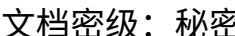
D1 Linux GPIO
开发指南

版本号: 1.0
发布日期: 2021.04.26

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.26	XAA0191	创建文档



目　录

1 概述 1
1.1 编写目的 1
1.2 适用范围 1
1.3 相关人员 1

2 模块介绍 2
2.1 模块功能介绍 2
2.2 相关术语介绍 2
2.3 总体框架 3
2.4 Pinctrl framework 简介 4
2.5 源码结构介绍 4

3 模块配置 6
3.1 kernel menuconfig 配置 6
3.2 device tree 源码结构和路径 8
3.2.1 SoC 级配置 8
3.2.2 board.dts 板级配置 9

4 模块接口说明 11
4.1 pinctrl 接口说明 11
4.1.1 pinctrl_get 11
4.1.2 pinctrl_put 11
4.1.3 devm_pinctrl_get 12
4.1.4 devm_pinctrl_put 12
4.1.5 pinctrl_lookup_state 12
4.1.6 pinctrl_select_state 13
4.1.7 devm_pinctrl_get_select 13
4.1.8 devm_pinctrl_get_select_default 13
4.1.9 pinctrl_gpio_set_config 14
4.2 gpio 接口说明 14
4.2.1 gpio_request 14
4.2.2 gpio_free 15
4.2.3 gpio_direction_input 15
4.2.4 gpio_direction_output 15
4.2.5 __gpio_get_value 16
4.2.6 __gpio_set_value 16
4.2.7 of_get_named_gpio 16
4.2.8 of_get_named_gpio_flags 17

5 使用示例 18
5.1 使用 pin 的驱动 dts 配置示例 18
5.1.1 配置通用 GPIO 功能/中断功能 18

ALLWINNER®

5.1.2 PIN 的特殊功能配置 19
5.2 接口使用示例 20
5.2.1 配置设备引脚 20
5.2.2 获取 GPIO 号 20
5.3 设备驱动使用 GPIO 中断功能 20
5.4 设备驱动设置中断 debounce 功能 21

6 FAQ **23**
6.1 常用 debug 方法 23
6.1.1 利用 sunxi_dump 读写相应寄存器 23
6.1.2 利用 sunxi_pinctrl 的 debug 节点 23
6.1.3 利用 pinctrl core 的 debug 节点 25
6.1.4 GPIO 中断问题排查步骤 27
6.1.4.1 GPIO 中断一直响应 27
6.1.4.2 GPIO 检测不到中断 27

版权所有 © 珠海全志科技股份有限公司。保留一切权利

插　图

2-1 pinctrl 驱动整体框架图 3
2-2 pinctrl 驱动 framework 图 4
3-1 内核 menuconfig 根菜单 6
3-2 内核 menuconfig device drivers 菜单 7
3-3 内核 menuconfig pinctrl drivers 菜单 7
3-4 内核 menuconfig allwinner pinctrl drivers 菜单 8
6-1 查看 pin 配置图 24
6-2 修改结果图 24
6-3 pin 设备图 25
6-4 pin 设备图 25

1 概述

1.1 编写目的

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 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
D1	Linux-5.4	pinctrl-sunxi.c

1.3 相关人员

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。

ALLWINNER®

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 模块功能介绍

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 相关术语介绍

表 2-1: 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
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可以有个确定的值

版权所有 © 珠海全志科技股份有限公司。保留一切权利

术语	解释说明
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的习惯
Device tree	犹如它的名字，是一颗包括 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

2.3 总体框架

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 interfaces、pinctrl common frame、sunxi pinctrl driver 以及 board configuration。

