ALLWIMER®

Pinctrl 接口使用说明文档

文档版本号: SDK-V1.0 大概 5 年 5 日期: 2017

版权所有©珠海全志科技股份有限公司 2019。保留一切权利。

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。

#### 商标声明

ALLWIMER 全志和其他全志商标均为珠海全志科技股份有限公司的商标。

本文档提及的其他所有商标或注册商标, 由各自的所有人拥有。

#### 注意

您购买的产品、服务或特性等应受全志公司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的购买或使用范围之内。除非合同另有约定,全志 司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因,本文档内容会不定期进行更新。除非另有约定,本文档仅 作为使用指导,本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

、任何与

ALLWIMER	Pinctrl 接口使用说明文档		A THE REAL PROPERTY OF THE PERSON OF THE PER	
	Y. T.	文档履	历	Y. T.
版本号	日期	制/修订人	内容描述	
V1.0	2019-03-30	Allwinner	V316 初始化版本	

The state of the s

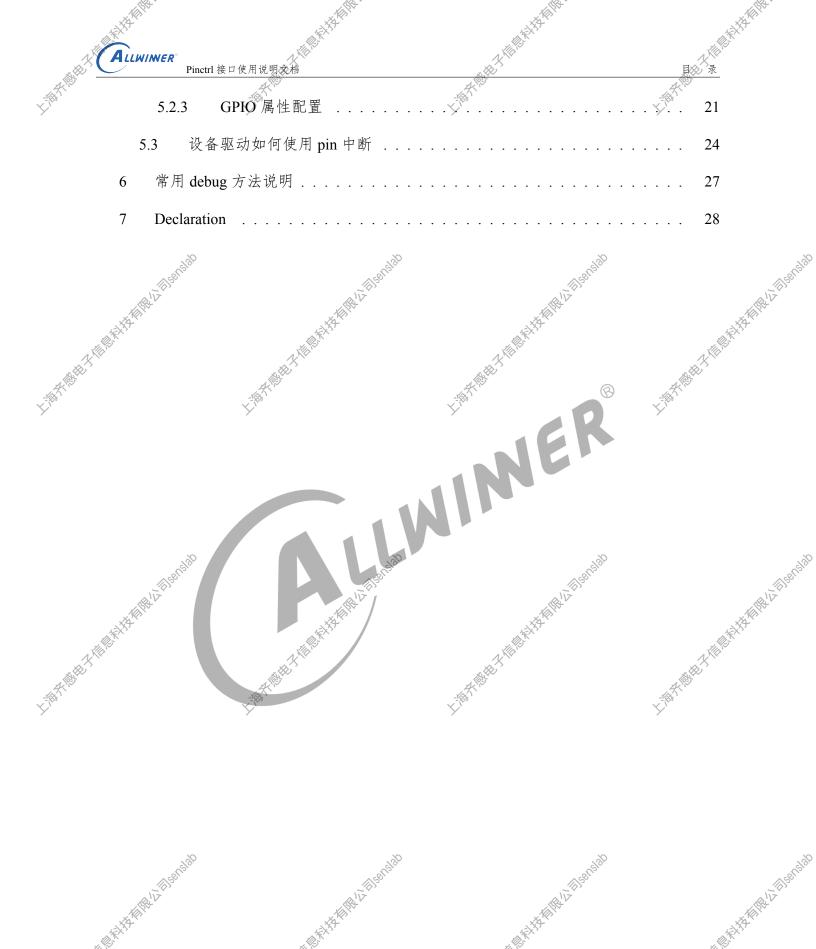


# 目 录

1	概述	
	1.1	編写目的       1         适用范围       1
J1	1.2	編写目的   1     适用范围   1
No. XX	1.3	相关人员
2	模块	介绍
	2.1	模块功能介绍
	2.2	相关术语介绍
	2.3	模块配置介绍
	2.4	源码结构介绍
3	驱动	雅架9
<u>.</u> ./1	3. T	总体框架
W. XY	3.2	state/pinmux/pinconfig
4	外部	接口
	4.1	pinetrl 11
	4.1.1	l pinctrl_get
	4.1.2	2 pinctrl_put
	4.1.3	devm_pinctrl_get
	4.1.	devm_pinctrl_put
<u>, 1</u>	4.1.5	pinctrl_lookup_state 12
W. XA	4.1.6	pinctrl_select_state

