Linux GPADC
开发指南

版本号: 1.9
发布日期: 2025.6.23

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.7.27	XAA0248	添加初版
1.1	2023.02.16	XAA0312	1. 根据最新模块刷新文档 2. 增加细节描述
1.2	2023.3.15	XAA0312	更新细节描述
1.3	2023.12.19	XAA0311	根据最新代码优化文档
1.4	2024.8.22	AWA2215	增加 MR536 平台 dts 配置说明
1.5	2024.8.30	AWA2215	增加 T536 平台 dts 配置说明
1.6	2024.10.22	AWA2215	1. 增加 Linux-5.4 内核版本支持 2. 优化文档结构
1.7	2024.11.25	AWA2215	1. 增加 Linux-6.6 内核版本支持
1.8	2025.2.20	XAA0331	1. 根据最新代码优化文档
1.9	2025.6.23	AWA2231	1. 增加 adc 模式的 dts 配置说明 2. 增加 gpadc 启动介质和 ddr 参数识别说明

目　录

插　图

表　格

1 前言

1.1 文档简介

介绍 GPADC 模块的使用方法，方便开发人员使用。

1.2 目标读者

GPADC 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	{SDK}/bsp/drivers/gpadc/sunxi_gpadc.c
Linux-5.15	{SDK}/bsp/drivers/gpadc/sunxi_gpadc.c
Linux-5.4	{SDK}/bsp/drivers/gpadc/sunxi_gpadc.c
Linux-6.6	{SDK}/bsp/drivers/gpadc/sunxi_gpadc.c

1.4 文档约定

1.4.1 标志说明

⚠ 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。

技巧

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：

表 1-2: 地址与数据描述方法约定

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：

表 1-3: 数值单位约定

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

1.5 相关术语介绍

1.5.1 硬件术语

表 1-4: 硬件术语

术语	解释说明
GPADC	高精度模数转换

1.5.2 软件术语

表 1-5: 软件术语

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
分辨率	分辨率越高，能够采集到更小的电压，是衡量 ADC 采样准确度的间接标准。
采样率	指每秒进行 AD 转换的次数，采样率越高，对信号的还原度越高。
SDK	指 longan 或 tina 的根目录

2 模块介绍

2.1 模块功能介绍

GPADC 是 12bit 分辨率，10 位采集精度的模数转换模块，具体精度和通道数可以参考芯片手册，模拟输入范围 0～1.8V，最高采样率 1MHz，并且支持数据比较，自校验功能，同时工作于可配置以下工作模式：

1. Single mode：在指定的通道完成一次转换并将数据放在对应数据寄存器中；
2. Single-cycle mode：在指定的通道完成一个周期转换并将数据放在响应数据寄存器中 (注：该模式在 R528 中没有)；
3. Continuous mode：在指定的通道持续转换并将数据放在响应数据寄存器中；
4. Burst mode：边采样边转换并将数据放入 32 字节的 FIFO，支持中断控制。

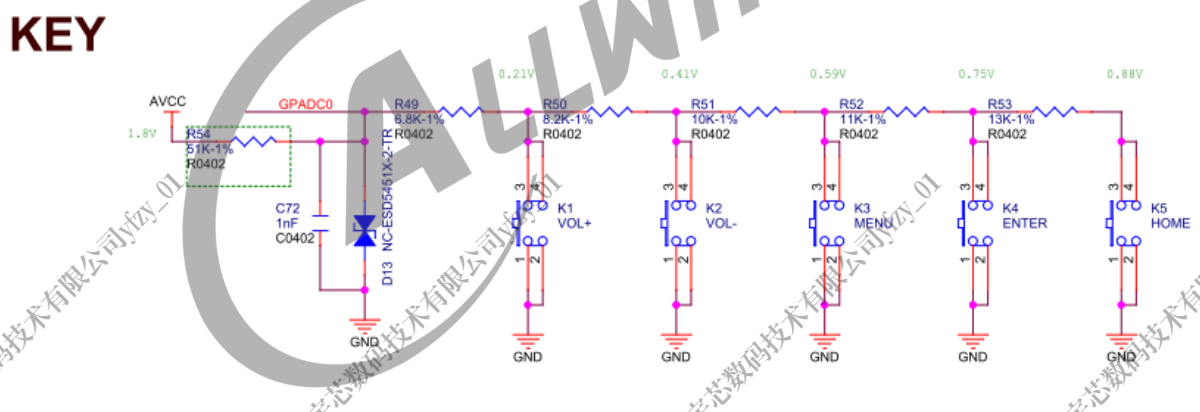

图 2-1: GPADC0

部分 GPADC 接口也开始慢慢用于 KEY 模块按键的读取，一般包括 VOL+、VOL-、HOME、MENU、ENTER 等等，GPADC0 用于 KEY 的电路如上图。

