Linux GPADC
开发指南

版本号: 1.1
发布日期: 2021.05.10

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.03.02	AWA1637	添加初版
1.1	2021.05.10	AWA1637	添加 R528 的说明

目　录

插　图

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-4.9	drivers/input/sensor/sunxi_gpadc.c
Linux-5.4	drivers/input/sensor/sunxi_gpadc.c

2 模块介绍

2.1 模块功能介绍

GPADC 是 12bit 分辨率，8 位采集精度的模数转换模块，具体通道数可以查看对应的 spec 说明文档，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，同时工作于可配置一下工作模式：

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。

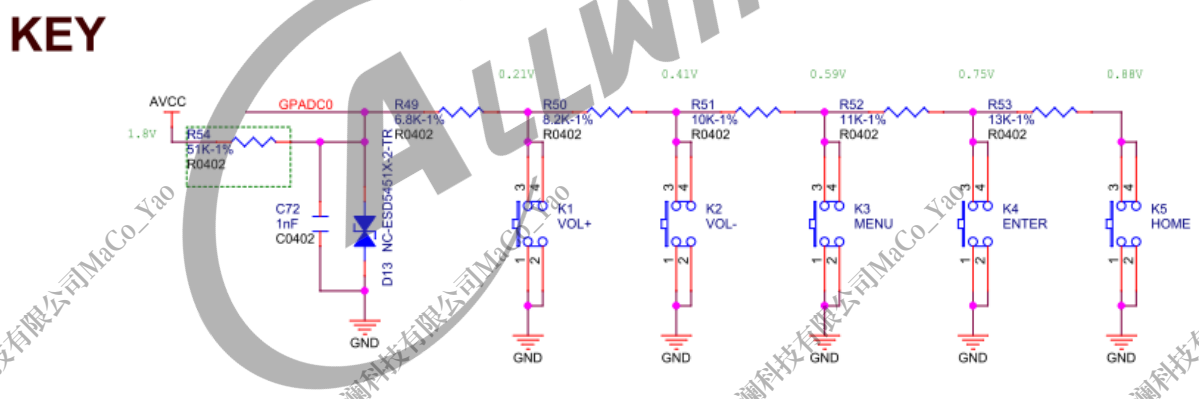

图 2-1: GPADC0

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/U-BOOT 对应的电压分别为 0.21V、0.41V、0.59V、0.75V、0.88V。