Pinctrl 接口使用说明文本	<u>"</u>	>
1 1110 (11 1× 1 1× 1) (1 1/1 1/1 1/1 1/1 1/1 1/1 1/1 1/1 1/1	- J. V.V.	

	4.1.7	devm_pinctrl_get_select			. 12
	4.1.8	devm_pinctrl_get_select_defaul	lt		. 13
	4.1.9	pin_config_get			. 13
	4.1.10	pin_config_set			. 13
	4.1.1	pin_config_group_get			. 14
x	4.1.12	pin_config_group_set		\$ 12	. 14
	4.2 gpi				. 14
	4.2.1	gpio_request			14
	4.2.2	gpio_free		· · · · · · · · · · · · · · · · · · ·	. 14
	4.2.3	gpio_direction_input			. 15
	4.2.4	gpio_direction_output			. 15
	4.2.5	gpio_get_value	<b>N.J.</b>		. 15
	4.2.6	_gpio_set_value			. 16
	4.2.7	of_get_named_gpio		\$\frac{1}{10} \rightarrow	. 16
	4.2.8	of_get_named_gpio_flags		,	. 16
5	使用例子	F			17
	5.1 配量	<u> </u>			. 17
	5.1.1	场景一			. 17
	5.1.2	场景二			. 18
	5.1.3	场景三			. 19
	5.2 接口	口使用示例			. 20
w	5.2.1	配置设备引脚		\$\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	. 20
A KXX	5.2.2	获取 GPIO 号		,	. 21



表目录

A Hilling High Land State of the State of th



图目录

2-1	figure1.png		4
2-2	figure2.png		5
2-3	figure3.png		6
2-4	figure4.png		7
3-1	figure6.png		9 Arrivan
3-2	figure5.png	· · · · · · · · · · · · · · · · · · ·	10

THE THE REAL PROPERTY OF THE PARTY OF THE PA

A. H. VIII SEE S.

A THE THE PARTY OF THE PARTY OF

ENA PROPERTY OF SERVICES AND SE





#### 1.1 编写目的

平台上 本文档对 Linux4.9 内核的 GPIO 接口使用进行详细的阐述,让用点明确掌握 GPIO 配 置、申请等操作的编程方法。

## 适用范围

本文档适用于 linux4.9 内核, 所有平台。

#### 1.3 相关人员

本文档适用于所有需要在Linux4.9内核 sunxi 平台上开发设备驱动的人。



# 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

sunxi: Allwinner 的 SOC 硬件平台

Pincontroller: 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,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"云态"(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。

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。

Script 脚本: 指的是打包到 img 中的 sys\_config.fex 文件. 包含系统各模块配置参数。

#### 2.3 模块配置介绍

在命令行中进入内核根目录, 执行 make ARCH=arm(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:

A THE THE REAL PROPERTY OF THE PARTY OF THE



```
Linux/arm 4.9.118 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes,
<N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < >
       General setup --->
    [*] Enable loadable module support --->
    [*] Enable the block layer --->
        System Type --->
       Bus support --->
        Kernel Features --->
       Boot options --->
       CPU Power Management
       Floating point emulation --->
       Userspace binary formats
       Power management options --->
   [*] Networking support --->
     Device Drivers --->
       Firmware Drivers -
       File systems --->
       Kernel hacking --->
       Security options --->
    -*- Cryptographic API --->
                    < Exit >
        <Select:
                                                        < Load >
```