AVCC-AP 为 1.8V 的供电，不同的按键按下，GPADC0 口的电压不同，CPU 通过对这个电压的采样来确定具体是那一个按键按下。如上图，VOL+、VOL-、MENU、ENTER、HOME/UBOOT 对应的电压分别为 0.21V、0.41V、0.59V、0.75V、0.88V。

2.2 软件框架介绍

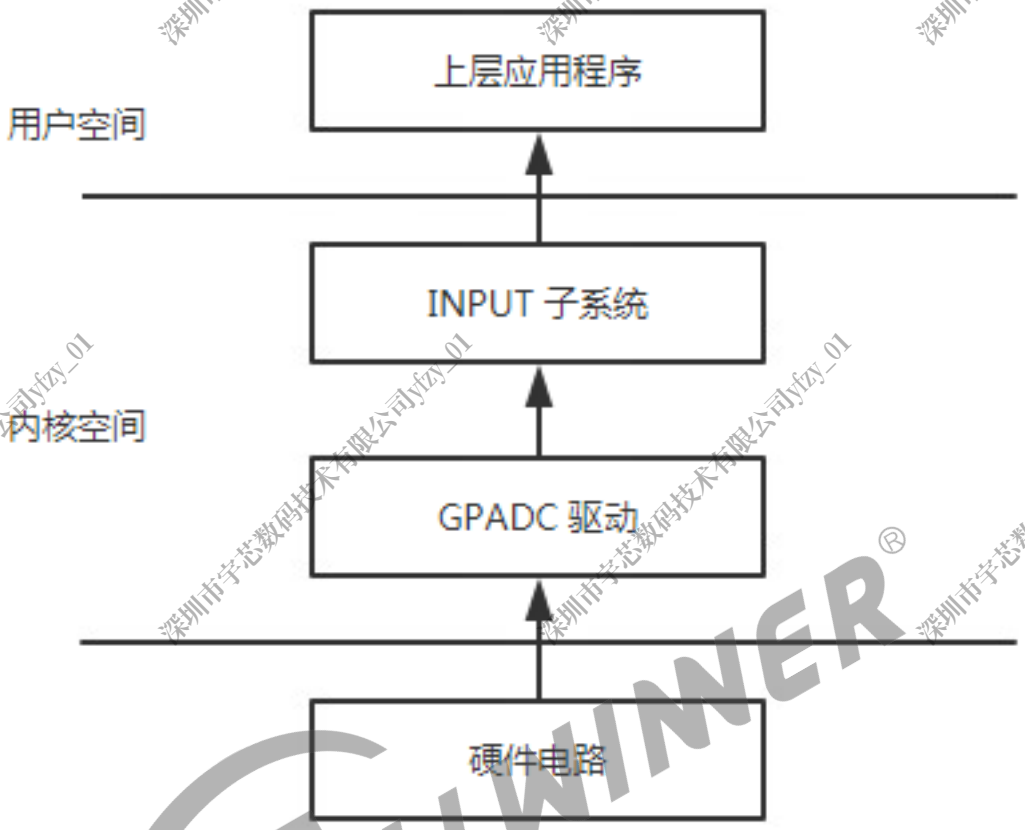

图 2-2: 结构框图

当 GPADC 模块中断触发之后，驱动会对数据进行采集。采集到的数据转换成相应的键值后通过 input 子系统将数据上传到/dev/input/event 节点，应用程序可从相应的节点获取数据。

2.3 模块配置介绍

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：sdk/bsp/configs/{linux-ver}/{CHIP}.dtsi(CHIP 为研发代号, 如 sun50iw10p1 等) 下面为 gpadc0 配置：

```
gpadc0: gpadc0@2009000 {
    compatible = "allwinner,sunxi-gpadc";
    reg = <0x0 0x02009000 0x0 0x400>;
    interrupts = <GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_BUS_GPADC0>;
    clock-names = "bus";
```

```
    resets = <&ccu RST_BUS_GPADC0>;
    status = "okay";
};
```