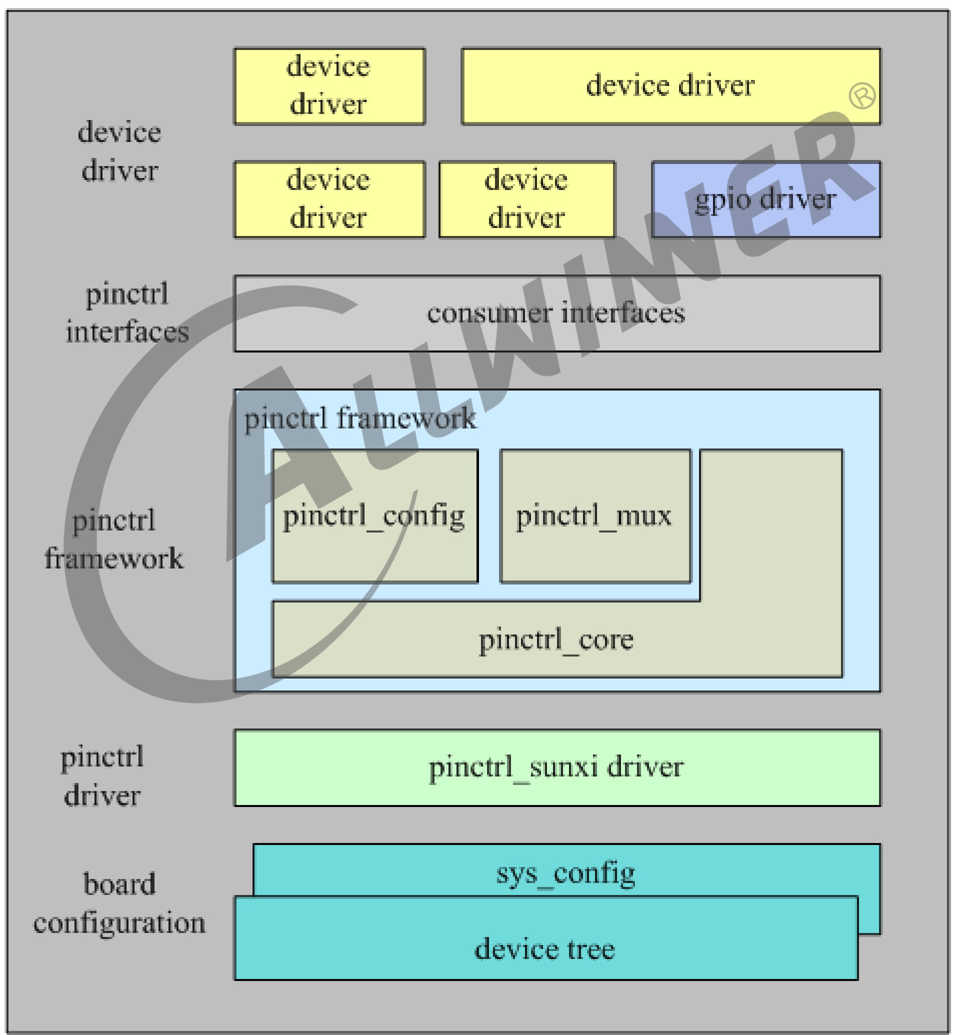


图 2-1: pinctrl 驱动整体框架图

Pinctrl interfaces: 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（dts）的方式进行配置。

