Linux GPIO
开发指南

版本号: 1.9
发布日期: 2025.6.17

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.08.04	XAA0249	初始版本
1.1	2023.07.28	XAA0312	增加 sun55iw3 部分 gpio 使用说明
1.2	2023.10.24	XAA0312	更新 sunxi_pinctrl 节点使用方法
1.3	2023.12.18	XAA0312	更新源码结构目录
1.4	2024.10.26	XAA0312	更新适用范围 更新驱动能力配置方式
1.5	2024.12.02	XAA0312	修改文档格式
1.6	2025.1.03	XAA0312	修改适用范围
1.7	2025.2.21	XAA0338	1. 修改内核设备树配置；2. 跟新源码结构目录；3. 更新调试方法配置细节；
1.7	2025.3.18	XAA0312	添加用户层 sysfs 节点使用方法
1.8	2025.5.21	XAA03338	修改 5.1.1 章节说明信息的对齐格式
1.9	2025.6.17	XAA0312	更新 debounce 配置说明

目　录

1 前言

1.1 文档简介

本文档对内核的 GPIO 接口使用进行详细的阐述，让用户明确掌握 GPIO 配置、申请等操作的编程方法。

1.2 目标读者

本文档适用于所有需要在 Linux 内核 sunxi 平台上开发设备驱动的相关人员。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.4-ansc	pinctrl-sunxi.c
Linux-5.10	pinctrl-sunxi.c
Linux-5.15	pinctrl-sunxi.c
Linux-6.6	pinctrl-sunxi.c

1.4 相关术语介绍

1.4.1 软件术语

表 1-2: Pinctrl 模块相关术语介绍

术语	解释说明
SUNXI	Allwinner 一系列 SOC 硬件平台
Pin controller	是对硬件模块的软件抽象，通常用来表示硬件控制器。能够处理引脚复用、属性配置等功能

术语	解释说明
Pin	根据芯片不同的封装方式，可以表现为球形、针型等。软件上采用常用一组无符号的整数 [0-maxpin] 来表示
Pin groups	外围设备通常都不只一个引脚，比如 SPI，假设接在 SoC 的 {0,8,16,24} 管脚，而另一个设备 I2C 接在 SoC 的 {24,25} 管脚。我们可以说这里有两个 pin groups。很多控制器都需要处理 pin groups。因此管脚控制器子系统需要一个机制用来枚举管脚组且检索一个特定组中实际枚举的管脚
Pinconfig	管脚可以被软件配置成多种方式，多数与它们作为输入/输出时的电气特性相关。例如，可以设置一个输出管脚处于高阻状态，或是 “三态”（意味着它被有效地断开连接）。或者可以通过设置将一个输入管脚与 VDD 或 GND 相连 (上拉/下拉)，以便在没有信号驱动管脚时使管脚拥有确认值
Pinmux	引脚复用功能，使用一个特定的物理管脚（ball/pad/finger/等等）进行多种扩展复用，以支持不同功能的电气封装习惯
Device tree	犹如它的名字，是一棵包括 cpu 的数量和类别、内存基地址、总线与桥、外设连接，中断控制器和 gpio 以及 clock 等系统资源的树，Pinctrl 驱动支持从 device tree 中定义的设备节点获取 pin 的配置信息

2 模块介绍

Pinctrl 框架是 linux 系统为统一各 SoC 厂商 pin 管理，避免各 SoC 厂商各自实现相同 pin 管理子系统而提出的。目的是为了减少 SoC 厂商系统移植工作量。

2.1 模块功能

许多 SoC 内部都包含 pin 控制器，通过 pin 控制器，我们可以配置一个或一组引脚的功能和特性。在软件上，Linux 内核 pinctrl 驱动可以操作 pin 控制器为我们完成如下工作：

- 枚举并且命名 pin 控制器可控制的所有引脚。
- 提供引脚的复用能力。
- 提供配置引脚的能力，如驱动能力、上拉下拉、数据属性等。
- 与 gpio 子系统的交互。
- 实现 pin 中断。

2.2 模块配置

2.2.1 Kernel Device Tree 配置说明

2.2.1.1 device tree 源码结构和路径

对于 Linux5.10 及以上平台:

- 设备树文件的配置是该 SoC 所有方案的通用配置，Linux-5.10 及以上内核中不再维护单独的 pinctrl 的 dtsi，直接将 pin 的信息放在了：bsp/configs/{kernel_ver}/sun*.dtsi。
- 板级设备树 (board.dts) 路径：/device/config/chips/{IC}/configs/{IC}/{kernel_ver}/board.dts。
- device tree 的源码包含关系如下：

```
board.dts
  |--------sun*.dtsi
```

2.2.1.2 device tree 对 gpio 控制器的通用配置

目前，在 sunxi 平台，我们根据电源域，注册两个 pinctrl 设备：r_pio 设备 (PL0 后的所有 pin) 和 pio 设备 (PL0 前的所有 pin)，两个设备的通用配置信息如下：

```
r_pio: pinctrl@07022000 {
    compatible = "allwinner,sun50iw9p1-r-pinctrl"; //兼容属性，用于驱动和设备绑定
    reg = <0x0 0x07022000 0x0 0x400>;      //寄存器基地址0x07022000和范围0x400
    clocks = <&clk_cpurpio>;               //r_pio设置使用的时钟
    device_type = "r_pio";                 //设备类型属性
    gpio-controller;                       //表示是一个gpio控制器
    interrupt-controller;                  //表示一个中断控制器，不支持中断可以删除
    #interrupt-cells = <3>;                //pin中断属性需要配置的参数个数，不支持中断可以删除
    #size-cells = <0>;                     //没有使用，配置0
    #gpio-cells = <3>;

    /*
     * 以下配置为模块使用的pin的配置，模块通过引用相应的节点对pin进行操作
     * 由于不同板级的pin经常改变，建议通过板级dts修改（参考下一小节）
     */
    s_rsb0_pins_a: s_rsb0@0 {
        allwinner,pins = "PL0", "PL1";
        allwinner,function = "s_rsb0";
        allwinner,muxsel = <2>;
        allwinner,drive = <2>;
        allwinner,pull = <1>;
    };

    /*
     * 以下配置为linux-5.10及以上内核模块使用pin的配置，模块通过引用相应的节点对pin进行操作
     * 由于不同板级的pin经常改变，建议将模块pin的引用放到board dts中
     * （类似pinctrl-0 = <&scr1_ph_pins>;),并使用scr1_ph_pins这种更有标识性的名字）。
     */
    scr1_ph_pins: scr1-ph-pins {
        pins = "PH0", "PH1";
        function = "sim1";
        drive-strength = <10>;
        bias-pull-up;
    };
};

pio: pinctrl@0300b000 {
    compatible = "allwinner,sun50iw9p1-pinctrl"; //兼容属性，用于驱动和设备绑定
    reg = <0x0 0x0300b000 0x0 0x400>;          //寄存器基地址0x0300b000和范围0x400
    interrupts = <GIC_SPI 51 IRQ_TYPE_LEVEL_HIGH>, /* AW1823_GIC_Spec: GPIOA: 83-32=51 */
                 <GIC_SPI 52 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 53 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 54 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 55 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 56 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 57 IRQ_TYPE_LEVEL_HIGH>;  //该设备每个bank支持的中断配置和gic中断号，每个中断号对应一个支持中断的bank
    device_type = "pio";        //设备类型属性
    clocks = <&clk_pio>, <&clk_losc>, <&clk_hosc>; //该设备使用的时钟
    gpio-controller;                       //表示是一个gpio控制器
    interrupt-controller;                  //表示是一个中断控制器
    #interrupt-cells = <3>;                //pin中断属性需要配置的参数个数，不支持中断可以删除
    #size-cells = <0>;                     //没有使用
```

```
        #gpio-cells = <3>;
        /* takes the debounce time in usec as argument */
    }
```