图 2-1: figure1.png

选择 Pin controllers, 进入下级配置, 如下图所示:



```
config - Linux/arm 4.9.118 Kernel Configuration
Device Drivers
                                Device Drivers
  Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
  submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes,
  <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for
  Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < >
      [*] Block devices --->
      < > NVMe Target support
          Misc devices --->
          SCSI device support --->
      Serial ATA and Parallel ATA drivers (libata) ----
      [ ] Multiple devices driver support (RAID and LVM)
      [*] Network device support --->
      [ ] Open-Channel SSD target support
          Input device support --->
          Character devices --->
          I2C support
      [*] SPI support --->
      < > SPMI support ----
      < > HSI support ----
          PPS support --->
          PTP clock support
         Pin controllers --->
      [*] GPIO Support --->
           <Select>
                       < Exit >
                                   < Help
                                                           < Load >
```

图 2-2: figure2.png

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



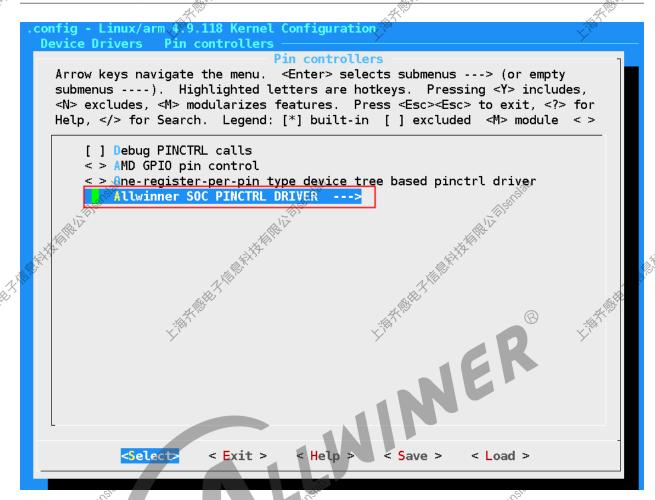


图 2-3: figure3.png

Sunxi pinctrl driver 默认编译进内核,如下图(以 sun8iw16 平台为例,其他平台类似)所示:



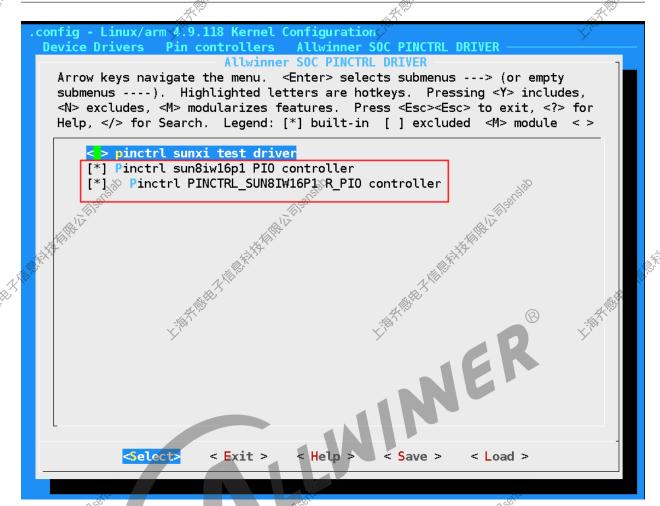


图 2-4: figure4.png

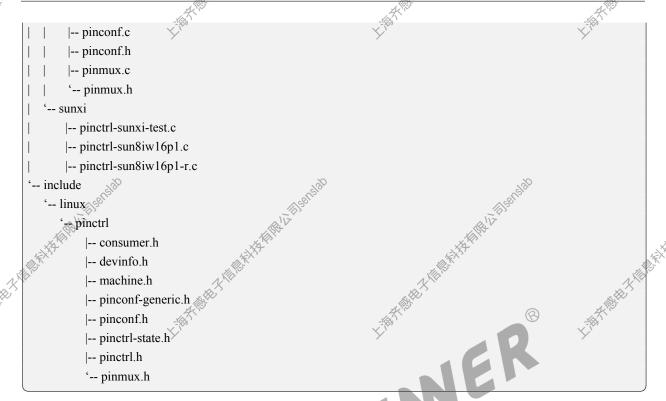
# 2.4 源码结构介绍

以 sun8iw16 平台为例, 其他平台类似:

```
|-- drivers
| |-- pinctrl
| | |-- Kconfig
| | |-- Makefile
| | |-- core.c
| | |-- core.h
| |-- devicetree.c
```