2.3.2 keypad 配置

gpadc 驱动支持多个 keypad 功能，支持最大 keypad 个数为 channel_num 数，具体配置哪几个通道为 keypad 功能如下 gpadc0:gpadc 节点所示，keypad 配置多个按键功能见 keyadc0 和 keyadc1 配置

若要配置 GPADC 为 keypad 功能，需要在 sdk/device/config/chips/{IC}/config /{BOARD}/{linux-x}/board.dts 里面配置相关参数，参考如下：

```
/*
*channel_num:  Max number of channels supported on the platform.
*channel_select:  channel enable slection. channel0:0x01  channel1:0x02 channel2:0x04 channel3:0x08
*channel_data_select: channel data enable.  channel0:0x01  channel1:0x02 channel2:0x04 channel3:0x08.
*channel_compare_select:  compare function enable channel0:0x01
*          channel1:0x02 channel2:0x04 channel3:0x08.
*channel_cld_select: compare function low data enable setection: channel0:0x01
*          channel1:0x02 channel2:0x04 channel3:0x08.
*channel_chd_select: compare function hig data enable setection: channel0:0x01
*          channel1:0x02 channel2:0x04 channel3:0x08.
*/
    gpadc0:gpadc{
     channel_num = <10>;              //最大通道数（即最大keypad数）
     channel_select = <0x3>;          //通道选择 （选择0 1 通道）
     channel_data_select = <0x3>;     //使能通道数据
     channel_compare_select = <0x3>;  //使用通道比较功能
     channel_cld_select = <0x3>;      //使用数据小于比较功能
     channel_chd_select = <0x3>;      //使用数据大于比较功能
     channel0_compare_lowdata = <0>;     //channel0数据大于该值触发中断，设置最小电压为0mV
     channel0_compare_higdata = <1200000>;  //channel0数据小于该值触发中断，设置最大电压为1200mV
     channel1_compare_lowdata = <100000>;  //channel1数据大于该值触发中断，设置最小电压为100mV
     channel1_compare_higdata = <1700000>;  //channel1数据小于该值触发中断，设置最大电压为1700mV
     status = "okay";
     //支持多个adc通道作按键，按键配置方式如下：
     keyadc0 {
          key_cnt = <5>; //按键个数
          key0_vol = <7>; //按键按下时电压值，单位为毫伏
          key0_val = <78>;//按键上报的值
          key1_vol = <240>;
          key1_val = <114>;
          key2_vol = <360>;
          key2_val = <139>;
          key3_vol = <480>;
          key3_val = <28>;
          key4_vol = <600>;
          key4_val = <102>;
     };
     keyadc1 {
          key_cnt = <5>;
          key0_vol = <115>;
          key0_val = <115>;
          key1_vol = <240>;
          key1_val = <114>;
```

```
        key2_vol = <360>;
        key2_val = <139>;
        key3_vol = <480>;
        key3_val = <28>;
        key4_vol = <600>;
        key4_val = <102>;
    };
    ..................
};
```

注意：

1. channel_compare_lowdata、channel_compare_higdata、keyadc 中的数字编号必须和 channel_select 对应如 channel_select = 0x300 即选择通道 8 9 为 keypad，相应上述编号必须对应为 channel8_compare_lowdata、channel8_compare_higdata、keyadc8、keyadc9
2. 当 gpadc 通道用作按键，那么此通道的 data_select 不能打开，因为打开 data 中断，会一直上报
3. 按键不按时，默认电压应该是 1800mV。如设置通道 0 的比较电压最大值为 1200mV，最小值为 0mV，配置通道 0 低于 1200mV，高于 0mV 时会产生中断

2.3.3 adc 配置

若要配置 GPADC 为 adc 模式，需要在 sdk/device/config/chips/{IC}/config /{BOARD}/{linux-x}/board.dts 里面配置相关参数，参考如下：

```
/*
*启用的通道及其配置：
*通道0-模式：ADC
*通道1-模式：ADC
*/
&gpadc0{
    channel_num = <10>;                 //最大通道数（即最大keypad数）
    channel_select = <0x3>;             //通道选择 （选择0 1 通道）
    channel_data_select = <0x3>;        //使能通道数据，打开data中断，会一直上报
    channel_compare_select = <0x3>;     //使用通道比较功能
    channel_cld_select = <0x3>;         //使用数据小于比较功能
    channel_chd_select = <0x3>;         //使用数据大于比较功能
    channel0_compare_lowdata = <1700000>; //channel0数据小于该值触发中断
    channel0_compare_higdata = <200000>;  //channel0数据大于该值触发中断
    channel1_compare_lowdata = <1700000>; //channel1数据小于该值触发中断
    channel1_compare_higdata = <200000>;  //channel1数据大于该值触发中断
    status = "okay";
}
```

