全志科技
Allwinner Technology

H3 项目

pinctrl 接口使用说明书 V1.0

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2015-01-10		正式版本

目　　录

1. 概述

1.1. 编写目的

本文档对H3平台的pin脚管理(pinctrl)接口使用进行详细的阐述，让用户明确掌握pin脚申请、配置等操作的编程方法。

1.2. 适用范围

本文档适用于H3 sdk配套的linux3.4内核。

1.3. 相关人员

本文档适用于所有需要在 H3 平台上开发设备驱动的人员。

2. 模块介绍

Pinctrl 框架是 linux 系统为统一各 SOC 厂商的 pin 脚管理，避免各 SOC 厂商各自实现相同的 pin 脚管理子系统而提出的。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 控制器的寄存器，我们可以配置一个或一组引脚的功能和特性。在软件上，Linux 内核的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。

Sunxi 平台实现的 pinctrl 驱动，是在 linux pinctrl 的通用框架上进行实现与扩张的，平台实现的 pinctrl 驱动除了拥有以上功能之外，还具有如下功能：

- 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。

pin 控制器(pin controller)：Pin 控制器是一种硬件模块，通常为一组可以对单个 pin 脚或者一组 pin 脚进行控制的寄存器。控制的方式有复用、偏置、pin 脚驱动力设置等。

管脚(Pin)：根据芯片不同的封装方式，可以表现为球形、针型等。管脚常用一组无符号的整数[0-maxpin]来表示。当实例化一个 pin 控制器，将为该 pin 控制器注册一个描述符（descriptor）。这个描述符还包含了一组由该 pin 控制器控制的 pin 的描述符。

管脚组(Pin groups)：外设通常都需要接到 soc 的多个 pin 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。

管脚属性(Pin configure)：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，可以使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）。你可以通过设置一个特定寄存器值将一个输入管脚与 VDD 或 GND 相连(上拉/下拉)，以便在没有信号驱动管脚或是未连接时管脚上可以有个确定的值。

管脚多路复用(Pin mux)：pin 的复用功能，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。

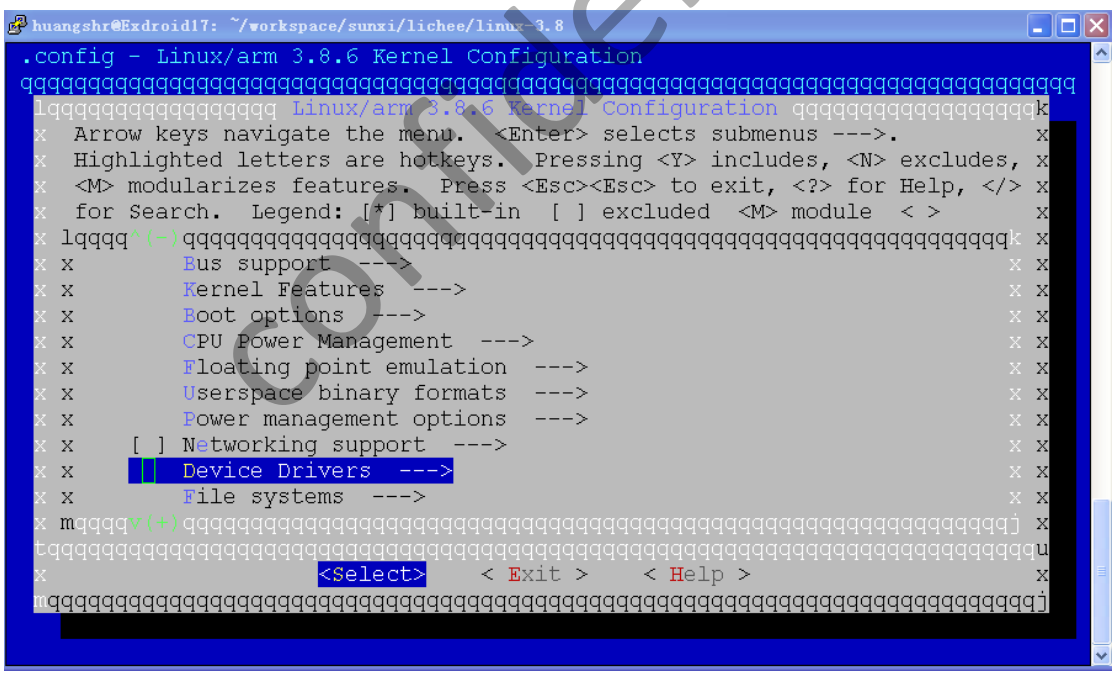
Device Tree：是一种数据结构。犹如它的名字，device tree 是一颗树，包含有许多结点，结点中含有属性，属性中含有名字（name）和值（value）。树的信息主要包括：cpu 的数量和类别，内存基地址，总线与桥，外设连接，中断控制器和中断使用情况，GPIO 以及 CLOCK 器等等，Device Tree 的源代码格式是.dts（device tree source)。在启动过程中，内核会展开 Device Tree（此处是源代码编译后的.dtb 文件)，并创建和注册相关设备，驱动因此也以新方式和.dts 中定义的设备结点进行匹配，然后工作。Pinctrl 驱动支持从 device tree 中定义的设备节点获取 pin 的配置信息，在 sunxi 平台实现的 pinctrl 中，并没有采用 device tree，而是采用了 sys_config 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。