2.4 Pinctrl framework 简介

Pinctrl framework 主要处理 Pinctrl Core、Pinctrl mux 和 Pinctrl conf 三个功能，三者映射关系如下图所示。

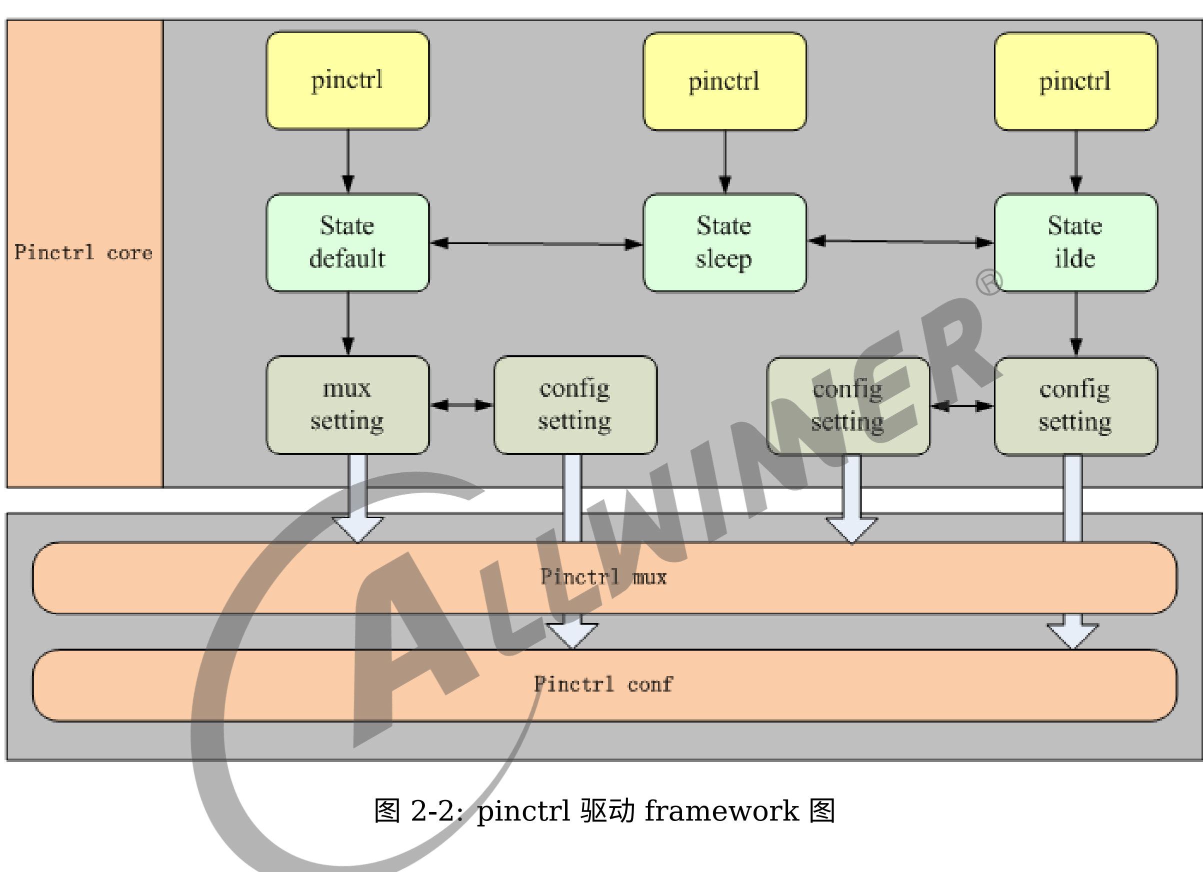


图 2-2: pinctrl 驱动 framework 图

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。

2.5 源码结构介绍

```
linux
|
|-- drivers
|     |-- pinctrl
|     |        |-- Kconfig
|     |        |-- Makefile
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
|     |        |-- core.c
|     |        |-- core.h
|     |        |-- devicetree.c
|     |        |-- devicetree.h
|     |        |-- pinconf.c
|     |        |-- pinconf.h
|     |        |-- pinmux.c
|     |        `-- pinmux.h
|     `-- sunxi
|             |-- pinctrl-sunxi-test.c
|             |-- pinctrl-sun8iw20.c
|
`-- include
     `-- linux
          `-- pinctrl
                  |-- consumer.h
                  |-- devinfo.h
                  |-- machine.h
                  |-- pinconf-generic.h
                  |-- pinconf.h
                  |-- pinctrl-state.h
                  |-- pinctrl.h
                  `-- pinmux.h
```

3 模块配置

3.1 kernel menuconfig 配置

进入 tina 开发环境根目录执行：

```
source build/envsetup.sh ----配置tina环境变量
lunch                    ----选择d1_nezha
make kernel_menuconfig   ----进入内核配置主界面
```

