



Unisoc Confidential For hiar

Kernel 4.14 GPIO 客制化指导手册

文档版本
发布日期

V1.2
2020-08-10

版权所有 © 紫光展锐科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。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。

Unisoc Confidential For hiar

紫光展锐科技有限公司



前言

概述

本文主要介绍基于 Kernel4.14 的 GPIO 配置以及调试方法。

读者对象


本文档主要适用于需要进行 GPIO 配置的客户。

缩略语

| 缩略语 | 英文全名 | 中文解释 |
|------|-------------------------------|-------------|
| GPIO | General Purpose Input/ Output | 通用目的输入/输出端口 |
| DTS | Device Tree Source | 设备树源码 |

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

| 符号 | 说明 |
|--|---|
|  说明 | 用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。 |

变更信息

| 文档版本 | 发布日期 | 修改说明 |
|------|------------|---|
| V1.0 | 2019-08-01 | 初稿。 |
| V1.1 | 2020-03-18 | 文档名修改、格式更新。 |
| V1.2 | 2020-08-10 | <ul style="list-style-type: none">更正 GPIO 路径增加 GPIO 编号说明 |

| 文档版本 | 发布日期 | 修改说明 |
|------|------|---|
| | | <ul style="list-style-type: none">• 示例代码采用可编辑文字代替原来的截图• 更新注意事项 |

关键字

GPIO、DTS、API、config。

Unisoc Confidential For hiar

目 录

| | |
|---------------------------------|---|
| 1 概览 | 1 |
| 1.1 GPIO 简介 | 1 |
| 1.2 环境配置 | 1 |
| 2 GPIO 配置与使用 | 2 |
| 2.1 驱动代码路径及 API 接口 | 2 |
| 2.1.1 驱动代码路径 | 2 |
| 2.1.2 常用 API 接口 | 2 |
| 2.1.3 GPIO 编号 | 2 |
| 2.1.4 使用实例 | 3 |
| 2.2 客制化配置 | 4 |
| 2.2.1 编译配置 | 4 |
| 2.2.2 DTS 配置 | 4 |
| 2.2.3 调试方法 | 4 |
| 3 注意事项 | 6 |
| 3.1 边沿触发与电平触发的区别 | 6 |
| 3.2 DeepSleep 时需要唤醒 AP 系统 | 6 |

图目录

| | |
|--------------------|---|
| 图 3-1 电平触发示意图..... | 6 |
|--------------------|---|

Unisoc Confidential For hiar

1 概览

1.1 GPIO 简介

GPIO (General Purpose Input/ Output, 通用目的输入/输出端口), 是一个灵活的软件控制的数字信号, 可以配置为输入或输出。当配置为输入模式时, 可以被编程来触发 ARM 中断。

1.2 环境配置

- 运行环境: Kernel4.14+Android (Android 9.0 以及之后的平台)
- 调试环境: Ubuntu14.04

Unisoc Confidential For hiar

2 GPIO 配置与使用

2.1 驱动代码路径及 API 接口

2.1.1 驱动代码路径

- Android9.0 平台对应的目录：
 - kernel4.14/drivers/gpio/gpiolib*.c
 - kernel4.14/drivers/gpio/gpio*sprd.c
- Android10.0 及之后的平台对应的目录：
 - bsp/kernel/kernel4.14/drivers/gpio/gpiolib*.c
 - bsp/kernel/kernel4.14/drivers/gpio/gpio*sprd.c

2.1.2 常用 API 接口

```
struct gpio_desc *gpio_to_desc(unsigned gpio);
int desc_to_gpio(const struct gpio_desc *desc);
int gpiod_get_direction(struct gpio_desc *desc);
int gpiod_request(struct gpio_desc *desc, const char *label);
void gpiod_free(struct gpio_desc *desc);
struct gpio_desc *gpio_to_desc(unsigned gpio);
int desc_to_gpio(const struct gpio_desc *desc);
int gpiod_get_direction(struct gpio_desc *desc);
int gpiod_request(struct gpio_desc *desc, const char *label);
void gpiod_free(struct gpio_desc *desc);
int gpiod_direction_input(struct gpio_desc *desc);
int gpiod_direction_output(struct gpio_desc *desc, int value);
int gpiod_set_debounce(struct gpio_desc *desc, unsigned debounce);
int gpiod_get_value(const struct gpio_desc *desc);
void gpiod_set_value(struct gpio_desc *desc, int value);
int gpiod_to_irq(const struct gpio_desc *desc);
```

2.1.3 GPIO 编号

Kernel GPIO Device 包含 GPIO、EIC、PMIC EIC 等驱动，在代码实际使用过程中会重新进行编号，查看编号的方法参考如下，以 SC9832E 为例：

```
sp9832e_1h10_go:/ # cat /sys/kernel/debug/gpio
gpiochip5: GPIOs 144-159, parent: platform/sc27xx-eic, sc27xx-eic:
gpio-144 (          |vbus          ) in hi IRQ
```



```
gpio-145 (          |Power Key          ) in hi IRQ
// GPIOs 160-415, 共256个, 说明是ap_gpio, 即: ap_gpio的编号=160+实际的GPIO ID
//如要DTS定义的是&ap_gpio 144 (实际的GPIO ID), 则在使用过程中会被编号为304
gpiochip4: GPIOs 160-415, parent: platform/40280000.gpio, 40280000.gpio:
  gpio-200 (          |power-down-gpios   ) in lo
  gpio-304 (          |adaptive_ts_int    ) in hi IRQ
  gpio-305 (          |adaptive_ts_rst    ) out hi
gpiochip3: GPIOs 416-439, parent: platform/402100c0.gpio, eic-sync:
gpiochip2: GPIOs 440-463, parent: platform/402100a0.gpio, eic-async:
gpiochip1: GPIOs 464-487, parent: platform/40210080.gpio, eic-latch:
gpiochip0: GPIOs 488-511, parent: platform/40210000.gpio, eic-debounce:
  gpio-490 (          |Volume Down Key    ) in hi IRQ
```