Pinctrl 接口使用说明文档 模块介绍



Light Hall the house of the state of the sta

A STANSON OF STANSON O

A HELLE LAND



3

# 驱动框架

#### 3.1 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 api、pinctrl common frame、sunxi pinctrl driver and board configuration。

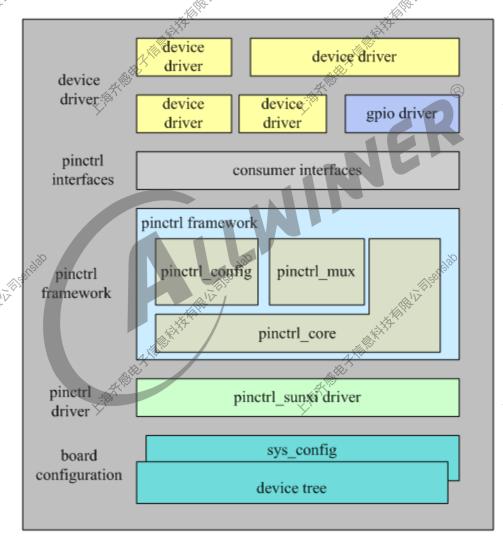


图 3-1: figure6.png

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,格式 device tree source 或者 sys config.

#### 3.2 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。

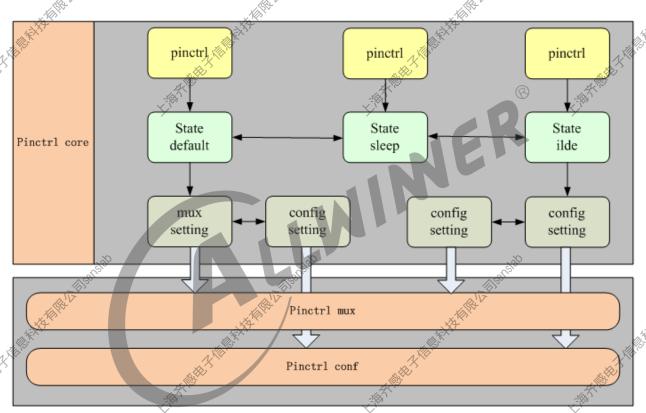


图 3-2: figure5.png

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。





#### 4.1 pinctrl

#### 4.1.1 pinctrl\_get

原型: struct pinctrl \*pinctrl get(struct device \*dev);

功能: 获取设备的 pin 操作句柄, 所有 pin 操作必须基于此 pinctrl 句柄;

.ip 息位 输入: 使用 pin 的设备, pinctrl 子系统通过设备名与 pin 配置信息匹配, 信息。

输出: pinctrl 句柄。

#### 4.1.2 pinctrl put

原型。void pinctrl put(struct pinctrl \*p);

功能:释放 pinctrl 句柄,必须与 pinctrl\_get 配对使用。

输入: pinctrl 句柄。

输出:无。

#### 4.1.3 devm pinctrl get

原型: struct pinctrl \*devm pinctrl get(struct device \*dev);

功能: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。

输入: 使用 pin 的设备, pinetrl 子系统通过设备名与 pin 配置信息匹配, 获取 pin 配置



输出: pinctrl 句柄。

#### 4.1.4 devm\_pinctrl\_put

原型: void devm pinctrl put(struct pinctrl \*p);

功能: 释放 pinctrl 句柄, 必须与 devm pinctrl get 配对使用。

输入: pinctrl 句柄。

输出:无。

#### 4.1.5 pinctrl\_lookup\_state

原型: struct pinctrl\_state \*pinctrl\_lookup\_state(struct pinctrl \*p, const char \*name)