首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

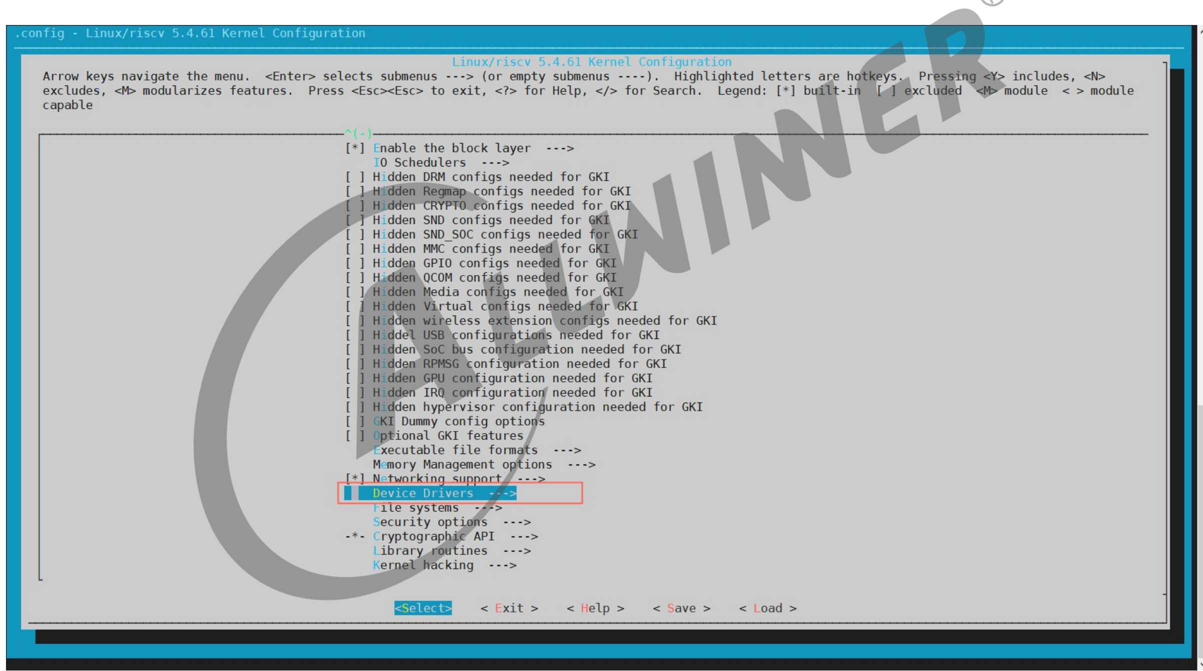


图 3-1: 内核 menuconfig 根菜单

选择 Pin controllers, 进入下级配置，如下图所示：

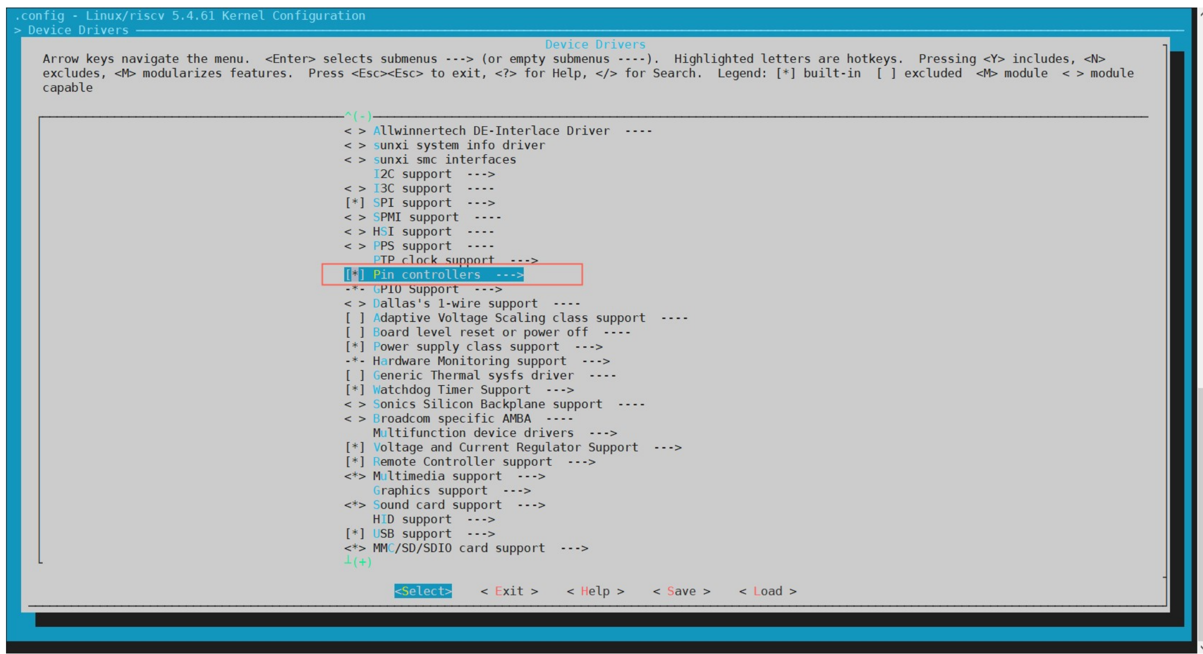


图 3-2: 内核 menuconfig device drivers 菜单