⚠ 注意

在配置 pinctrl 的配置时，我司提供两种配置驱动能力的方式：

1. **allwinner,drive = <0>，参数可选择 0 、1、2、3 ，此处的配置仅代表芯片手册中的不同驱动能力档位。**
2. **drive-strength = <10>，参数可选择 10、20、30、40，此处的配置仅代表芯片手册中的不同驱动能力档位。**

建议选择配置方式 1，具体配置哪个档位根据模块需求选择。

2.2.1.3 board.dts 板级配置

board.dts 用于保存每个板级平台的设备信息 (如 demo 板、demo2.0 板等等)，以 demo 板为例，board.dts 路径如下：

/device/config/chips/{CHIP}/configs/demo/board.dts

在 board.dts 中的配置信息如果在 *.dtsi 中 (如 sun50iw9p1.dtsi 等) 存在，则会存在以下覆盖规则：

- 相同属性和结点，board.dts 的配置信息会覆盖 *.dtsi 中的配置信息。
- 新增加的属性和结点，会追加到最终生成的 dtb 文件中。

对于 linux-5.10 及以上内核，修改驱动 pinctrl-0 引用的节点。

linux-5.10 及以上内核上面 board.dts 的配置如下：

```
&pio{
    input-debounce = <0 0 0 0 1 0 0 0>;    //配置中断采样频率，每个对应一个支持中断的bank，单位us
    vcc-pe-supply = <&reg_pio1_8>;    //配置IO口耐压值，例如这里的含义是将pe口设置成1.8v耐压值
};
```

2.2.2 menuconfig 配置说明

进入 {SDK} 根目录，执行./build.sh menuconfig

进入配置主界面，并按以下步骤操作:

首先，选择 Allwinner BSP 选项进入下一级配置，如下所示：

```
Allwinner BSP  --->      /* 选中 */
General setup  --->
Platform selection  --->
```

```
    Kernel Features  --->
    Boot options  --->
    Power management options  --->
    CPU Power Management  --->
[ ] 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  --->
```

选择 Device Drivers，进入下级配置，如下所示：

```
Platform Information  --->
Allwinner's Debugging Options  --->
Allwinner's Turbo Options  --->
Device Drivers  --->        /* 选中 */
Library routines  --->
NAND Drivers  --->
GPU Drivers  --->
```

选择 Pinctrl Drivers，进入下级配置，如下所示：

```
Clock Drivers  --->
Pinctrl Drivers  --->       /* 选中 */
UART Drivers  --->
Timer Drivers  --->
DMA Drivers  --->
RTC Drivers  --->
Bus Drivers  --->
Dump-Reg Drivers  --->
Watchdog Drivers  --->
IOMMU Drivers  --->
MBUS Drivers  --->
SD/MMC Drivers  --->
SUNXI UFS Drivers  ----
NAND Drivers  --->
......
```

选择 Pinctrl Support for Allwinner SoCs，并选择对应平台，如下所示：

```
<*> Pinctrl Support for Allwinner SoCs    /* 选中 */
Pinctrl Debugfs Driver
  ...
<*>  SUNxxIWx PIO       /* 选中对应平台 */
<*>  SUNxxIWx R_PIO     /* 选中对应平台 */
  ...
[*] Enable pinctrl dynamic debug        /* 选中 */
```

2.3 源码结构

```
.bsp
├──── drivers
   ├──── pinctrl
      ├──── Kconfig
      ├──── Makefile
      ├──── pinctrl-debugfs.c
      ├──── pinctrl-sunXXiwX.c    /* 不同平台驱动源码 */
      ├──── pinctrl-sunxi.c
      └──── pinctrl-sunxi.h
```

2.4 软件框架

Sunxi Pinctrl 驱动模块的框架如下图所示，整个驱动模块可以分成 4 个部分：pinctrl api、pinctrl common frame、sunxi pinctrl driver，以及 board configuration。（图中最上面一层 device driver 表示 Pinctrl 驱动的使用者）