注意：adc 模式只需要把 channel_select 这个属性，对应 bit 置 1 就行, 如果需要用作按键，才去添加 keyadcX 节点和配置其它的属性

2.3.4 工作模式配置

GPADC 支持不同工作模式的配置，支持模式如下：

- **单次模式**：GPADC 在这种模式下只进行一次采样，然后停止。
- **单周期模式**：GPADC 在每个采样周期内进行一次采样。
- **连续模式**：GPADC 连续的进行采样。
- **突发模式**：GPADC 在短时间内快速进行多次采样，然后停止。

默认 GPADC 采用连续模式。

若要配置 GPADC 为不同工作模式，需要在 sdk/device/config/chips/{IC}/config/{BOARD}/{linux-x}/board.dts 里面配置相关参数，参考如下：

```
&gpadc0 {
    gpadc_mode_select = <0x2>;
    status = "okay";
};
```

gpadc_mode_select 配置值参考如下:

工作模式	gpadc_mode_select 值
单次模式	0x0
单周期模式	0x1
连续模式	0x2
突发模式	0x3

2.3.5 pinmux 配置

适用范围

Table: 适用产品列表

- MR536
- T536

若 gpadc 支持 pinmux，则需要在 dts 中进行 pinmux 配置：

```
&pio {
  ..............
  gpadc0_pins_default: gpadc0@0 {
    pins = "PA0", "PA1", "PA2", "PA3", "PA4", "PA5", "PA6", "PA7", "PA8", "PA9";
    function = "gpadc0";
  };
```

```
    gpadc0_pins_sleep: gpadc0@1 {
        pins = "PA0", "PA1", "PA2", "PA3", "PA4", "PA5", "PA6", "PA7", "PA8", "PA9";
        function = "gpio_in";
    };
    ..............
};

&gpadc0 {
    ..............
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&gpadc0_pins_default>;
    pinctrl-1 = <&gpadc0_pins_sleep>;
    ..............
};
```

2.3.6 only used for gpadc

适用范围

Table: 适用产品列表

- MR536
- T536

部分平台存在 gpadc 和 tpadc 共用通道情况，例如 MR536 gpadc1 channel0-3 与 tpadc channel0-3 共用

若需配置通道仅用作 gpadc 功能，则需在 dts 添加 only_used_for_gpadc 属性：

```
&gpadc1 {
    channel_num = <10>; /* the value range is 4 to 10 */
    only_used_for_gpadc; /* gapdc1 channel0~3 only used for gpadc, can't open is when tpadc used */
    ..............
};
```

添加 only_used_for_gpadc 属性后，需要禁用 RTP 功能，在 dts 中将 rtp 的 status 设置为 disabled：

```
&rtp {
    allwinner,tp-sensitive-adjust = <0x2>;
    allwinner,filter-type = <0x3>;
    status = "disabled";
};
```

2.3.7 kernel 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：

```
    Allwinner BSP  --->
    General setup  --->
[*] Support DMA zone
[*] Support DMA32 zone
    Platform selection  --->
    Kernel Features  --->
    Boot options  --->
    Power management options  --->
    CPU Power Management  --->
    Firmware Drivers  --->
[ ] Virtualization  ----
[ ] ARM64 Accelerated Cryptographic Algorithms  ----
    General architecture-dependent options  --->
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    IO Schedulers  --->
    Executable file formats  --->
    Memory Management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Security options  --->
-*- Cryptographic API  --->
    Library routines  --->
    Kernel hacking  --->
```

图 2-3: Allwinner BSP

选择 Device Drivers 选项进入下一级配置，如下图所示：

```
    Platform Selection  --->
    Device Drivers  --->
    NAND Drivers  --->
    GPU Drivers  --->
```