选择 Allwinner SoC PINCTRL DRIVER, 进入下级配置，如下图所示：

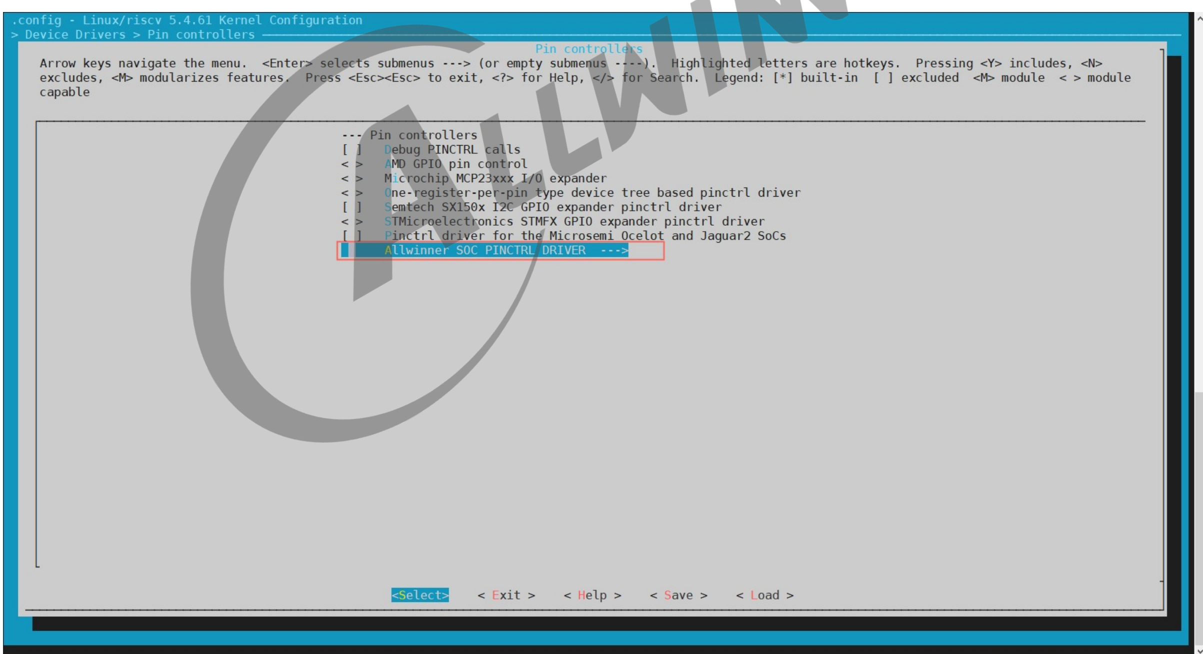


图 3-3: 内核 menuconfig pinctrl drivers 菜单

Sunxi pinctrl driver 选择 sun8iw20 PIO controller（默认已编译进内核），如下图所示：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