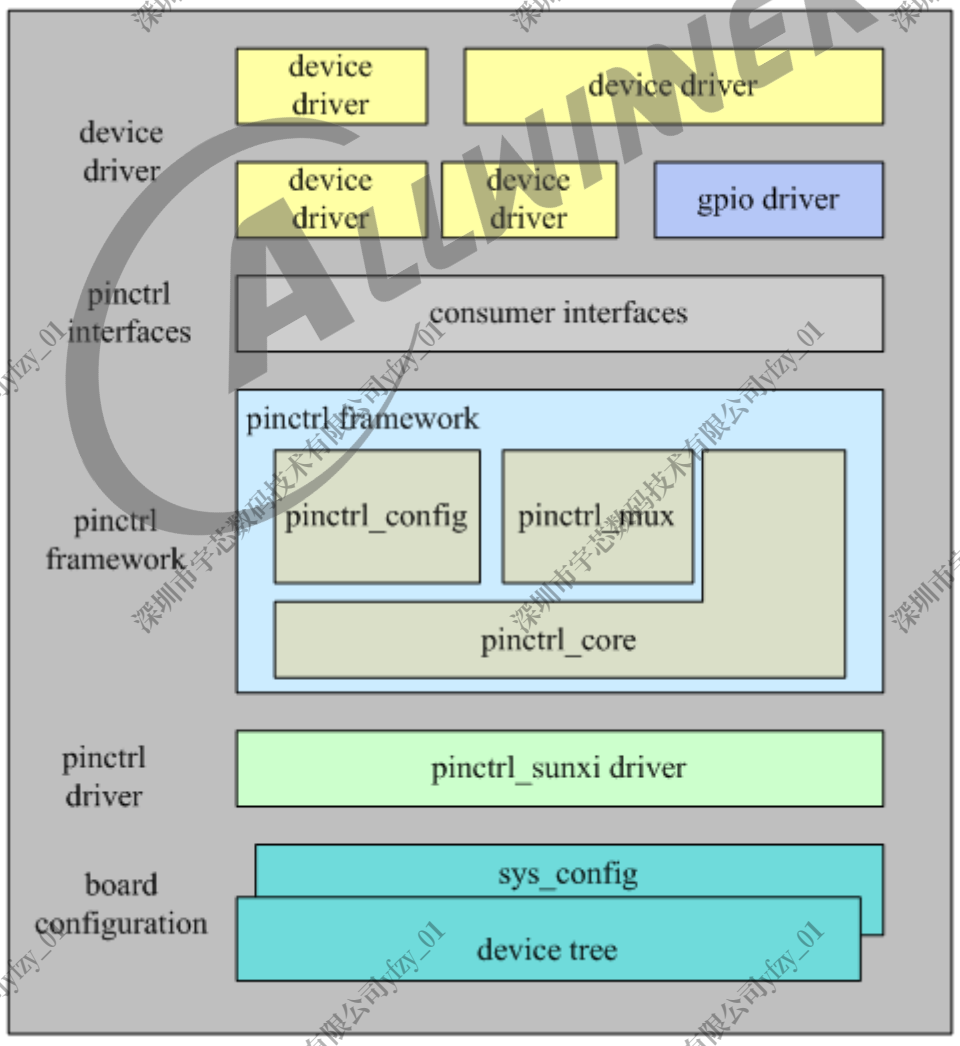

图 2-1: pinctrl 驱动整体框架图

Pinctrl api: pinctrl 提供给上层用户调用的接口。

Pinctrl framework：Linux 提供的 pinctrl 驱动框架。

Pinctrl sunxi driver：sunxi 平台需要实现的驱动。

Board configuration：设备 pin 配置信息，一般采用设备树进行配置。

2.5 state/pinmux/pinconfig

Pinctrl framework 主要处理 pinstate、pinmux 和 pinconfig 三个功能，pinstate 和 pinmux、pinconfig 映射关系如下图所示。

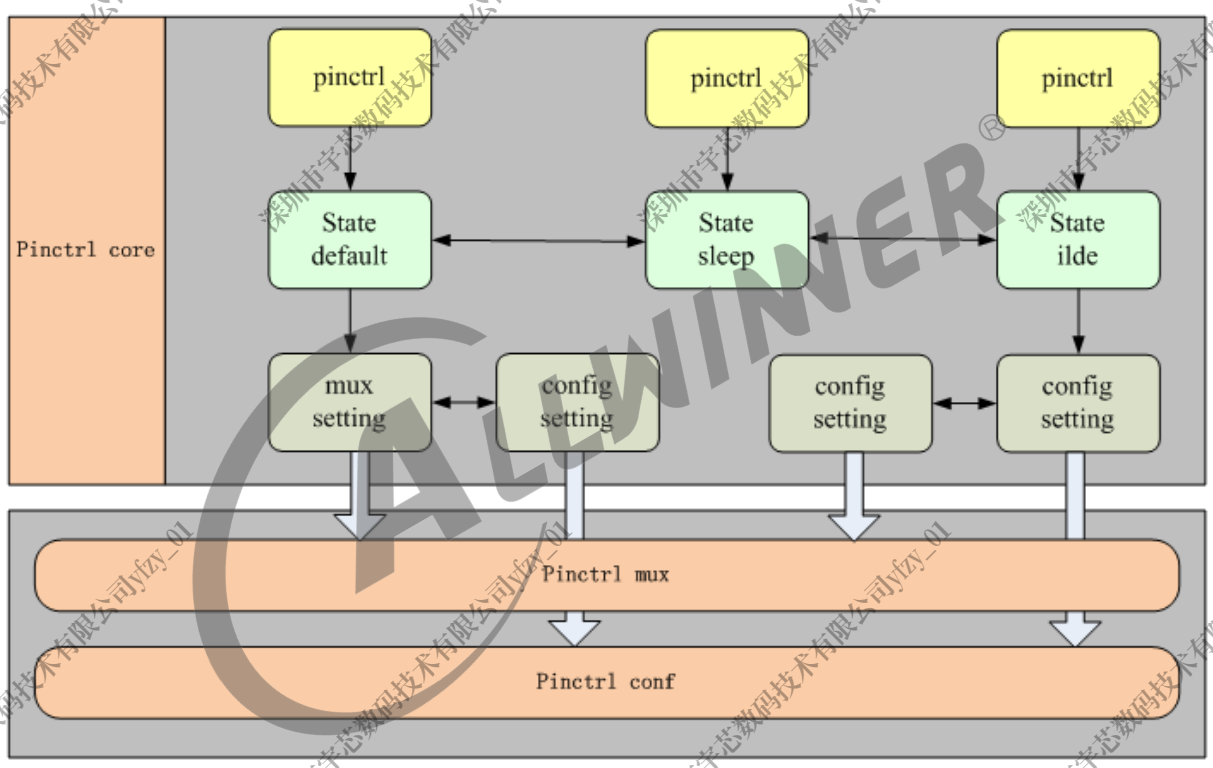

图 2-2: pinctrl 驱动 framework 图

系统运行在不同的状态，pin 配置有可能不一样，比如系统正常运行时，设备的 pin 需要一组配置，但系统进入休眠时，为了节省功耗，设备 pin 需要另一组配置。Pinctrl framwork 能够有效管理设备在不同状态下的引脚配置。

3 模块接口说明

3.1 pinctrl 接口说明

3.1.1 pinctrl_get

- 函数原型：struct pinctrl pinctrl_get(struct device dev)
- 作用：获取设备的 pin 操作句柄，所有 pin 操作必须基于此 pinctrl 句柄。
- 参数：
  - dev：指向申请 pin 操作句柄的设备句柄。
- 返回：
  - 成功，返回 pinctrl 句柄。
  - 失败，返回 NULL。

3.1.2 pinctrl_put

- 函数原型：void pinctrl_put(struct pinctrl *p)
- 作用：释放 pinctrl 句柄，必须与 pinctrl_get 配对使用。
- 参数：
  - p：指向释放的 pinctrl 句柄。
- 返回：
  - 没有返回值。

⚠ 注意

必须与 pinctrl_get 配对使用。

3.1.3 devm_pinctrl_get(推荐使用)