图 2-4: Device Drivers

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

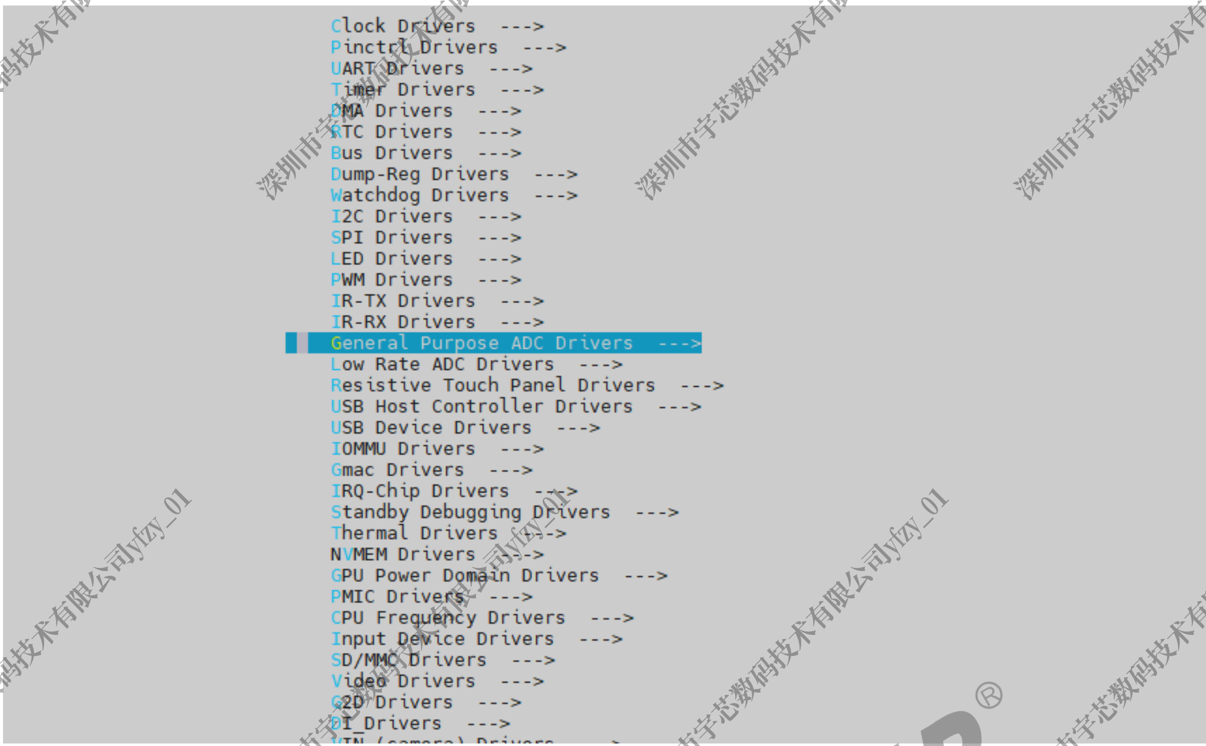

图 2-5: General Purpose ADC Drivers

选择 GPADC 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：

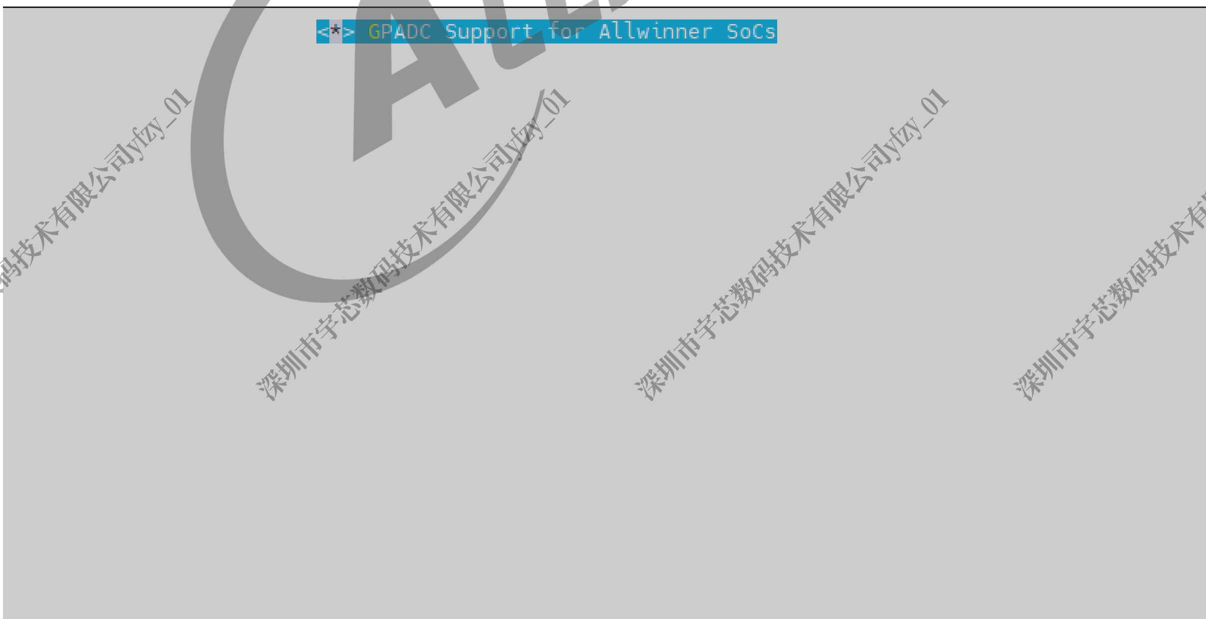

图 2-6: GPADC Support for Allwinner SoCs

2.4 源码结构

GPADC 驱动的源代码位于内核在 drivers/gpadc 目录下，具体的路径如下所示：

```
drivers/gpadc/
    ├──── sunxi_gpadc.c  // Sunxi平台的GPADC驱动代码
```

3 模块接口说明

3.1 内部接口

GPADC 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 GPADC 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 gpadc 设备，可以像文件读写的方式往 gpadc 设备中读写数据;
- 参数说明：
  - inode：inode 节点
  - 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。像往文件里面读数据一样从 gpadc 设备中读数据。底层调用 gpadc_xfer 传输数据;
- 参数说明：
  - file: file 结构体
  - buf: 写数据 buf
  - 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。像往文件里面写数据一样往 gpadc 设备中写数据。底层调用 gpadc_xfer 传输数据;
- 参数说明：
  - file: file 结构体
  - buf: 读数据 buf
  - ppos: 文件偏移
- 返回值：
  - 0: 成功
  - 负数：失败