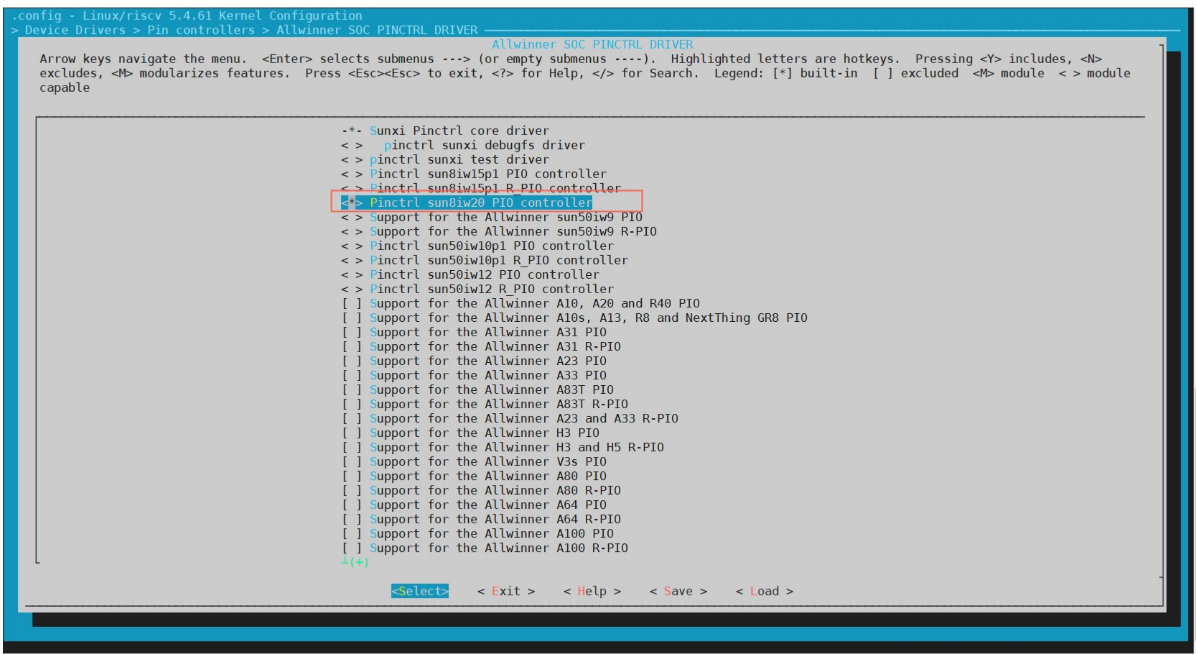


图 3-4: 内核 menuconfig allwinner pinctrl drivers 菜单

3.2 device tree 源码结构和路径

- SoC 级设备树路径：

```
kernel/arch/riscv/boot/dts/sunxi/sun20iw1p1.dtsi
```

- 板级设备树 (board.dts) 路径：

```
device/config/chips/d1/configs/nezha/board.dts
```

- device tree 的源码包含关系如下：

```
board.dts
  `--------sun20iw1p1.dtsi
```

板级以及 SoC 级的设备树文件共同描述了驱动的设备信息。下面分别对其进行表述。

3.2.1 SoC 级配置

在 kernel/arch/riscv/boot/dts/sunxi/sun20iw1p1.dtsi 文件中，配置了该 SoC 的 pinctrl 控制器的通用配置信息，一般不建议修改。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    pio: pinctrl@2000000  {
      compatible = "allwinner,sun20iw1-pinctrl"; /* 兼容属性，用于驱动和设备绑定 */
      reg = <0x0 0x02000000 0x0 0x500>; /* 寄存器基地址0x02000000和范围0x500 */
      interrupts-extended = <&plic0 85 IRQ_TYPE_LEVEL_HIGH>,
             <&plic0 87 IRQ_TYPE_LEVEL_HIGH>,
             <&plic0 89 IRQ_TYPE_LEVEL_HIGH>,
             <&plic0 91 IRQ_TYPE_LEVEL_HIGH>,
             <&plic0 93 IRQ_TYPE_LEVEL_HIGH>,
             <&plic0 95 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 = <&ccu CLK_APB0>, <&dcxo24M>, <&rtc_ccu CLK_OSC32K>;
      clock-names = "apb", "hosc", "losc";
      gpio-controller; /* 表示是一个gpio控制器 */
      #gpio-cells = <3>; /* gpio属性需要配置的参数个数 */
      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>;
      vcc-pf-supply = <&reg_pio1_8>;
      vcc-pfo-supply = <&reg_pio3_3>;
    }
```

3.2.2 board.dts 板级配置

board.dts 用于保存每个板级平台的设备信息，以 nezha 板为例，board.dts 路径如下：

device/config/chips/d1/configs/nezha/board.dts

在 board.dts 中的配置信息如果在 sun20iw1p1.dtsi 中存在，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 sun20iw1p1.dtsi 中的配置信息。
- board.dts 中新增加的属性和结点，会追加到最终生成的 dtb 文件中。

说明

建议 *PIN* 的配置和引用都放到 *board.dts* 中，不要在 *dtsi* 中修改。

以 UART 模块的配置过程为例，board.dts 的配置如下：