- 函数原型：struct pinctrl devm_pinctrl_get(struct device dev)
- 作用：根据设备获取 pin 操作句柄，所有 pin 操作必须基于此 pinctrl 句柄，与 pinctrl_get 功能完全一样，只是 devm_pinctrl_get 会将申请到的 pinctrl 句柄做记录，绑定到设备句柄信息中。设备驱动申请 pin 资源，推荐优先使用 devm_pinctrl_get 接口。
- 参数：
  - dev：指向申请 pin 操作句柄的设备句柄。
- 返回：
  - 成功，返回 pinctrl 句柄。
  - 失败，返回 NULL。

3.1.4 devm_pinctrl_put(推荐使用)

- 函数原型：void devm_pinctrl_put(struct pinctrl *p)
- 作用：释放 pinctrl 句柄，必须与 devm_pinctrl_get 配对使用。
- 参数：
  - p：指向释放的 pinctrl 句柄。
- 返回：
  - 没有返回值。

⚠ 注意

必须与 devm_pinctrl_get 配对使用，可以不显式的调用该接口。

3.1.5 pinctrl_lookup_state

- 函数原型：struct pinctrl_state pinctrl_lookup_state(struct pinctrl p, const char *name)
- 作用：根据 pin 操作句柄，查找 state 状态句柄。
- 参数：
  - p：指向要操作的 pinctrl 句柄。
  - name：指向状态名称，如“default”、“sleep”等。
- 返回：

- 成功，返回执行 pin 状态的句柄 struct pinctrl_state *。
- 失败，返回 NULL。

3.1.6 pinctrl_select_state

- 函数原型：int pinctrl_select_state(struct pinctrl p, struct pinctrl_state s)
- 作用：将 pin 句柄对应的 pinctrl 设置为 state 句柄对应的状态。
- 参数：
  - p：指向要操作的 pinctrl 句柄。
  - s：指向 state 句柄。
- 返回：
  - 成功，返回 0。
  - 失败，返回错误码。

3.1.7 devm_pinctrl_get_select

- 函数原型：struct pinctrl devm_pinctrl_get_select(struct device dev, const char *name)
- 作用：获取设备的 pin 操作句柄，并将句柄设定为指定状态。
- 参数：
  - dev：指向管理 pin 操作句柄的设备句柄。
  - name：要设置的 state 名称，如“default”、“sleep”等。
- 返回：
  - 成功，返回 pinctrl 句柄。
  - 失败，返回 NULL。

3.1.8 devm_pinctrl_get_select_default

- 函数原型：struct pinctrl devm_pinctrl_get_select_default(struct device dev)
- 作用：获取设备的 pin 操作句柄，并将句柄设定为默认状态。
- 参数：
  - dev：指向管理 pin 操作句柄的设备句柄。
- 返回：

- 成功，返回 pinctrl 句柄。
- 失败，返回 NULL。

3.2 gpio 接口说明

3.2.1 gpio_request

- 函数原型：int gpio_request(unsigned gpio, const char *label)
- 作用：申请 gpio，获取 gpio 的访问权。
- 参数：
  - gpio：gpio 编号。
  - label：gpio 名称，可以为 NULL。
- 返回：
  - 成功，返回 0。
  - 失败，返回错误码。

3.2.2 gpio_free

- 函数原型：void gpio_free(unsigned gpio)
- 作用：释放 gpio。
- 参数：
  - gpio：gpio 编号。
- 返回：
  - 无返回值。

3.2.3 gpio_direction_input

- 函数原型：int gpio_direction_input(unsigned gpio)
- 作用：设置 gpio 为 input。
- 参数：
  - gpio：gpio 编号。

- 返回：
  - 成功，返回 0。
  - 失败，返回错误码。

3.2.4 gpio_direction_output

- 函数原型：int gpio_direction_output(unsigned gpio, int value)
- 作用：设置 gpio 为 output。
- 参数：
  - gpio：gpio 编号。
  - value：期望设置的 gpio 电平值，非 0 表示高, 0 表示低。
- 返回：
  - 成功，返回 0.
  - 失败，返回错误码。

3.2.5 __gpio_get_value

- 函数原型：int __gpio_get_value(unsigned gpio)
- 作用：获取 gpio 电平值 (gpio 已为 input/output 状态)。
- 参数：
  - gpio：gpio 编号。
- 返回：
  - 返回 gpio 对应的电平逻辑，1 表示高, 0 表示低。

3.2.6 __gpio_set_value

- 函数原型：void __gpio_set_value(unsigned gpio, int value)
- 作用：设置 gpio 电平值 (gpio 已为 input/output 状态)。
- 参数：
  - gpio：gpio 编号。
  - value：期望设置的 gpio 电平值，非 0 表示高, 0 表示低。
- 返回：
  - 无返回值

3.2.7 of_get_named_gpio

- 函数原型：int of_get_named_gpio(struct device_node np, const char propname, int index)
- 作用：通过名称从 dts 解析 gpio 属性并返回 gpio 编号。
- 参数：
  - np：指向使用 gpio 的设备结点。
  - propname：dts 中属性的名称。
  - index：dts 中属性的索引值。
- 返回：
  - 成功，返回 gpio 编号。
  - 失败，返回错误码。

3.2.8 of_get_named_gpio_flags

- 函数原型：int of_get_named_gpio_flags(struct device_node np, const char list_name, int index, enum of_gpio_flags *flags)
- 作用：通过名称从 dts 解析 gpio 属性并返回 gpio 编号。
- 参数：
  - np：指向使用 gpio 的设备结点。
  - propname：dts 中属性的名称。
  - index：dts 中属性的索引值
  - flags：在 sunxi 平台上，必须定义为 struct gpio_config * 类型变量，因为 sunxi pinctrl 的 pin 支持上下拉，驱动能力等信息，而内核 enum of_gpio_flags * 类型变量只能包含输入、输出信息，后续 sunxi 平台需要标准化该接口。
- 返回：
  - 成功，返回 gpio 编号。
  - 失败，返回错误码。

⚠ 注意

linux-5.10 及以上内核已经标准化该接口，直接采用 enum of_gpio_flags 的定义。
linux-6.6 内核不在支持该接口，请替换为 of_get_named_gpio。

4 调试方法

4.1 内核驱动调试节点

4.1.1 利用 sunxi_dump 读写相应寄存器

需要开启 SUNXI_DUMP 模块：

```
make kernel_menuconfig

选择AW_DUMP_REG [=y]
选择AW_DUMPREG_DYNAMIC_DEBUG [=y]
```

使用方法：

```
cd /sys/class/sunxi_dump
1.查看一个寄存器
  echo 0x0300b048 > dump ;cat dump

2.写值到寄存器上
  echo 0x0300b058 0xfff > write ;cat  write

3.查看一片连续寄存器
  echo 0x0300b000,0x0300bfff > dump;cat dump

4.写一组寄存器的值
  echo 0x0300b058 0xfff,0x0300b0a0 0xfff > write;cat write

通过上述方式，可以查看，修改相应gpio的寄存器，从而发现问题所在。
```