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。像对文件管理 i/o 一样对 gpadc 设备管理。该功能比较强大，可以修改 gpadc 设备的地址，往 i2 设备里面读写数据，使用 smbus 等等，详细的可以查阅该函数;
- 参数说明：
  - file: file 结构体
  - cmd: 指令
  - arg: 其他参数
- 返回值：
  - 0: 成功
  - 负数：失败

找到 GPADC 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

- 函数原型：u32 sunxi_gpadc_read_channel_data(u32 controller_num, u8 channel)；
- 功能描述：获取某个 gpadc 控制器具体通道的电压值
- 参数说明：
  - controller_num：具体的 gpadc 控制器
  - channel：具体的 gpadc 控制器的通道
- 返回值：
  - 获取到的电压

4 GPADC 功能开发

4.1 功能概述

GPADC 功能主要用于获取输入的电压，可用于电压获取、按键等场景。

4.2 开发流程

步骤 1：调用 open 打开文件路径，获取文件描述符。

```
gpadc_fd = open(/dev/input/event3, O_RDONLY);
```

步骤 2：调用 read 读取 gpadc 数据。

```
read(gpadc_fd, &data, sizeof(data));
```

步骤 3：判断上报事件

如果是按键，调用以下接口:

```
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);
}
```

如果是 adc，调用以下接口:

```
if(data.type == EV_MSC)
{
   printf("adc data %d\n", data.value); //接收到的数据
   printf("adc vol: %d mv\n", 18000 * data.value / 4096 ); //电压值，单位mv
}
```