功能:根据 pin 操作句柄,查找 state 状态句柄;

输入: pin 句柄 state name

输出: state 状态句柄。

# 4.1.6 pinctrl\_select\_state

原型: int pinctrl\_select\_state(struct pinctrl \*p, struct pinctrl\_state \*s)

功能:将 pin 句柄对应的 pinctrl 设置为 state 句柄对应的状态;

输入: pin 句柄 state 句柄

输出: state 设置结果, 0-成功, 其他-失败。

#### 4.1.7 devm\_pinctrl\_get\_select

原型: struct pinctrl \*devm pinctrl get select(struct device \*dev, const char \*name)

功能:获取设备的 pin 操作句柄,并将句柄设定为指定状态;



输入: 使用 pin 的设备, pinctrl 子系统通过设备名与 pin 配置信息匹配, state name

输出: pinctrl 句柄。

#### 4.1.8 devm pinctrl get select default

原型: struct pinctrl \*devm pinctrl get select default(struct device \*dev) &

功能: 获取设备的 pin 操作句柄, 并将 pin 句柄对应的 pinctrl 设置为 default 状态;

输入:使用 pin 的设备, pinetrl 子系统会通过设备名与 pin 配置信息匹配;

输出: pinctrl 句柄。

#### 4.1.9 pin\_config\_get

原型: int pin\_config\_get(const char \*dev\_name, const char \*name, unsigned long \*config)

功能: 获取指定 pin 的属性;

输入: pinctrl 名称 pin 名称 pin 配置属性

输出: 获取 pin 属性结果, 0-成功, 其他 -失败。

#### 4.1.10 pin config set

原型: int pin config set(const char \*dev\_name, const char \*name, unsigned long config)

功能:设置指定 pin 的属性;

输入: pinctrl 名称 Pin 名称 pin 配置属性

输出:设置 pin 属性结果, 0-成功, 其他 -失败。

Page 13 of 28

Copyright © 2019 by Allwinner. All rights reserved



#### 4.1.11 pin\_config\_group\_get

原型: int pin\_config\_group\_get(const char \*dev\_name, const char \*pin\_group,unsigned long \*config)

功能: 获取指定 group 的属性;

输入: pinctrl 名称 group 名称 pin 配置属性

输出: 获取 group 属性结果, 0-成功, 其他-失败。

#### 4.1.12 pin\_config\_group\_set

原型: int pin\_config\_group\_set(const char \*dev\_name, const char \*pin\_group,unsigned long config)

功能:设置指定 group 的属性;

输入: pinctrl 名称 group 名称 pin 配置属性

输出:设置 group 属性结果, 0-成功, 其他-失败。

## 4.2 gpio

#### 4.2.1 gpio request

原型: int gpio\_request(unsigned gpio, const char \*label)

功能: 申请 gpio. 获取 gpio 的访问权.

参数: gpio: gpio 编号. label: gpio 名称, 可以为 NULL.

输出: 0 表示成功, 否则表示失败.

#### 4.2.2 gpio\_free

原型: void gpio\_free(unsigned gpio)



功能: 释放 gpio.

参数: gpio: gpio 编号.

输出: 无.

#### 4.2.3 gpio\_direction\_input

原型: int gpio direction input(unsigned gpio)

功能:将gpio设置为input。

参数: gpio: gpio 编号

输出: 0表示成功, 否则表示失败.

#### 4.2.4 gpio direction output

Te) 原型: int gpio direction output(unsigned gpio, int value)

功能:将gpio设置为output,并设置电平值.

参数: gpio: gpio 编号. value: gpio 电平值, 非 0 表示高, 0 表示低。

输出: 0表示成功, 否则表示失败

#### 4.2.5 gpio get value

原型: int gpio get value(unsigned gpio)

功能: 获取 gpio 电平值. (gpio 已为 input/output 状态)

参数: gpio: gpio 编号.