4.1.2 利用 sunxi_pinctrl 的 debug 节点

需要开启 CONFIG_DEBUG_FS 与 CONFIG_AW_PINCTRL_DEBUGFS：

```
make kernel_menuconfig

---> Kernel hacking
  ---> Generic Kernel Debugging Instruments --->
    ---> Debug Filesystem  /* 选中 */

---> Allwinner BSP --->
    ---> Device Drivers --->
      ---> Pinctrl Drivers --->
        ---> Pinctrl Debugfs Driver /* 选中 */
```

挂载文件节点，并进入相应目录：

```
mount -t debugfs none /sys/kernel/debug
cd /sys/kernel/debug/sunxi_pinctrl
```

1. 指定 debug 的 pin 域

查看/修改 pin 的配置前，需要指定当前需要查看的 pin 域。可通过查看 pinctrl 节点下的 pin 设备名称来确定，如下所示：

一般有两类设备名称：

- 第一种是以 addr+pinctrl 的方式命名，如下：

```
/sys/kernel/debug/sunxi_pinctrl # ls ../pinctrl/
2000000.pinctrl  pinctrl-devices  pinctrl-maps
7022000.pinctrl  pinctrl-handles
```

- 第二种是以 pio 或者 r_pio 命名，如下：

```
/sys/kernel/debug/sunxi_pinctrl # ls ../ pinctrl /
pio   pinctrl-devices  pinctrl-maps
r_pio   pinctrl-handles
```

若需要查看 PA~PK 引脚配置：

```
echo 2000000.pinctrl > dev_name
或者
echo pio > dev_name
```

若需要查看 PL~PM 引脚配置：

```
echo 7022000.pinctrl > dev_name
或者
echo r_pio > dev_name
```

2. 查看 pin 的配置

```
  echo PC2 > sunxi_pin
  cat sunxi_pin_configure
```

结果如下图所示：

```
/sys/kernel/debug # cd sunxi_pinctrl/
/sys/kernel/debug/sunxi_pinctrl # ls
data                    function                sunxi_pin
device                  platform                sunxi_pin_configur
dlevel                  pull
/sys/kernel/debug/sunxi_pinctrl # echo PC2 > sunxi_pin
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 0
```

图 4-1: 查看 pin 配置图

3. 修改 pin 属性

每个 pin 都有四种属性，如复用 (function)，数据 (data)，驱动能力 (dlevel)，上下拉 (pull)，修改 pin 属性的命令如下：

```
echo PC2 1 > pull;cat pull
cat sunxi_pin_configure    //查看修改情况
```

修改后结果如下图所示：

```
/sys/kernel/debug/sunxi_pinctrl # echo PC2 > sunxi_pin
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 0
/sys/kernel/debug/sunxi_pinctrl # echo PC2 1 > pull
/sys/kernel/debug/sunxi_pinctrl # cat sunxi_pin_configure
pin[PC2] funciton: 4
pin[PC2] data: 0
pin[PC2] dlevel: 1
pin[PC2] pull: 1
```

图 4-2: 修改结果图

4.1.3 利用 pinctrl core 的 debug 节点

```
mount -t debugfs none /sys/kernel/debug

cd /sys/kernel/debug/pinctrl
```

1. 查看 pin 的管理设备：

```
cat pinctrl-devices
```

结果如下图所示:

```
130|console:/sys/kernel/debug/pinctrl # ls
pinctrl-devices pinctrl-handles pinctrl-maps pio r_pio
console:/sys/kernel/debug/pinctrl # cat pinctrl-devices
name [pinmux] [pinconf]
r_pio yes yes
pio yes yes
console:/sys/kernel/debug/pinctrl #
```

图 4-3: pin 设备图

2. 查看 pin 的状态和对应的使用设备

```
1 cat pinctrl-handles
```

结果如下图 log 所示：

```
console:/sys/kernel/debug/pinctrl # ls
pinctrl-devices pinctrl-handles pinctrl-maps pio r_pio
console:/sys/kernel/debug/pinctrl # cat pinctrl-handles
Requested pin control handlers their pinmux maps:
device: twi3 current state: sleep
 state: default
   type: MUX_GROUP controller pio group: PA10 (10) function: twi3 (15)
   type: CONFIGS_GROUP controller pio group PA10 (10)config 00001409
config 00000005
   type: MUX_GROUP controller pio group: PA11 (11) function: twi3 (15)
   type: CONFIGS_GROUP controller pio group PA11 (11)config 00001409
config 00000005
 state: sleep
   type: MUX_GROUP controller pio group: PA10 (10) function: io_disabled (5)
   type: CONFIGS_GROUP controller pio group PA10 (10)config 00001409
config 00000001
   type: MUX_GROUP controller pio group: PA11 (11) function: io_disabled (5)
   type: CONFIGS_GROUP controller pio group PA11 (11)config 00001409
config 00000001
device: twi5 current state: default
 state: default
   type: MUX_GROUP controller r_pio group: PL0 (0) function: s_twi0 (3)
   type: CONFIGS_GROUP controller r_pio group PL0 (0)config 00001409
config 00000005
   type: MUX_GROUP controller r_pio group: PL1 (1) function: s_twi0 (3)
   type: CONFIGS_GROUP controller r_pio group PL1 (1)config 00001409
config 00000005
 state: sleep
   type: MUX_GROUP controller r_pio group: PL0 (0) function: io_disabled (4)
   type: CONFIGS_GROUP controller r_pio group PL0 (0)config 00001409
config 00000001
   type: MUX_GROUP controller r_pio group: PL1 (1) function: io_disabled (4)
   type: CONFIGS_GROUP controller r_pio group PL1 (1)config 00001409
config 00000001
device: soc@03000000:pwm5@0300a000 current state: active
 state: active
   type: MUX_GROUP controller pio group: PA12 (12) function: pwm5 (16)
   type: CONFIGS_GROUP controller pio group PA12 (12)config 00000001
config 00000000
 config 00000000
```

```
  state: sleep
    type: MUX_GROUP controller pio group: PA12 (12) function: io_disabled (5)
    type: CONFIGS_GROUP controller pio group PA12 (12)config 00000001
config 00000000
config 00000000
device: uart0 current state: default
  state: default
  state: sleep
device: uart1 current state: default
  state: default
    type: MUX_GROUP controller pio group: PG6 (95) function: uart1 (37)
    type: CONFIGS_GROUP controller pio group PG6 (95)config 00001409
config 00000005
    type: MUX_GROUP controller pio group: PG7 (96) function: uart1 (37)
    type: CONFIGS_GROUP controller pio group PG7 (96)config 00001409
config 00000005
    type: MUX_GROUP controller pio group: PG8 (97) function: uart1 (37)
    type: CONFIGS_GROUP controller pio group PG8 (97)config 00001409
config 00000005
    type: MUX_GROUP controller pio group: PG9 (98) function: uart1 (37)
    type: CONFIGS_GROUP controller pio group PG9 (98)config 00001409
config 00000005
  state: sleep
    type: MUX_GROUP controller pio group: PG6 (95) function: io_disabled (5)
    type: CONFIGS_GROUP controller pio group PG6 (95)config 00001409
config 00000001
    type: MUX_GROUP controller pio group: PG7 (96) function: io_disabled (5)
    type: CONFIGS_GROUP controller pio group PG7 (96)config 00001409
config 00000001
    type: MUX_GROUP controller pio group: PG8 (97) function: io_disabled (5)
    type: CONFIGS_GROUP controller pio group PG8 (97)config 00001409
config 00000001
    type: MUX_GROUP controller pio group: PG9 (98) function: io_disabled (5)
    type: CONFIGS_GROUP controller pio group PG9 (98)config 00001409
    ....
```