```
&pio{
    input-debounce = <0 0 0 0 1 0 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
    */

  uart0_pins_a: uart0_pins@0 {  /* For nezha board */
    pins = "PB8", "PB9";
    function = "uart0";
    drive-strength = <10>;
    bias-pull-up;
  };
};

&uart0 {
```

```
14     pinctrl-names = "default", "sleep";
15     pinctrl-0 = <&uart0_pins_a>;
16     pinctrl-1 = <&uart0_pins_b>;
17     status = "okay";
18 };
```

4 模块接口说明

4.1 pinctrl 接口说明

4.1.1 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。

4.1.2 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。

⚠ **警告**

必须与 `pinctrl_get` 配对使用。

4.1.3 devm_pinctrl_get

- 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。

4.1.4 devm_pinctrl_put

- 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。

⚠ **警告**

必须与 `devm_pinctrl_get` 配对使用，可以不显示调用该接口。

4.1.5 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。

4.1.6 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。

4.1.7 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。

4.1.8 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。

4.1.9 pinctrl_gpio_set_config

- 函数原型：`int pinctrl_gpio_set_config(unsigned gpio, unsigned long config)`
- 作用：设置一个 gpio 的相关属性。
- 参数：
  - gpio:gpio 子系统的 gpio 编号。
  - config：gpio 属性的配置。
- 返回：
  - 成功，返回 0，否则表示失败

说明

***config* 属性可以使用 *pinconf_to_config_pack* 函数进行封装，包括以下配置类型：**

- **上下拉：*PIN_CONFIG_BIAS_PULL_UP/PIN_CONFIG_BIAS_PULL_DOWN/PIN_CONFIG_BIAS_DISABLE***
- **驱动能力：*PIN_CONFIG_DRIVE_STRENGTH***

4.2 gpio 接口说明

4.2.1 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。

4.2.2 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。

4.2.3 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。

4.2.4 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。

4.2.5 __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。

4.2.6 __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

4.2.7 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。

4.2.8 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:enum of_gpio_flags 类型变量，包含普通 IO 配置、上下拉配置、驱动能力配置等信息。
- 返回：
  - 成功，返回 gpio 编号。
  - 失败，返回错误码。

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 GPIO 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

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 = <&pio PB 4 GPIO_ACTIVE_HIGH>;
            linux,input-type = <1>;
            linux,code = <0x1c>;
            wakeup-source = <0x1>;
        };
    };
    ...
};
```

通用 GPIO 功能以及中断功能采用 dts 的配置方法，配置参数解释如下：

```
gpios = <&pio PB  4 GPIO_ACTIVE_HIGH>;
          |    |   |    |
          |    |   |    `-----------active时状态。可用GPIO_PULL_UP、GPIO_PULL_DOWN配置上下拉
          |    |   `----------------当前bank中哪个引脚
          |    `--------------------哪个bank
          `-------------------------指向哪个pio
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

使用上述方式配置 gpio 时，需要驱动调用以下接口解析 dts 的配置参数：

```
int of_get_named_gpio_flags(struct device_node *np, const char *list_name, int index, enum
    of_gpio_flags *flags)
```

拿到 gpio 的配置信息后 (保存在 flags 参数中，见 4.2.8 小节)，再根据需要调用相应的标准接口实现自己的功能。

5.1.2 PIN 的特殊功能配置

device tree 配置 demo 如下所示：