2.2 结构框图

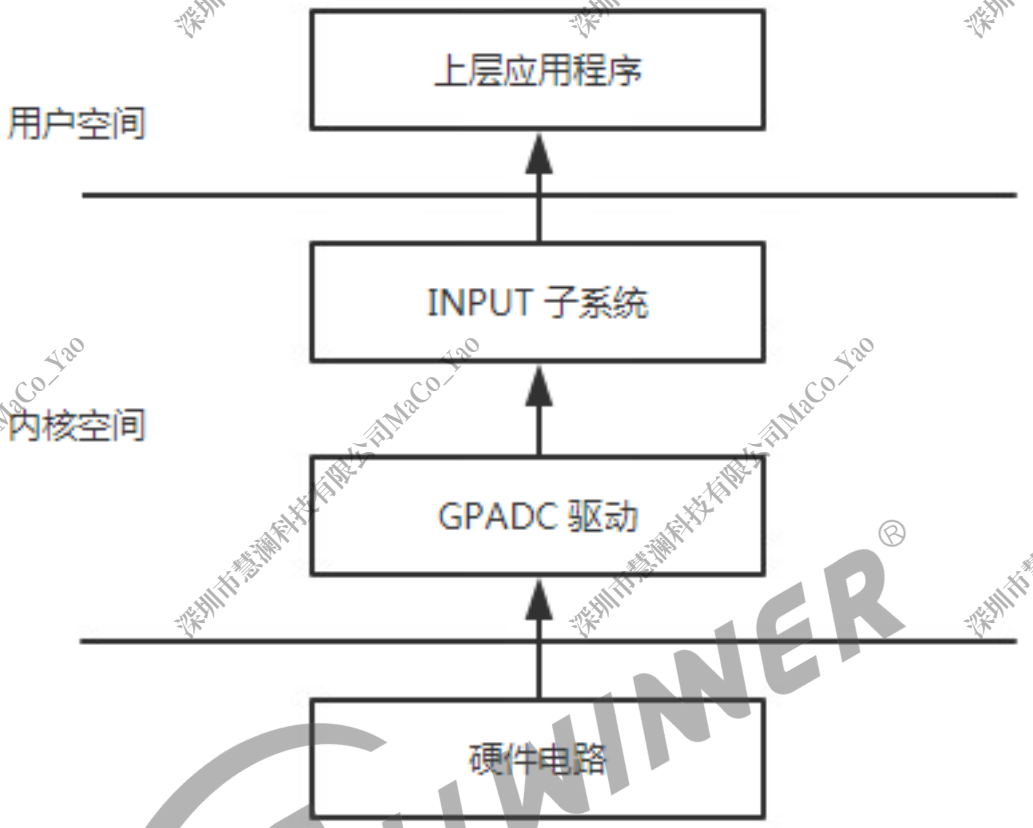

图 2-2: 结构框图

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 相关术语介绍

表 2-1: 术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台
GPADC	高精度模数转换

2.4 模块配置介绍

2.4.1 设备树配置

GPADC 模块的设备树配置位于 longan 的内核目录，如 arch/arm/boot/dts/xxx.dtsi，64 位为：arch/arm64/boot/dts/sunxi/xxx.dtsi，下面为一个配置：

```
    gpadc:gpadc{
        compatible = "allwinner,sunxi-gpadc";   //表征具体的设备，用于驱动和设备的绑定;
        reg = <0x0 0x05070000 0x0 0x400>;       //设备使用的寄存器地址;
        interrupts = <GIC_SPI 0 IRQ_TYPE_NONE>; //设备使用的中断配置;
        clocks = <&clk_gpadc>;          //设备使用的时钟配置
        status = "disabled";            //设备是否使能
    };
```

若要配置 GPADC 相关的采样功能以及按键功能，需要在 longan/device/config/chips/{IC}/config/{BOARD}/{linux-x}/board.dts 里面配置相关参数：

```
/*
*channel_num:   Maxinum 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.
*/
    gpadc:gpadc{
        channel_num = <2>;          //使用的通道
        channel_select = <0x05>;        //通道选择
        channel_data_select = <0>;      //使能通道数据
        channel_compare_select = <0x05>;    //使用通道比较功能
        channel_cld_select = <0x05>;        //使用数据小于比较功能
        channel_chd_select = <0>;       //使用数据大于比较功能
        channel0_compare_lowdata = <1700000>;   //数据小于该值触发中断
        channel0_compare_higdata = <1200000>;   //数据大于该值触发中断
        channel1_compare_lowdata = <460000>;
        channel1_compare_higdata = <1200000>;
        channel0_compare_lowdata = <1500000>;   //数据小于该值触发中断
        channel0_compare_higdata = <1200000>;   //数据大于该值触发中断
        key_cnt = <5>;              //按键数量，注:以下部分只有GPADC0用于按键会用到
        key0_vol = <210>;           //按键电压
        key0_val = <115>;           //按键上报的值
        key1_vol = <410>;
        key1_val = <114>;
        key2_vol = <590>;
        key2_val = <139>;
        key3_vol = <750>;
        key3_val = <28>;
        key4_vol = <880>;
        key4_val = <102>;
```

```
        status = "okay";
    };
```

部分老的平台采用 sys_config.fex，新平台基本不做使用。

路径为 longan/device/config/chips/{IC}/config/{BOARD}/sys_config.fex，相关参数如下：

```
;------------------------------------------------------------------------------
;resistance gpadc configuration
;channel_num:   Maxinum number of channels supported on the platform.
;channel_select:   channel enable setection. 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 channel2:0
    x04 channel3:0x08.
;channel_cld_select:  compare function low data enable setection: channel0:0x01  channel1:0
    x02 channel2:0x04 channel3:0x08.
;channel_chd_select:  compare function hig data enable setection: channel0:0x01  channel1:0
    x02 channel2:0x04 channel3:0x08.
;------------------------------------------------------------------------------
[gpadc]
gpadc_used              = 1
channel_num             = 1
channel_select          = 0x01
channel_data_select     = 0
channel_compare_select  = 0x01
channel_cld_select      = 0x01
channel_chd_select      = 0x01
channel0_compare_lowdata    = 1700000
channel0_compare_higdata    = 1200000
gpadc_sample_rate       = 1000
key_cnt                 = 2
key0_vol                = 90
key0_val                = 115
key1_vol                = 260
key1_val                = 114
```