从上面的部分 log 可以看到那些设备管理的 pin 以及 pin 当前的状态是否正确。以 twi3 设备为例，twi3 管理的 pin 有 PA10/PA11，分别有两组状态 sleep 和 default，default 状态表示使用状态，sleep 状态表示 pin 处于 io disabled 状态，表示 pin 不可正常使用，twi3 设备使用的 pin 当前状态处于 sleep 状态的。

4.2 用户层调试节点

需要打开内核配置项：GPIO_SYSFS

启动后小机端会存在sys/class/gpio节点

节点说明：

/sys/class/gpio/export：通过向该文件写入 GPIO 编号，可以将 GPIO 引脚导出（export）为用户空间可见的节点。导出后，可以访问相应的 GPIO 目录。

/sys/class/gpio/unexport：通过向该文件写入 GPIO 编号，可以取消导出（unexport）已导出的 GPIO 引

脚，使其不再可见。

/sys/class/gpio/gpioX/direction：该文件用于设置 GPIO 引脚的方向，可以将其配置为输入（input）或输出（output）。通过向文件写入 “in” 或 “out”，可以设置相应的方向。

/sys/class/gpio/gpioX/value：对于配置为输出的 GPIO 引脚，可以通过读取或写入该文件来获取或设置引脚的电平值。读取文件可以获取当前的引脚电平值（0 或 1），写入文件可以将引脚设置为高电平（1）或低电平（0）。

使用示例如下：

如果需要将 PB3 设置为高电平，操作流程如下：

```
cd sys/class/gpio

echo 35 > export /* 将PB3引脚导出 */
echo out > gpio35/direction /* 将引脚设置为output */
echo 1 > gpio35/value /* 设置引脚为高电平 */
```

如何计算引脚对应的值：

GPIO 分为多个 bank，比如 PA、PB、PC、PD.. 每个 bank 最多有 32 个引脚比如 PA0...PA30...

PA0 对应的值为 0，PB0 对应的值为 32

若需要计算 PF9，则计算公式为：PF9 = 5 * 32 + 9

5 使用示例

5.1 使用 pin 的驱动 dts 配置示例

对于使用 pin 的驱动来说，驱动主要设置 pin 的常用的几种功能，列举如下：

- 驱动使用者只配置通用 GPIO, 即用来做输入、输出和中断的。
- 驱动使用者设置 pin 的 pin mux，如 uart 设备的 pin,lcd 设备的 pin 等，用于特殊功能。
- 驱动使用者既要配置 pin 的通用功能，也要配置 pin 的特性。

下面对常见使用场景进行分别介绍。

5.1.1 配置通用 GPIO 功能/中断功能

用法一：配置 GPIO 中断，device tree 配置 demo 如下所示：

```
soc{
    ...
    gpiokey {
        device_type = "gpiokey";
        compatible = "gpio-keys";

        ok_key {
            device_type = "ok_key";
            label  = "ok_key";
            gpios = <&r_pio 0 4 GPIO_ACTIVE_HIGH>;
            linux, input-type = "1>";
            linux, code = <0x1c>;
            wakeup-source = <0x1>;
        };
    };
    ...
};
```

说明

说明：gpio in/gpio out/ interrupt采用dts的配置方法，配置参数解释如下：

对于linux-5.10及以上内核:
gpios = <&r_pio 0 4 GPIO_ACTIVE_HIGH>;
&r_pio：指向哪个pio，属于cpus要用&r_pio
0：哪个bank
4：哪个pin
GPIO_ACTIVE_HIGH：gpio active状态，如果需要上下拉，可以或上GPIO_PULL_UP、GPIO_PULL_DOWN标志

5.1.2 用法二

用法二：配置设备引脚，device tree 配置 demo 如下所示：

```
device tree对应配置
soc{
    pio: pinctrl@0300b000 {
        ...
        mmc2_ds_pin: mmc2-ds-pin {
            pins = "PC1";
            function = "mmc2";
            drive-strength = <30>;
            bias-pull-up;
        };
        ...
    };
    ...
    uart0: uart@05000000 {
        compatible = "allwinner,sun8i-uart";
        device_type = "uart0";
        reg = <0x0 0x05000000 0x0 0x400>;
        interrupts = <GIC_SPI 49 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clk_uart0>;
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&uart0_pins_a>;
        pinctrl-1 = <&uart0_pins_b>;
        uart0_regulator = "vcc-io";
        uart0_port = <0>;
        uart0_type = <2>;
    };
    ...
};
```

其中：

- pinctrl-0 对应 pinctrl-names 中的 default，即模块正常工作模式下对应的 pin 配置。
- pinctrl-1 对应 pinctrl-names 中的 sleep，即模块休眠模式下对应的 pin 配置。

5.2 接口使用示例

5.2.1 配置设备引脚

一般设备驱动只需要使用一个接口 devm_pinctrl_get_select_default 就可以申请到设备所有 pin 资源。

```
static int sunxi_pin_req_demo(struct platform_device *pdev)
{
    struct pinctrl *pinctrl;

    /* request device pinctrl, set as default state */
    pinctrl = devm_pinctrl_get_select_default(&pdev->dev);
```

```
    if (IS_ERR_OR_NULL(pinctrl))
        return -EINVAL;

    return 0;
}
```

5.2.2 获取 GPIO 号

```
static int sunxi_pin_req_demo(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct device_node *np = dev->of_node;
    unsigned int gpio;

    #get gpio config in device node.
    gpio = of_get_named_gpio(np, "vdevice_3", 0);
    if (! gpio_is_valid(gpio)) {
        if (gpio != -EPROBE_DEFER)
            dev_err(dev, "Error getting vdevice_3\n");
        return gpio;
    }
}
```