步骤 4：使用完毕后，关闭设备。

```
close(gpadc_fd);
```

4.3 注意事项

- 确保 open 的路径正确，确认 event 编号正确。
- 进行电压换算时，确认参考电压，通常是 1.8V。

4.4 编程示例

下面是一个用来读取 GPADC 的按键输入的一个 Demo。代码如下：

```
#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/event3"  /* 查看devices节点下Handlers时间编号，不同gpadc修改event编号，如event1 */
const int key_exit = 102;
static int gpadc_fd = 0;

unsigned int test_gpadc(const char * event_file)
{
  int code = 0, i;

  struct input_event data;

  gpadc_fd = open(event_file, O_RDONLY);

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

  for(i = 0; i < 10; i++) /* 读10次 */
  {
    read(gpadc_fd, &data, sizeof(data));
    /* 如果是按键，调用用以下接口 */
    if(data.type == EV_KEY && data.value == 1) /* 按键按下判断 */
    {
      printf("key %d pressed\n", data.code);
    }
    else if(data.type == EV_KEY && data.value == 0) /* 按键释放判断 */
    {
      printf("key %d releaseed\n", data.code);
    }
    /* 如果是adc，调用以下接口 */
    if(data.type == EV_MSC)
    {
      printf("adc data %d\n", data.value); /* 接收到的数据 */
```

```
            printf("adc vol: %d mv\n", 18000 * data.value / 4096 ); /* 电压值，单位mv */
        }
    }

    close(gpadc_fd);
    return 0;
}

int main(int argc,const char *argv[])
{
    test_gpadc(DEV_PATH);
    return 0;
}
```

该 Demo 用来读取 GPADC 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 调试方法

5.1 EVENT 接口

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

```
/ # cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="sunxi-gpadc0/channel1/input0"
P: Phys=sunxi-gpadc0/channel1/input0
S: Sysfs=/devices/platform/soc@3000000/2009000.gpadc0/input/input3
U: Uniq=
H: Handlers=event3
B: PROP=0
B: EV=11
B: MSC=10
```

然后直接在内核中 hexdump 相应的 event 节点，当 GPADC 模块采集到数据的时候，可以看到 GPADC 模块上报的数据。

注：不同的板子 event 节点的顺序不同。

```
/ # hexdump /dev/input/event3
0000000 bcc6 0000 3dbd 0009 0001 008b 0001 0000
0000010 bcc6 0000 3dbd 0009 0000 0000 0000 0000
0000020 bcc6 0000 0e1d 000b 0001 008b 0000 0000
0000030 bcc6 0000 0e1d 000b 0000 0000 0000 0000
```

5.2 调试节点

gpadc_chipX 代表的是某套控制器的资源, 如查看 gpadc 的第一套资源 gpadc_chip0, 内核进入 /sys/class/gpadc/gpadc_chip0 目录下面，可调试 GPADC 的相关状态。

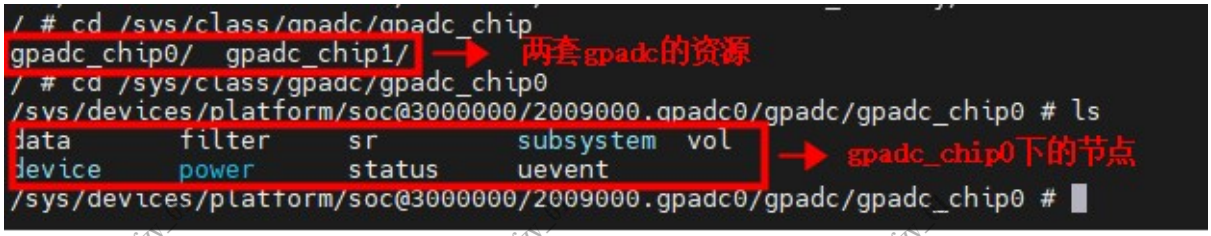

图 5-1: GPADC_CHIP 资源

5.3 gpadc 通道开关

```
echo gpadc0,0 > status #关闭gpadc0，',后可以为0或1， 0表示关闭，1表示开启；
cat status #查看gpadc各通道开关状态；
```

5.4 gpadc 采样率

```
echo 5000 > sr #设置gpadc采样率为10000，gpadc采样率范围为400~100000；
cat sr #查看gpadc当前采样率。
```

5.5 按键电压值

```
echo vol0,125 > vol #设置gpad按键0的采样电压为125；
cat vol #查看所有按键的采样电压值与按键索引映射；![gpadc识别](figures/adc识别.png)
```