```
device tree对应配置
soc{
    &pio {
        ...
    uart0_pins_a: uart0_pins@0 {  /* For nezha board */
      pins = "PB8", "PB9";
      function = "uart0";
      drive-strength = <10>;
      bias-pull-up;
    };
    uart0_pins_b: uart0_pins@1 {  /* For nezha board */
      pins = "PB8", "PB9";
      function = "gpio_in";
    };
        ...
    };
    ...
  &uart0 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&uart0_pins_a>;
    pinctrl-1 = <&uart0_pins_b>;
    status = "okay";
  };
    ...
};
```

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。

```
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;
```

5.2.2 获取 GPIO 号

```
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)) {
        dev_err(dev, "Error getting vdevice_3\n");
}
```

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。

```

/* 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;
}
/* 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)) {
```

```
13         pr_warn("request virq %d failed, errno = %d\n", virq, ret);
14         return -EINVAL;
15     }
```

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：

```
1  dts配置如下:
2  soc {
3    ...
4    Vdevice: vdevice@0 {
5      compatible = "allwinner,sun8i-vdevice";
6      device_type = "Vdevice";
7      interrupt-parent = <&pio>;                /* 依赖的中断控制器(带interrupt-controller属性的结
       点) */
8      interrupts = < PD 3 IRQ_TYPE_LEVEL_HIGH>;
9                       | |      |------------------中断触发条件、类型
10                      | |-------------------------pin bank内偏移
11                      |---------------------------哪个bank
12     status = "okay";
13   };
14   ...
15 };
```

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

```
1      ....
2      /* 解析DTS中配置的中断号 */
3      irq = platform_get_irq(pdev, 0);
4      if (irq < 0) {
5          printk("Get irq error!\n");
6          return -EBUSY;
7      }
8      /* 申请中断 */
9      ret = request_irq(irq, sunxi_pinctrl_irq_handler,
10                     IRQF_TRIGGER_HIGH, "PIN_EINT", NULL);
11     if (IS_ERR(ret)) {
12         pr_warn("request irq failed !\n");
13         return -EINVAL;
14     }
15 }
```

5.4 设备驱动设置中断 debounce 功能

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

```
1  &pio {
2      /* takes the debounce time in usec as argument */
3    input-debounce = <0 0 0 0 0 0 >;
4                      | | | | | |
5                      | | | | | `------------PG bank
6                      | | | | `--------------PF bank
```

```
                    | | | `----------------PE bank
                    | | `------------------PD bank
                    | `--------------------PC bank
                    `----------------------PB 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 usec）。

6 FAQ

6.1 常用 debug 方法

6.1.1 利用 sunxi_dump 读写相应寄存器

需要开启 SUNXI_DUMP 模块：

```
make kernel_menuconfig

---> Device Drivers
  ---> dump reg driver for sunxi platform (选中)
```

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。
```

6.1.2 利用 sunxi_pinctrl 的 debug 节点

需要开启 DEBUG_FS：

```
make kernel_menuconfig

---> Kernel hacking
  ---> Compile-time checks and compiler options
    ---> Debug Filesystem (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. 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 6-1: 查看 pin 配置图

2. 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 6-2: 修改结果图

ALLWINNER

修改结果如下图所示：

```
/sys/kernel/debug/sunxi_pinctrl # echo r_pio > dev_name ;cat dev_name
r_pio
/sys/kernel/debug/sunxi_pinctrl # echo pio > dev_name ;cat dev_name
pio
/sys/kernel/debug/sunxi_pinctrl #
```

图 6-3: pin 设备图

6.1.3 利用 pinctrl core 的 debug 节点

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

cd /sys/kernel/debug/sunxi_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 6-4: pin 设备图

2. 查看 pin 的状态和对应的使用设备

```
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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    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。

6.1.4 GPIO 中断问题排查步骤

6.1.4.1 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

6.1.4.2 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

ALLWINNER

版权所有 © 珠海全志科技股份有限公司。保留一切权利

著作权声明

版权所有 ©2021 珠海全志科技股份有限公司。保留一切权利。

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。