输出: gpio 电平,1表示高,0表示低.

Copyright © 2019 by Allwinner. All rights reserved



#### 4.2.6 \_\_gpio\_set\_value

原型: void \_\_gpio\_set\_value(unsigned gpio, int value)

功能:设置 gpio 电平值. (gpio 已为 output 状态)

参数: gpio: gpio 编号. value: gpio 电平值, 非 0 表示高, 0 表示低.

输出: 无

#### 4.2.7 of get\_named\_gpio

原型: int of\_get\_named\_gpio(struct device\_node \*np; const char \*propname) int index

功能: 通过名称获取 GPIO 索引号

参数: np: 要获取 GPIO 信息的节点 Propname: 节点中包含 gpio 描述信息的属性. Index: 所要查找的 gpio 在名称为 propname 的属性中的索引号.

输出: GPIO 号

#### 4.2.8 of get\_named gpio flags

原型:int of\_get\_named\_gpio\_flags(struct device\_node \*np, const char \*list\_name, int index.enum of\_gpio\_flags \*flags)

功能:通过名称获取 GPIO 索引号,并通过 flags 获取 dts 配置信息

参数: np: 要获取 GPIO 信息的节点 Propname: 节点中包含 gpio 描述信息的属性. Index: 所要查找的 gpio 在名称为 propname 的属性中的索引号. 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.

输出: GPIO 号.



# 使用例子

#### 5.1 配置

总结 linux-4.9 平台上 sys\_config.fex 的配置,用户在主键中的管脚配置主要有以下几 种情形、针对这几种情形,文档描述了在 device tree 配置文件和 sws\_config.fex 文件中,用 户如何实现对应配置.

- 用户只配置通用 GPIO, 即用来做输入、输出、中断
- INIER ● 用户只配置设备管脚,如 Uart 设备的引脚、LCD的引脚等;
- 用户既要配置通用 GPIO, 也要配置设备引脚;

#### 5.1.1 场景一

场景一: 用户只需要配置 GPIO, 中断, device tree 配置 demo 如下所示:

	V2,	
sys_config.配置fex: [gpiokey] compatible = "gpio-keys" [gpiokey/ok_key]	上海·林···································	
compatible = "gpio-keys"	A THE PARTY OF THE	
A SECTION AND A	A. T. C.	
	A CONTRACTOR OF THE PARTY OF TH	4.125
label = "ok_key"	- T. S	-1634 ,
gpios = port:PL04<0> <default><default>&lt;1&gt;</default></default>	,	,
电平		
驱动能力		
上下拉		
复用类型		
pin 内偏移bank		
哪个 bank		
linux,input-type = <1>	.%	
linux,code = 28	a series	
wakeup-source = 1等价于	A Hillight HA HILLIGH BELEVE TO SEE THE SEE TH	
device 对应配置tree	A KXX	
soc {	N. Miller	
Assoc (	***	



```
gpiokey {
 device_type = "gpiokey";
 compatible = "gpio-keys";
 ok key {
  device_type = "ok_key";
  label = "ok key";
  gpios = <&rpio PL 0x4 0x0 0xffffffff 0xffffffff 0x1%
  linux,input-type = "1>";
  linux,code = <0x1c>;
  wakeup-source = <0x1>;
};说明:
gpio in/gpio out/采用的配置方法,配置参数解释如下: interruptdts
gpios = <&r_pio PL 0x4 0x0 0xffffffff 0xffffffff 0x1>;
                      电平|-----
                  上下拉, 值为|-----0时采样默认值xffffffff
                                           ---0时采样默认值xffffffff
         | | 驱动能力,值为 |------
     | | | 复用类型|-----
                               ----pin 内偏移bank
     指向哪个|-----
                                       -----,属于要用piocpus&r pio使用企述方式配置时,需要调
    用以下接口解析的配置参数:
gpiodts
int of get named gpio flags(struct device node *np, const char *list name, int index, enum of gpio flags *flags) $
    到的配置信息后
gpio保存在(参数中,见flags4小节.2.8.),在根据需要调用相应的标准接口实现自己的功能
```