5.2.3 GPIO 属性配置

通过 pin_config_set/pin_config_get/pin_config_group_set/pin_config_group_get 接口单独控制指定 pin 或 group 的相关属性。

```
static int pctrltest_request_all_resource(void)
{
    struct device *dev;
    struct device_node *node;
    struct pinctrl *pinctrl;
    struct sunxi_gpio_config *gpio_list = NULL;
    struct sunxi_gpio_config *gpio_cfg;
    unsigned gpio_count = 0;
    unsigned gpio_index;
    unsigned long config;
    int ret;

    dev = bus_find_device_by_name(&platform_bus_type, NULL, sunxi_ptest_data->dev_name);
    if (! dev) {
        pr_warn("find device [%s] failed...\ n", sunxi_ptest_data->dev_name);
        return -EINVAL;
    }

    node = of_find_node_by_type(NULL, dev_name(dev));
    if (! node) {
        pr_warn("find node for device [%s] failed...\n", dev_name(dev));
        return -EINVAL;
    }
    dev->of_node = node;
```

```
25
26
27      pr_warn("++++++++++++++++++++++++++%s++++++++++++++++++++++++++\n", __func__);
28      pr_warn("device[%s] all pin resource we want to request\n", dev_name(dev));
29      pr_warn("-----------------------------------------\          n");
30
31      pr_warn("step1: request pin all resource.\n");
32      pinctrl  = devm_pinctrl_get_select_default(dev);
33      if  (IS_ERR_OR_NULL(pinctrl)) {
34          pr_warn("request pinctrl handle for device [%s] failed...\ n",  dev_name(dev));
35          return -EINVAL;
36      }
37
38      pr_warn("step2: get device[%s] pin count.\n", dev_name(dev));
39      ret  = dt_get_gpio_list(node, &gpio_list, &gpio_count);
40      if  (ret  < 0 ||  gpio_count == 0) {
41          pr_warn(" devices own 0 pin resource or look for main key failed!\n");
42          return -EINVAL;
43      }
44
45      pr_warn("step3: get device[%s] pin configure and check.\n", dev_name(dev));
46      for  (gpio_index = 0; gpio_index < gpio_count; gpio_index++) {
47          gpio_cfg = &gpio_list[gpio_index];
48
49          /*check function config */
50          config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_FUNC, 0xFFFF);
51          pin_config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
52          if  (gpio_cfg->mulsel != SUNXI_PINCFG_UNPACK_VALUE(config)) {
53              pr_warn("failed! mul value isn't  equal as dt.\ n");
54              return -EINVAL;
55          }
56
57          /*check pull config */
58          if  (gpio_cfg->pull != GPIO_PULL_DEFAULT) {
59              config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_PUD, 0xFFFF);
60              pin_config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
61              if  (gpio_cfg->pull !=  SUNXI_PINCFG_UNPACK_VALUE(config)) {
62                  pr_warn("failed! pull  value isn' t  equal as dt.\ n");
63                  return -EINVAL;
64              }
65          }
66
67          /*check dlevel config */
68          if  (gpio_cfg->drive != GPIO_DRVLVL_DEFAULT) {
69              config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DRV, 0XFFFF);
70              pin_config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
71              if  (gpio_cfg->drive != SUNXI_PINCFG_UNPACK_VALUE(config)) {
72                  pr_warn("failed! dlevel value isn' t  equal as dt.\ n");
73                  return -EINVAL;
74              }
75          }
76
77          /*check data config */
78          if  (gpio_cfg->data != GPIO_DATA_DEFAULT) {
79              config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DAT, 0XFFFF);
80              pin_config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
81              if  (gpio_cfg->data != SUNXI_PINCFG_UNPACK_VALUE(config)) {
82                  pr_warn("failed! pin data value isn' t  equal as dt.\ n");
83                  return -EINVAL;
84              }
```

```
        }
    }

    pr_warn("-------------------------------------------\              n");
    pr_warn("test pinctrl request all  resource success!\n");
    pr_warn("++++++++++++++++++++++++++end++++++++++++++++++++++++++\n\n");
    return 0;
}
注：需要注意，存在SUNXI_PINCTRL和SUNXI_R_PINCTRL两个pinctrl设备，cpus域的pin需要使用SUNXI_R_PINCTRL
```

⚠ 注意

linux5.10 及以上内核中使用 pinctrl_gpio_set_config 配置 gpio 属性，对应使用 pinconf_to_config_pack 生成 config 参数：

- **SUNXI_PINCFG_TYPE_FUNC 已不再生效，暂未支持 FUNC 配置（建议使用 pinctrl_select_state 接口代替）。**
- **SUNXI_PINCFG_TYPE_PUD 更新为内核标准定义（PIN_CONFIG_BIAS_PULL_UP/PIN_CONFIG_BIAS_PULL_DOWN）。**
- **SUNXI_PINCFG_TYPE_DRV 更新为内核标准定义（PIN_CONFIG_DRIVE_STRENGTH），相应的 val 对应关系为（0->10, 1->20...）。**
- **SUNXI_PINCFG_TYPE_DAT 已不再生效，暂未支持 DAT 配置（建议使用 gpio_direction_output 或者 __gpio_set_value 设置电平值）。**

5.3 设备驱动使用 GPIO 中断功能

方式一：通过 gpio_to_irq 获取虚拟中断号，然后调用申请中断函数即可。

目前 sunxi-pinctrl 使用 irq-domain 为 gpio 中断实现虚拟 irq 的功能，使用 gpio 中断功能时，设备驱动只需要通过 gpio_to_irq 获取虚拟中断号后，其他均可以按标准 irq 接口操作。

```
static int sunxi_gpio_eint_demo(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    int virq;
    int ret;
    /* map the virq of gpio */
    virq = gpio_to_irq(GPIOA(0));
    if (IS_ERR_VALUE(virq)) {
        pr_warn("map gpio [%d] to virq failed, errno = %d\n",
        GPIOA(0), virq);
        return -EINVAL;
    }
    pr_debug("gpio [%d] map to virq [%d] ok\n", GPIOA(0), virq);
    /* request virq, set virq type to high level trigger */
    ret = devm_request_irq(dev, virq, sunxi_gpio_irq_test_handler,
    IRQF_TRIGGER_HIGH, "PA0_EINT", NULL);
    if (IS_ERR_VALUE(ret)) {
        pr_warn("request virq %d failed, errno = %d\n", virq, ret);
        return -EINVAL;
    }
    return 0;
```

```
}
```

