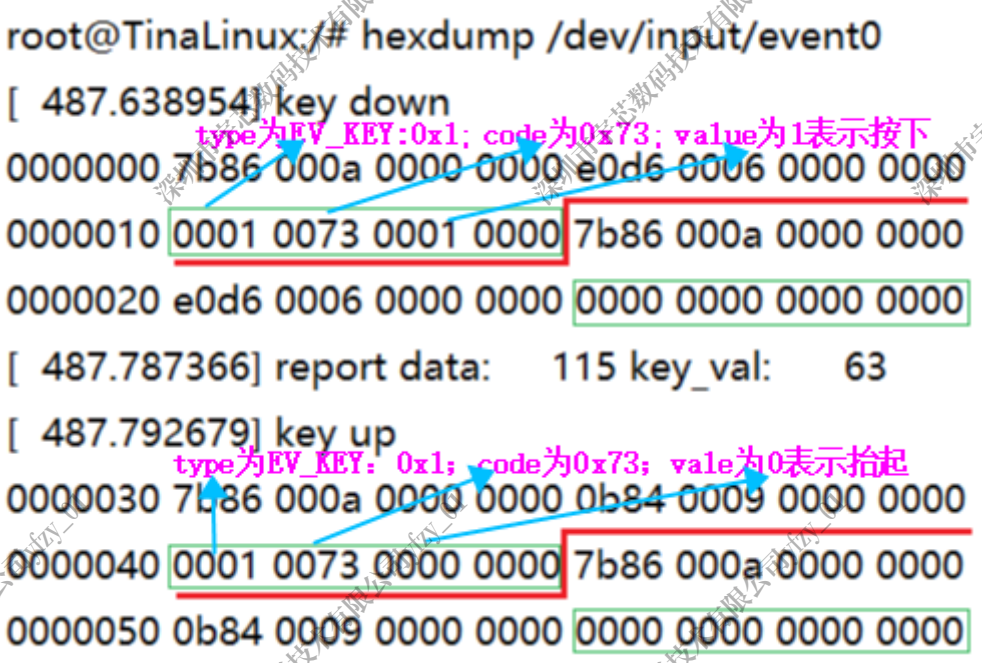

图 3-1: hexdump 分析

input_event 发送给 event0 节点的数据分为四次，数据部分为绿色框选区域，第 1 个 16 位表示类型，第 2 个 16 位表示按键键值，第 3 个 16 位表示传入的 value 值。

input_value 结构体的定义如下：

```
/**
 * struct input_value - input value representation
 * @type: type of value (EV_KEY, EV_ABS, etc)
 * @code: the value code
 * @value: the value
 */
struct input_value {
__u16 type;
__u16 code;
__s32 value;
};
```

3.2.3 查看 LRADC 模块的中断次数

查看 /proc/interrupts 可以查看相关的 LRADC 模块中断次数。

```
/ # cat /proc/interrupts
       CPU0       CPU1       CPU2       CPU3
 67:        280          0          0          0  wakeupgen  34 Level     sunxi-mmc2
 68:          0          0          0          0  wakeupgen  32 Level     sunxi-mmc0
 69:         12          0          0          0  wakeupgen  33 Level     sunxi-mmc1
 85:          0          0          0          0  wakeupgen  31 Edge      sunxikbd   //sunxi keyboard
```

4 模块使用范例

为了演示 LRADC 模块的使用，下面将演示用 C 语言对 LRADC 模块上报的数据进行读写：

```
#include <stdio.h>
#include <linux/input.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <limits.h>
#include <unistd.h>
#include <signal.h>

#define DEV_PATH "/dev/input/event0"   //difference is possible
const int key_exit = 102;
static int keys_fd = 0;

unsigned int test_keyboard(const char * event_file)
{
   int code = 0,i;
   struct input_event data;

   keys_fd = open(DEV_PATH, O_RDONLY);

   if(keys_fd <= 0)
   {
      printf("open %s error!\n", DEV_PATH);
      return -1;
   }

   for(i = 0;i < 10;i++)
   {
      read(keys_fd, &data, sizeof(data));

      if(data.type == EV_KEY && data.value == 1)
      {
         printf("key %d pressed\n", data.code);
      }
      else if(data.type == EV_KEY && data.value == 0)
      {
         printf("key %d releaseed\n", data.code);
      }
   }

   close(keys_fd);
   return 0;
}

int main(int argc,const char *argv[])
{
   int rang_low = 0, rang_high = 0;
```

```
    return test_keyboard(DEV_PATH);
}
```

该 Demo 用来读取 LRADC 模块用于 KEY 的按键上报事件（其他类似）。其循环 10 次读取按键上报事件输入，并且显示出相应按键的值。

5 FAQ

无

著作权声明

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





（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。