以上三个配置文件优先级依次为 sys_config.fex 最高，board.dts 次之，最后为内核下面的 dtsi 配置。

2.4.2 menuconfig 配置

linux-4.9 在命令行中进入内核根目录，执行 make ARCH=arm menuconfig（64 位系统采用 make ARCH=arm64 menuconfig）进入配置主界面，并按以下步骤操作。

- 首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

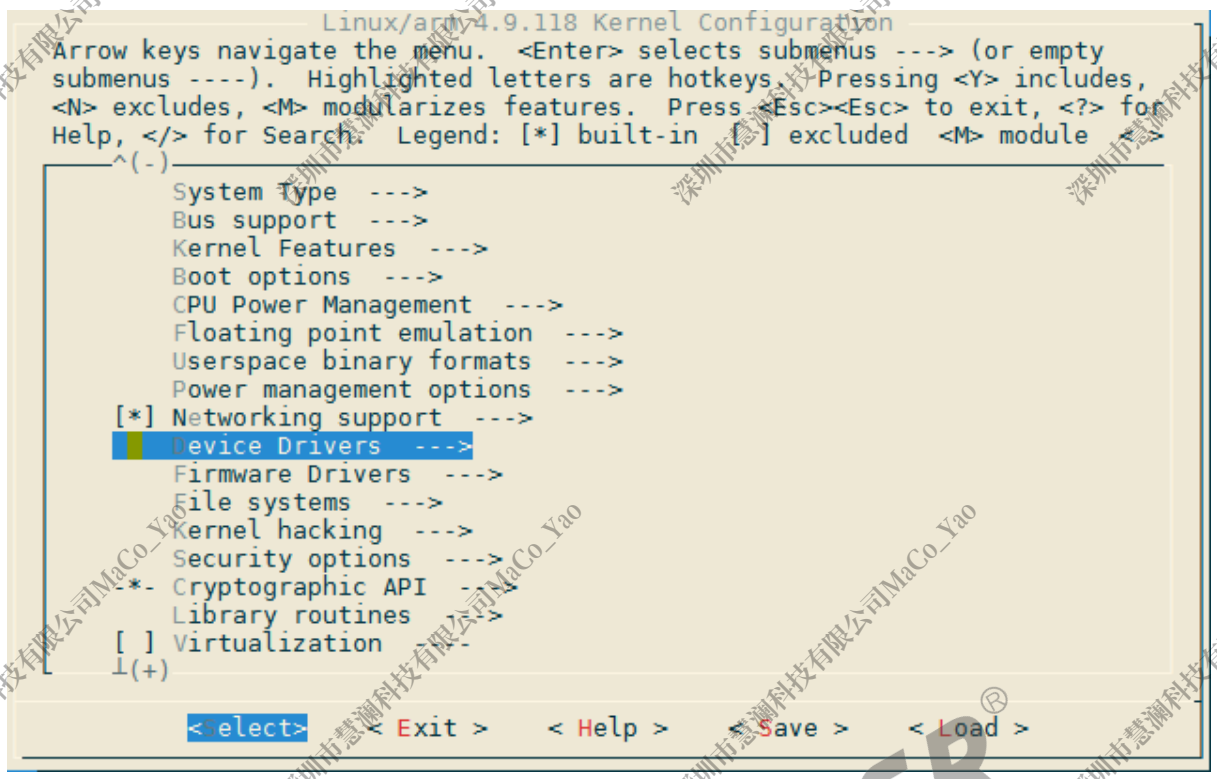

图 2-3: Device Drivers

- 然后，选择 Input device support 选项，进入下一级配置，如下图所示：

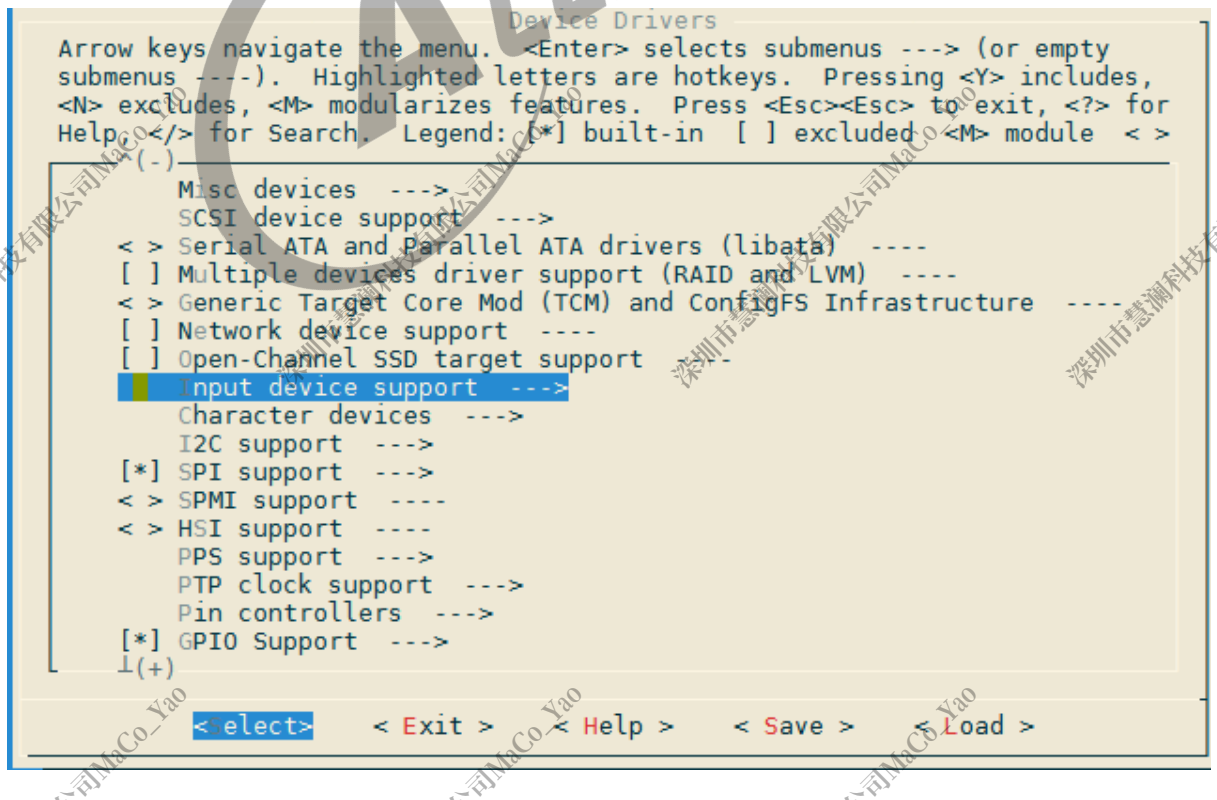

图 2-4: Input device support

- 接着，选择 Sensors 选项，进入下一级配置，如下图：

图 2-5: Sensors

- 选择 SUNXI GPADC 选项，可选择直接编译进内核，也可编译成模块。如下图：

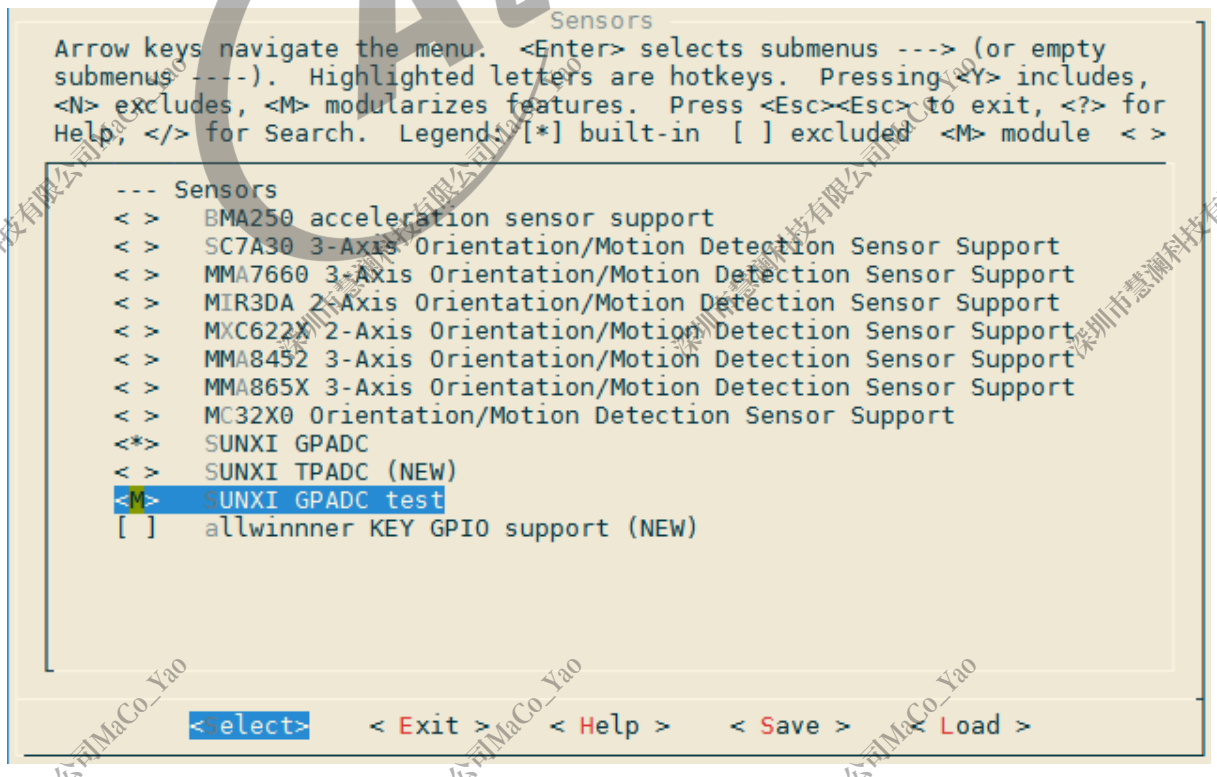

图 2-6: SUNXI GPADC