方式二：通过 dts 配置 gpio 中断，通过 dts 解析函数获取虚拟中断号，最后调用申请中断函数即可，demo 如下所示：

```
dts配置如下：
soc{
  ...
  Vdevice: vdevice@0 {
    compatible = "allwinner,sun8i-vdevice";
    device_type = "Vdevice";
    interrupt-parent = <&pio>;                    /*依赖的中断控制器(带interrupt-controller属性的结点)*/
    interrupts = < PD 3 IRQ_TYPE_LEVEL_HIGH>;
                   | |        `------------------    中断触发条件、类型
                   | `--------------------------    pin bank内偏移
                   `----------------------------    哪个bank
    pinctrl-names = "default";
    pinctrl-0 = <&vdevice_pins_a>;
    test-gpios = <&pio PC 3 1 2 2 1>;
    status = "okay";
  };
  ...
};
```

在驱动中，通过 platform_get_irq() 标准接口获取虚拟中断号，如下所示：

```
static int sunxi_pctrltest_probe(struct platform_device *pdev)
{
    struct device_node *np = pdev->dev.of_node;
    struct gpio_config config;
    int gpio, irq;
    int ret;

    if (np == NULL) {
        pr_err("Vdevice failed to get of_node\n");
        return -ENODEV;
    }
  ....
  irq = platform_get_irq(pdev, 0);
  if (irq < 0) {
    printk("Get irq error!\n");
    return -EBUSY;
  }
  .....
  sunxi_ptest_data->irq = irq;
  ......
  return ret;
}

//申请中断：
static int pctrltest_request_irq(void)
{
    int ret;
    int virq = sunxi_ptest_data->irq;
    int trigger = IRQF_TRIGGER_HIGH;

    reinit_completion(&sunxi_ptest_data->done);

    pr_warn("step1: request irq(%s level) for irq:%d.\n",
```

```
                trigger == IRQF_TRIGGER_HIGH ? "high" : "low", virq);
    ret = request_irq(virq, sunxi_pinctrl_irq_handler_demo1,
                      trigger, "PIN_EINT", NULL);
    if (IS_ERR_VALUE(ret)) {
        pr_warn("request irq failed !\n");
        return -EINVAL;
    }

    pr_warn("step2: wait for irq.\n");
    ret = wait_for_completion_timeout(&sunxi_ptest_data->done, HZ);
    if (ret == 0) {
        pr_warn("wait for irq timeout!\n");
        free_irq(virq, NULL);
        return -EINVAL;
    }

    free_irq(virq, NULL);

    pr_warn("-----------------------------------------\n");
    pr_warn("test pin eint success !\n");
    pr_warn("++++++++++++++++++++++++++end++++++++++++++++++++++++++\n\n\n");

    return 0;
}
```

5.4 设备驱动设置中断 debounce 功能

方式一：通过 dts 配置每个中断 bank 的 debounce，以 pio 设备为例，如下所示：

```
&pio {
    /* takes the debounce time in usec as argument */
    input-debounce = <0 0 0 0 0 0 0>;
                      | | | | | | `----------  PA bank
                      | | | | | `------------  PC bank
                      | | | | `--------------  PD bank
                      | | | `----------------  PF bank
                      | | `------------------  PG bank
                      | `--------------------  PH bank
                      `----------------------  PI  bank
};
```

注意：input-debounce 的属性值中需把 pio 设备支持中断的 bank 都配上，如果缺少，会以 bank 的顺序设置相应的属性值到 debounce 寄存器，缺少的 bank 对应的 debounce 应该是默认值（启动时没修改的情况）。debounce 取值范围是 0~1000000（单位：ns）。

比如，pinctrl-sunxxiwx.c 驱动文件中，定义 irq_bank 如下：

```
static const unsigned int sun55iw3_bank_base[] = {
    SUNXI_BANK_OFFSET('B', 'A'),
    SUNXI_BANK_OFFSET('C', 'A'),
    SUNXI_BANK_OFFSET('D', 'A'),
    SUNXI_BANK_OFFSET('E', 'A'),
    SUNXI_BANK_OFFSET('F', 'A'),
    SUNXI_BANK_OFFSET('G', 'A'),
};
```

则 dtsi 的配置为：

```
input-debounce = <0 0 0 0 0 0 >;
```

从左到右依次为 PB、PC、PD、PE、PF、PG 的 debounce 配置。

方式二：驱动模块调用内核 gpio 相关接口设置中断 debounce

```
1 static inline int gpio_set_debounce(unsigned gpio, unsigned debounce); /* 较旧的接口，直接使用GPIO编号 */
2 int gpiod_set_debounce(struct gpio_desc *desc, unsigned debounce);   /* 较新的接口，使用GPIO描述符（struct
      gpio_desc *） */
```

在驱动中，调用上面两个接口即可设置 gpio 对应的中断 debounce 寄存器；

注意：

1.1.7 版本之前的 pinctrl-sunxi.c 驱动中的 debounce 参数是以 us 为单位；

1.1.7 版本之后的 pinctrl-sunxi.c 驱动中的 debounce 参数是以 ns 为单位；

6 FAQ

6.1 GPIO 中断问题排查步骤

6.1.1 GPIO 中断一直响应

1. 排查中断信号是否一直触发中断。
2. 利用 sunxi_dump 节点，确认中断 pending 位是否没有清 (参考 6.1.1 小节)。
3. 是否在 gpio 中断服务程序里对中断检测的 gpio 进行 pin mux 的切换，不允许这样切换，否则会导致中断异常。

6.1.2 GPIO 检测不到中断

1. 排查中断信号是否正常，若不正常，则排查硬件，若正常，则跳到步骤 2。
2. 利用 sunxi_dump 节点，查看 gpio 中断 pending 位是否置起，若已经置起，则跳到步骤 5，否则跳到步骤 3。
3. 利用 sunxi_dump 节点，查看 gpio 的中断触发方式是否配置正确，若正确，则跳到步骤 4，否则跳到步骤 5。
4. 检查中断的采样时钟，默认应该是 32k，可以通过 sunxi_dump 节点，切换 gpio 中断采样时钟到 24M 进行实验。
5. 利用 sunxi_dump，确认中断是否使能。

6.2 如何使用 PD/PK/PJ 引脚

针对sun55iw3系列产品，若需要使用PD、PK、PJ作为普通GPIO使用，需要在dts中加上以下配置。具体操作如下：

在board.dts中增加节点：

```
&pd1 {
  /* pk */
  pd1_vi@A523_PCK_VI {
    ppu-always-on;
  };
```

```
    /* pd */
    pd1_vo0@A523_PCK_VO0 {
        ppu-always-on;
    };

    /* pj */
    pd1_vo1@A523_PCK_VO1 {
        ppu-always-on;
    };
};
```

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。