2.1.4 使用实例

```
struct gpio_desc *desc;
struct gpio_chip *chip;

desc = gpio_to_desc(num); //将GPIO编号转化为描述符, 注意: num并不是实际的GPIO ID
chip = gpiod_to_chip(desc);
if (!chip) {
    pr_err("get gpio chip failed.\n");
    return -EINVAL;
}

ret = gpiod_request(desc, "autotest-gpio"); //使能GPIO
if (ret < 0 && ret != -EBUSY) {
    pr_err("gpio request failed.\n");
    return ret;
}

if (dir) {
    ret = gpiod_direction_output(desc, val); //配置为输出
    if (ret < 0) {
        pr_err("set direction failed, %d", ret);
        return ret;
    }
} else {
    gpiod_direction_input(desc); //配置为输入
    val = gpiod_get_value(desc) ? 1 : 0; //获取外部电平是高或低
```

2.2 客制化配置

2.2.1 编译配置

```
config GPIO_SPRD
    tristate "Spreadtrum GPIO support"
    depends on GPIOLIB && ARCH_SPRD
    help
        Say yes here to support Spreadtrum's GPIO interface. Most pins
        on Spreadtrum SoCs can be selected for GPIO which are
        controlled by this driver
```

2.2.2 DTS 配置

- GPIO DTS 配置：以 SC9832E 为例，其他类似。

```
ap_gpio: gpio@40280000 {
    compatible = "sprd,sharkle-gpio";
    reg = <0x40280000 0x1000>; //0x40280000为GPIO控制寄存器的基地址
    gpio-controller;
    #gpio-cells = <2>;
    interrupt-controller;
    #interrupt-cells = <2>;
    interrupts = <GIC_SPI 35 IRQ_TYPE_LEVEL_HIGH>;
};
```

- 其他模块使用 GPIO DTS 的配置

```
gpio-keys {
    compatible = "gpio-keys";

    key-volumedown {
        label = "Volume Down Key";
        linux,code = <KEY_VOLUMEDOWN>;
        gpios = <&ap_gpio 124 GPIO_ACTIVE_LOW>; //GPIO是124，并配置为低电平有效
        debounce-interval = <2>;
        wakeup-source;
    };
};
```

2.2.3 调试方法

- 查看这个 pin 是否为 GPIO 功能
adb shell lookat 0x402a0000 + offset, 如果[5:4]等于 3, 表明是 GPIO 功能。
- 设置输入和输出
adb shell lookat -s 0x2 0x40280000 //设置 goio1 的 msk 位
adb shell lookat -s 0x2 0x40280008 //设置 gpio1 的方向为输出, 设置为 0 则为输入。

- 查看中断是否产生

adb shell lookat 0x40280020
生)。

//查看对应 GPIO 的 pending 位是否为 1 (1 表示中断产

说明

上述寄存器地址为示例，实际地址请查阅 SPEC 或联系 UNISOC 技术支持人员确认。

Unisoc Confidential For hiar

3 注意事项

3.1 边沿触发与电平触发的区别

- 边沿触发中断是可以锁住的。

即：产生中断后，如果外部电平发生切换，也不会导致 GPIO 中断消失，要配置对应的 Interrupt clear register 为 1 才会清除。

- 电平触发中断是随着电平的消失就消失，不会锁住。

即：产生中断后，如果外部电平高低发生切换，会导致 GPIO 中断自动消失。

使用电平触发要求电平稳定持续一定的时间（不同的场景，时间可能不同），确保软件中断处理程序可以处理完，避免引起 AP CPU 挂死等异常。因此按键、触摸、指纹等人为操作，抖动严重，不能使用电平触发中断。

例外：SC9863A 支持防抖，故可以使用电平触发中断。

图3-1 电平触发示意图



3.2 DeepSleep 时需要唤醒 AP 系统

- 边沿触发

– 没有 RCO 时钟方案的项目，如 SC9820E\SC9832E\SC7731E 等，在 chip deep sleep 时，AON_APB 是没有时钟的，不能使用边沿触发唤醒 AP 系统。

– 有 RCO 时钟方案的项目，如 UMS312\UMS512 等，在 chip deep sleep 且 sp 未进入 deep sleep 时（如：sensorhub 工作时 sp 则不会进入 deep sleep），AON_APB 是有时钟的，可以使用边沿触发唤醒 AP 系统。

考虑此场景下用边沿触发唤醒系统有依赖，因此不要使用边沿触发唤醒系统功能。

- 如果需要支持唤醒 AP 系统，建议使用支持 EIC 的 PIN 脚，具体请参考 SPEC。
- GPIO_plus 支持 debounce、latch 等模式，支持唤醒 AP 系统，如：SC9863A。

📖 说明

项目是否支持 RCO 方案请联系对应的 FAE 确认。

- 电平触发

一般都支持中断唤醒系统，正如 3.1 所述，电平触发要求稳定持续一定的时间。像按键、触模、指纹等人为操作由于电平抖动严重，变化不可控，因此不能使用电平触发唤醒系统。例外：SC9863A 使用的是 GPIO_plus，支持防抖，可以使用电平触发唤醒系统。

Unisoc Confidential For hiar