#### 5.1.2 场景二

场景二: 用户只需要配置设备引脚, device tree 配置 demo 如下所示:

```
[uart0]

uart_used = 1

uart_port = 0

uart_type = 2

uart_tx = port:PH07<3><1><default><default>

uart_rx = port:PH08<3><1><default><default>
```



```
uart0 regulator = "vcc-io"等价于
device 对应配置tree
  pio: pinctrl@0300b000 {
  uart0_pins_a: uart0@0 {
        allwinner,pins = "PH7", "PH8";
        allwinner,pname = "uart0 tx", "uart0 rx";
        allwinner, function = "uart0";
        allwinner, muxsel = <3>;
        allwinner,drive = <0xffffffff>;
        allwinner,pull = <0xfffffff5;
       };
                                                               MER
  ; }
  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>
   status = "okay";
};
```

#### 5.1.3 场景三

场景三: 用户既要配置通用 GPIO, 也要配置设备引脚, device tree 配置 demo 如下:

```
sys_config.配置fex:
[Vdevice]
Vdevice_used = 1
Vdevice_0 = port:PC00<5><1>2><default>
```



```
Vdevice 1
              = port:PC01<5><1><2><default>
             = port:PC03<1><2><2><1>对应配置:
test-gpios
device tree
soc{
 pio: pinctrl@01c20800 {
     vdevice pins a: vdevice@0 {
       allwinner,pins = "PC0", "PC1";
       allwinner, function = "vdevice";
       allwinner, muxsel = <5>;
       allwinner,drive = <1>;
       allwinner,pull = <1>;
                                          vdevie: vdevie@0{
  pinctrl-names = "default";
  pinctrl-0 = <&vdevice_pins_a>;
  test-gpios = <&pio PC 3 1 2 2 1>;
};
```

# 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;
    pr_warn("device [%s] probe enter\n", dev_name(&pdev->dev));
    /* request device pinctrl, set as default state */
    pinctrl = devm_pinctrl_get_select_default(&pdev->dev);

if (IS_ERR_OR_NULL(pinctrl)) {
```

```
pr_warn("request pinctfl handle for device [%s] failed\n",
    dev_name(&pdev->dev));
    return -EINVAL;
}
pr_debug("device [%s] probe ok\n", dev_name(&pdev->dev));
return 0;
}
```

#### 5.2.2 获取 GPIO 号

#### 5.2.3 GPIO 属性配置

通过 pin\_config\_set/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);
   pr_warn("find device [%s] failed...\n", sunxi_ptest_data->dev_name);
   return -EINVAL;
node = of_find_node_by_type(NULL_dev_name(dev));
   pr warn("find node for device [%s] failed...\n", dev name(dev)):
   return -EINVAL;
 dev->of node = node;
 pr warn("device[%s] all pin resource we want to request\n", dev name(dev));
 pr warn("-----\n");
 pr warn("step1: request pin all resource.\n");
 pinctrl = devm pinctrl get select default(dev);
 if (IS ERR OR NULL(pinctrl)) {
 pr_warn("request pinctrl handle for device [%s] failed...\n", dev name(dev))
   return -EINVAL;
 pr_warn("step2: get device[%s] pin count.\n", dev_name(dev));
 ret = dt get gpio list(node, &gpio list, &gpio count);
 if (ret < 0 \parallel gpio count == 0) {
   pr warn(" devices own 0 pin resource or look for main key failed!\n");
   return -EINVAL;
 }
 pr warn("step3: get device[%s] pin configure and check.\n", dev name(dev));
 for (gpio index = 0; gpio index < gpio count; gpio index++) {
   gpio_cfg = &gpio_list[gpio_index];
   /*check function config */
   config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_FUNC, 0xFFFF);
   pin_config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
```



```
if (gpio cfg->mulsel != SUNXI PINCFG UNPACK VALUE(config)) {
     pr warn("failed! mul value isn't equal as dt.\n");
     return -EINVAL;
   /*check pull config */
   if (gpio_cfg->pull != GPIO_PULL_DEFAULT) {
     config = SUNXI PINCFG PACK(SUNXI PINCFG TYPE PUD, 0xFFFF);
     pin config get(SUNXI PINCTRL, gpio cfg->name, &config);
     if (gpio cfg->pull != SUNXI PINCFG UNPACK VALUE(config)) {
       pr warn("failed! pull value isn't equal as dt.\n");
       return -EINVAL;
   /*check dlevel config */
   if (gpio cfg->drive != GPIO DRVLVL DEFAULT) {
     config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DRV, 0XFFFF)
     pin config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
     if (gpio_cfg->drive != SUNXI_PINCFG_UNPACK_VALUE(config))
       pr_warn("failed! dlevel value isn't equal as dt.\n");
       return -EINVAL;
   /*check data config */
   if (gpio cfg->data != GPIO DATA DEFAULT) {
     config = SUNXI PINCFG PACK(SUNXI PINCFG TYPE DAT, 0XFFFF);
     pin config_get(SUNXI_PINCTRL, gpio_cfg->name, &config);
     if (gpio_cfg->data != SUNXI_PINCFG_UNPACK_VALUE(config))
       pr warn("failed! pin data value isn't equal as dt.\n");
       return -EINVAL;
 pr warn("-----\n");
 pr warn("test pinctrl request all resource success!\n");
 return 0;
}注: 需要注意,存在和两个设备,域的需要使用
SUNXI PINCTRLSUNXI R PINCTRLpinctrlcpuspinSUNXI R PINCTRL
```