linux-5.4 在 longan 根目录中执行./build.sh menuconfig

- 首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

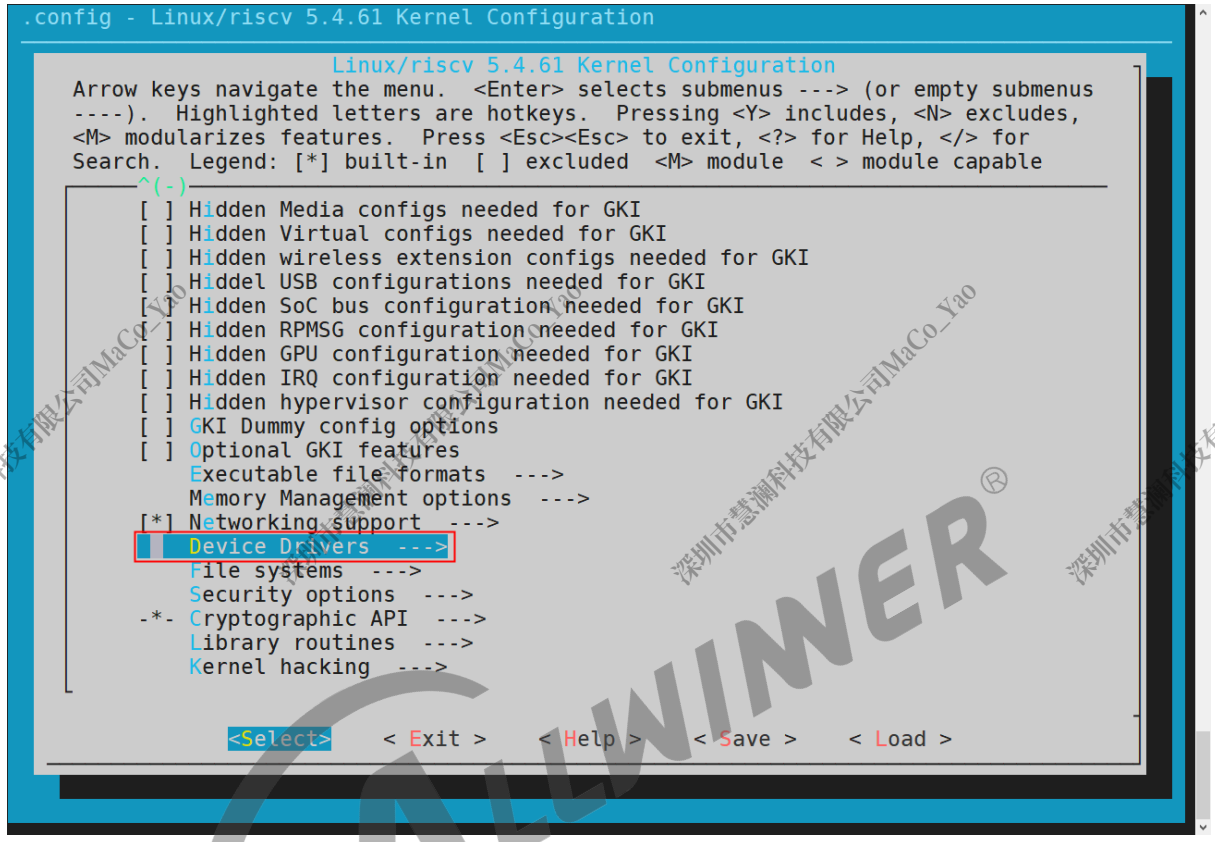

图 2-7: Device Drivers

- 然后，选择 Input device support 选项，进入下一级配置，如下图所示：

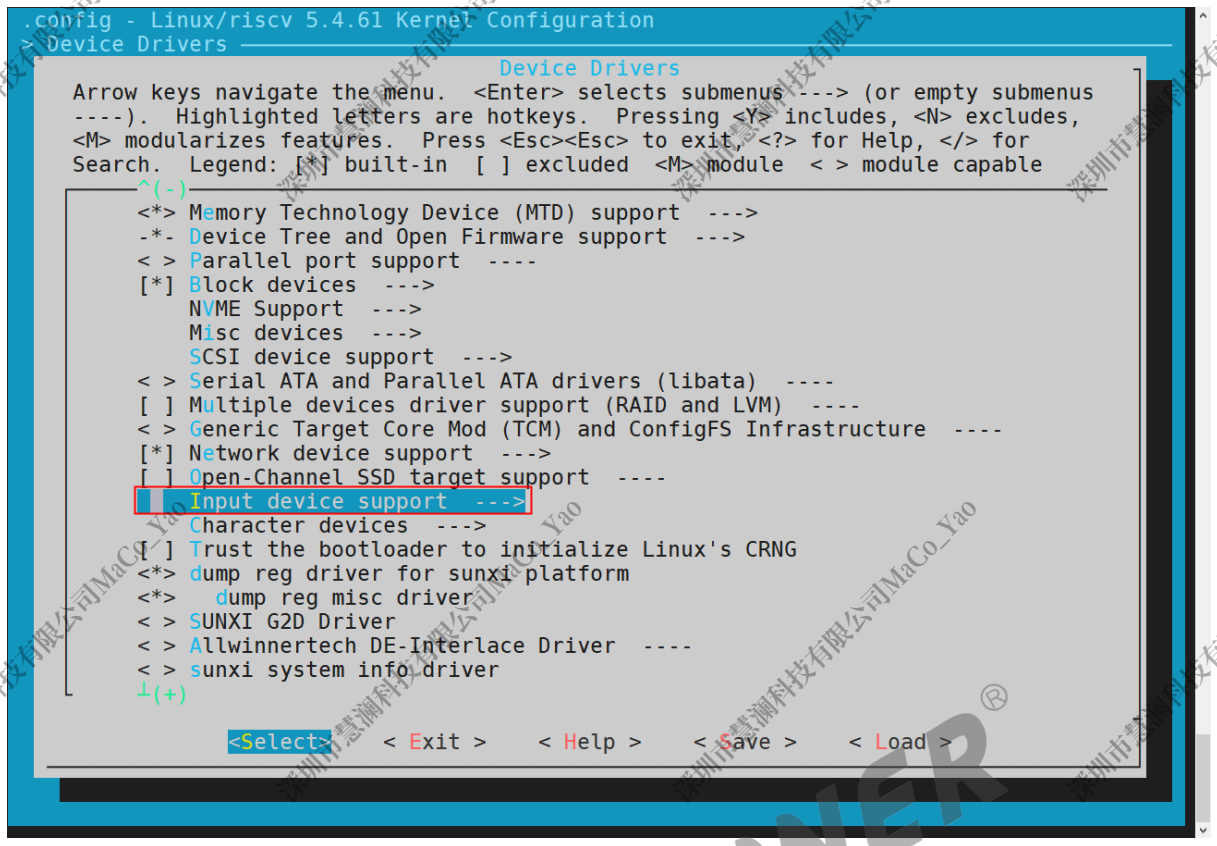

图 2-8: Device Drivers

- 接着，选择 Sensors 选项，进入下一级配置，如下图：

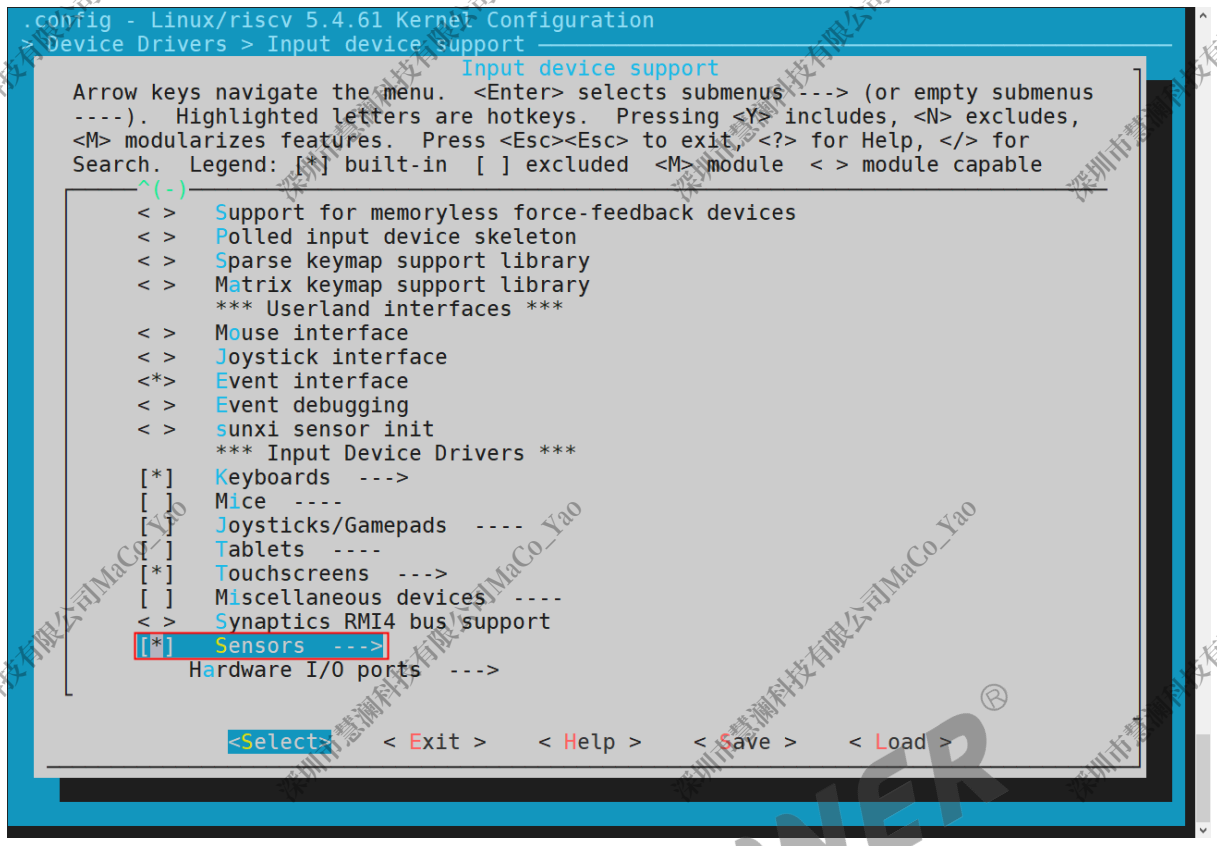

图 2-9: Device Drivers

- 选择 SUNXI GPADC 选项，可选择直接编译进内核，也可编译成模块。如下图：

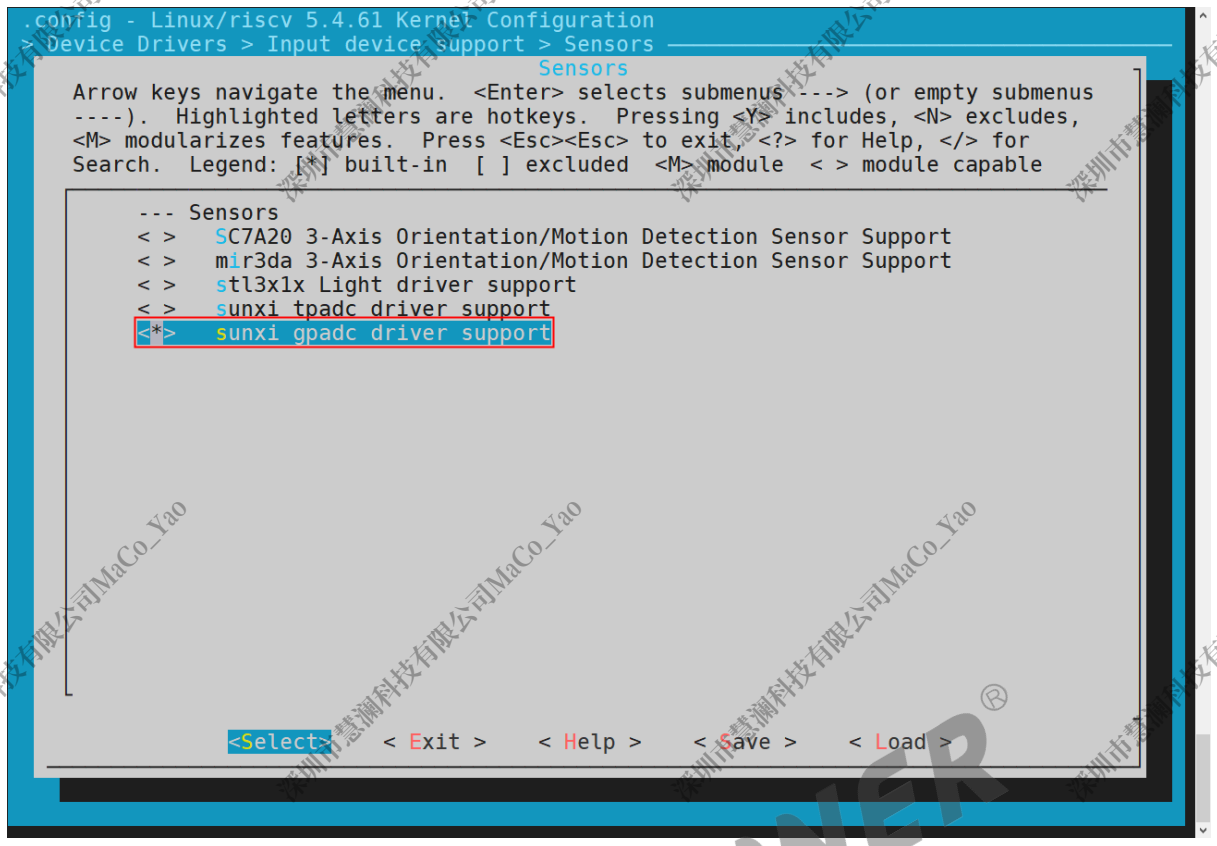

图 2-10: Device Drivers

2.5 源代码结构介绍

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

```
drivers/input/
├── sensor
│   ├── sunxi_gpadc.c  // Sunxi平台的GPADC驱动代码