5.6 滤波阈值

```
echo 6 > filter #设置滤波阈值为6；
cat filter #查看滤波阈值。
```

6 GPADC 识别功能

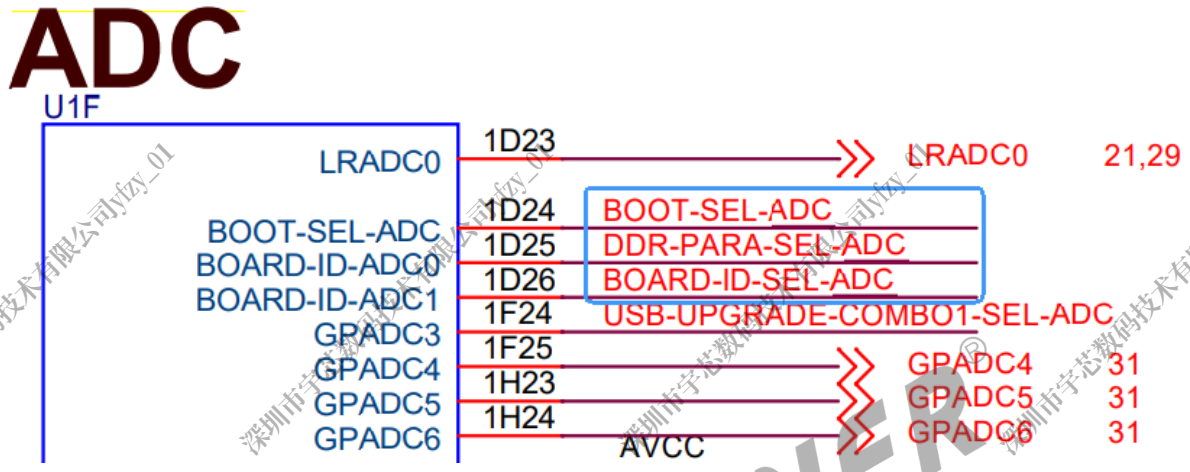

图 6-1: gpadc 识别

其中 BOOT-SEL-ADC、DDR-PARA-SEL-ADC、BOARD-ID-SEL-ADC 代表启动介质识别、DDR 参数识别、板级外设识别

6.1 GPADC 启动介质识别

6.1.1 功能介绍

通过 GPADC 选择启动介质 (SD 卡、eMMC、NAND) 并进行启动，系统上电后，BROM 会读取 BOOT_MODE 引脚的状态，以决定使用哪种方式来选择启动介质。BOOT_MODE 的配置可以通过 eFuse 映射设置，在 eFuse 配置中它被称为 BROM_Config。当从 GPADC 所选的启动介质启动失败时，BROM 会按如下顺序尝试其他启动介质：

eMMC_USR → eMMC_BOOT → SLC_NAND → MLC_NAND → SPI NOR → SPI NAND → UFS

其中，当系统使用 GPADC 启动方式（由 BOOT_MODE=0 触发）时，根据采集到的模拟电压转换成的数字值（KEY_VALUE），选择不同的启动介质顺序。

6.2 GPADC DDR 参数识别

6.2.1 功能介绍

通过 GPADC 选择不同的 ddr 参数，主要原理是 boot0 启动以后通过 gpadc 读取的电压值来判断选择使用哪一套 ddr 参数

6.2.2 功能配置

以 A733 为例，sys_config.fex 文件的路径为：device/config/chips/a733/configs/evb1/sys_config.fex, gpadc ddr para-select 的配置如下

```
;*****************************************************************************
;
;dram select configuration
;
;select_mode  :    dram模式选择，0:不进行自动识别
;                  1:gpio识别模式(dram_para, dram_para1-15, 共16组有效)
;                  2:gpadc识别模式(dram_para, dram_para1-7, 共8组有效)
;                  3:2个IO+gpadc识别模式(dram_para, dram_para1-32, 共33组有效)。其中IO配置优先级按select_gpio0>
      select_gpio1>select_gpio2>select_gpio3
;gpadc_channel :    选择gpadc通道  有效值(0-3)
;select_gpio1-4 :   选择gpio pin
;*****************************************************************************

[dram_select_para]
select_mode   = 2
gpadc_channel = 1
```

6.3 GPADC 板级外设识别

6.3.0.1 功能介绍

用于识别不同的屏幕参数，不同的外设地址。

7 FAQ

7.1 按键出现误报问题

常见的问题为按键出现误报问题，是由于硬件抖动过于频繁导致的，可以把滤波次数调大，调到一个可以接受的范围，然后到 sunxi_gpadc.c 下修改 filter_cnt 数值。

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。