# 5.3 设备驱动如何使用 pin 中断

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 virg;
  int ret;
  /* map the virq of gpio */
  virq = gpio_to_irq(GPIOA(0));
  if (IS ERR VALUE(virg)) {
    pr warn("map gpio [%d] to virq failed, errno = %d\n",
    GPIOA(0), virg);
    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:

```
配置如下:
dts
soc{
....
Vdevice: vdevice@0 {
    compatible = "allwinner,sun8i-vdevice";
    device_type = "Vdevice";
    interrupt-parent = <&pio>; /*依赖的中断控制器带(interrupt-属性的结点controller)*/
    interrupts = < PD 3 IRQ_TYPE_LEVEL_HIGH>;
```



```
中断触发条件、类型|-----
                                                      -----pin 内偏移bank
                                     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,
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)
                                                                                                                                              All to the state of the state o
       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 \stackrel{\perp}{=} 0) {
 pr_warn("wait for irq timeout!\n");
  free_irq(virq, NULL);
  return -EINVAL;
free irq(virq, NULL);
pr_warn("-----
pr warn("test pin eint success !\n");
-++++ \ln n'n';
return 0;
```

THE TO SEE STATE OF THE SEC STATE OF THE

THE REPORT OF THE PARTY OF THE

Page 26 of 28

6

# 常用 debug 方法说明

待添加

Miller HALVIII Series

ARIV SALSAR

A THE VIEW



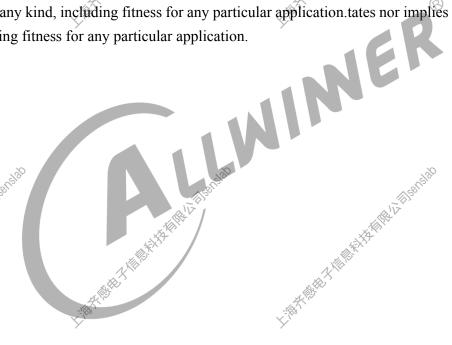


## Declaration

This document is the original work and copyrighted property of Allwinner Technology ("Allwinner"

'). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application tates nor implies warranty of any kind, including fitness for any particular application.



- AARTHAN

14 This sersion

A STATE OF THE PARTY OF THE PAR

A STANTANT OF SELECTION OF SELE