│   ├── sunxi_gpadc.h  // 为Sunxi平台的GPADC驱动定义了一些宏、数据结构
```

3 接口设计

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 i2c 设备，可以像文件读写的方式往 i2c 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 *buf, 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 i2c 设备中读数据。底层调用 i2c_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_read(struct file *file, const char __user *buf,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 i2c 设备中写数据。底层调用 i2c_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：成功返回 0，失败返回负数。

3.1.4 evdev_ioctl()

- 函数原型：static long evdev_read(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 i2c 设备管理。该功能比较强大，可以修改 i2c 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：成功返回 0，失败返回负数。

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

在内核中，查看 /proc/bus/input/devices，确认 GPADC 的数据上报节点，看下面的 Handlers 属性。

```
/ # cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="sunxi-gpadc0"
//如果作为按键使用，N: Name="sunxi-keyboard"
P: Phys=sunxikbd/input0
S: Sysfs=/devices/virtual/input/input0
U: Uniq=
H: Handlers=kbd event0
```

```
B: PROP=0
B: EV=3
B: KEY=800 c0040 0 0 10000000
```

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。

```
/ # hexdump /dev/input/event0
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
```

4 模块使用范例

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/event0"   //查看3.2章节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(DEV_PATH, O_RDONLY);

    if(gpadc_fd <= 0)
    {
        printf("open %s error!\n", DEV_PATH);
        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)
        {
            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.data); //接收到的数据
            printf("adc vol:%d\n", 18000 * data / 4096 ); //电压值，单位mv
        }
    }
```

```
    close(gpadc_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 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 FAQ

5.1 调试方法

5.1.1 调试节点

内核进入 /sys/class/gpadc 目录下面，可调试 GPADC 的相关状态。想要使用下面功能，最好到 sunxi_gpadc.c 下，修改 debug_mask 为 3，然后把系统打印等级调高，如：echo 8 > /proc/sys/kernel/printk。

5.1.2 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;
```

5.1.3 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。
```

5.1.4 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;
```

5.1.5 滤波阈值

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

5.2 常见问题

5.2.1 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


、 
、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。