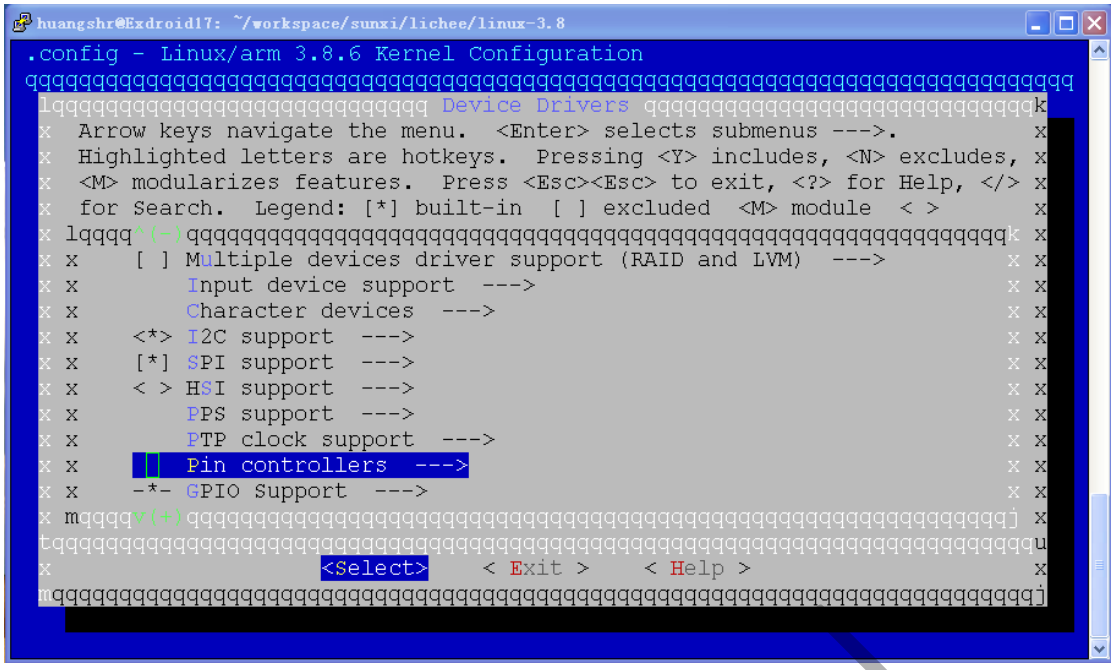
2.3. 模块配置介绍

在命令行中进入内核根目录，执行 make ARCH=arm menuconfig 进入配置主界面，并按以下步骤操作：

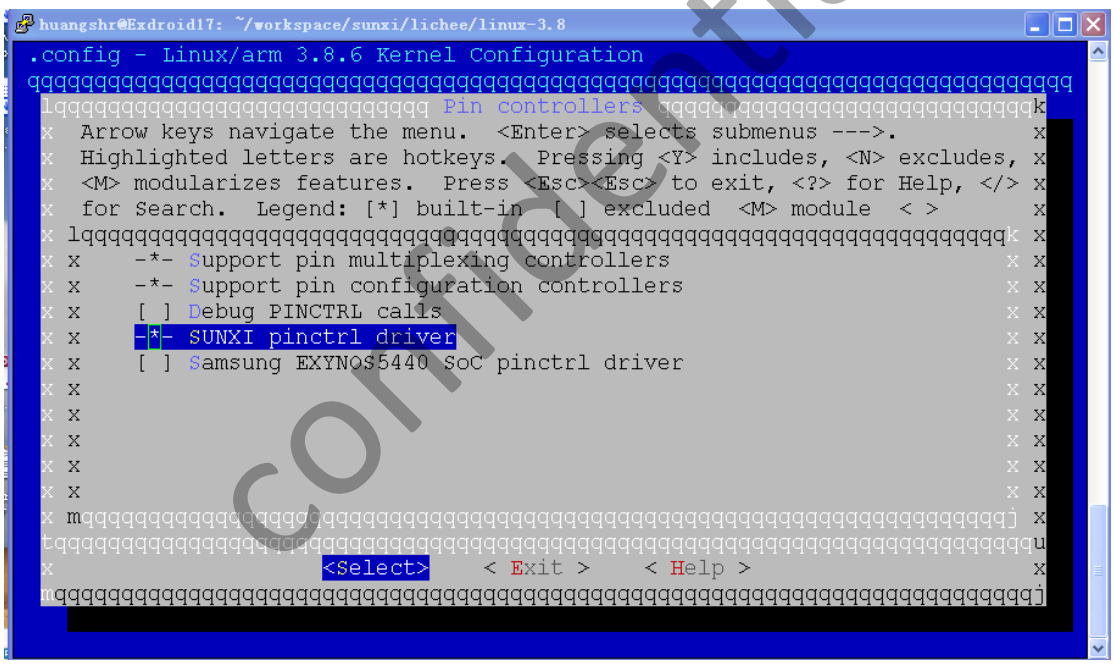
首先，选择 Device Drivers 选项进入下一级配置，如下图所示：



选择 Pin controllers,进入下级配置，如图所示：



Sunxi pinctrl driver 默认编译进内核，如图所示：



2.4. 源码结构介绍

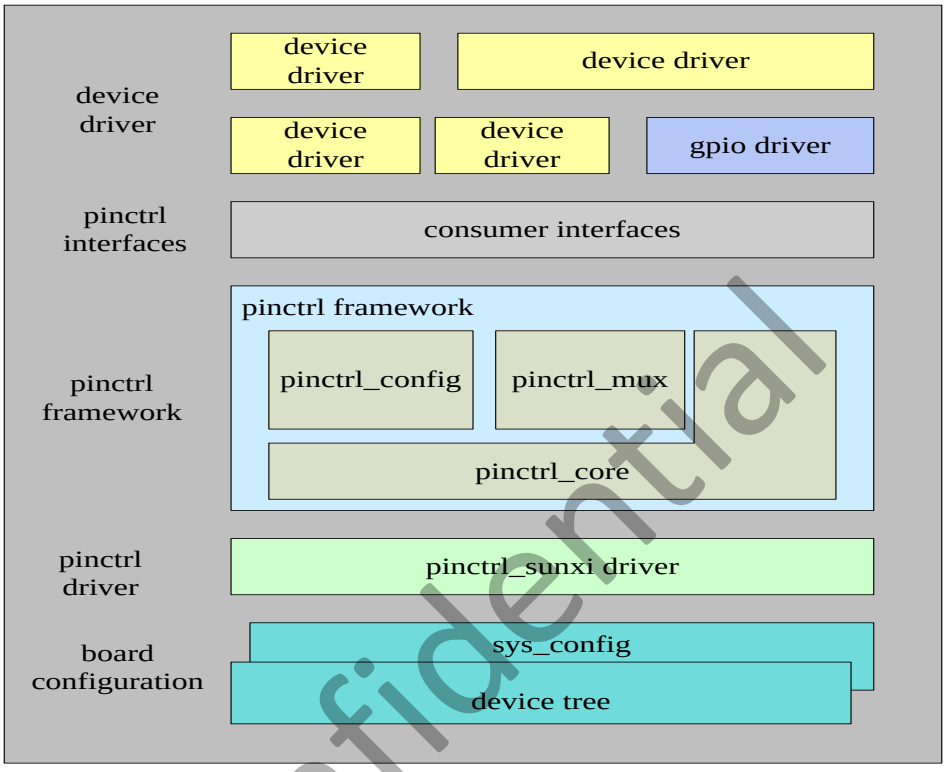
```
linux3.4

|-- arch
|   `-- arm
|       |-- Kconfig
|       `-- mach-sunxi
|           `-- Kconfig
|
|-- drivers
|    `-- pinctrl
|         |-- Kconfig
|         |-- Makefile
|         |-- core.c
|         |-- core.h
|         |-- devicetree.c
|         |-- devicetree.h
|         |-- pinconf.c
|         |-- pinconf.h
|         |-- pinctrl-sunxi-test.c
|         |-- pinctrl-sun8iw7.c
|         |-- pinctrl-sunxi.c
|         |-- pinctrl-sunxi.h
|         |-- pinmux.c
|         `-- pinmux.h
`-- include
    `-- linux
        `-- pinctrl
            |-- consumer.h
            |-- devinfo.h
            |-- machine.h
            |-- pinconf-generic.h
            |-- pinconf.h
            |-- pinconf-sunxi.h
            |-- pinctrl-state.h
            |-- pinctrl.h
            `-- pinmux.h
```

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。



Pinctrl api：

pinctrl 提供给上层用户调用的接口。通过调用接口，可以获得设备的 pin 操作句柄，配置 pin 信息等功能。

Pinctrl common frame：

Linux 提供的 pinctrl 驱动框架。

Pinctrl-sunxi driver：

SUNXI 平台需要实现的驱动，需要填充 pinctrl common frame 规定好的 callback 函数。

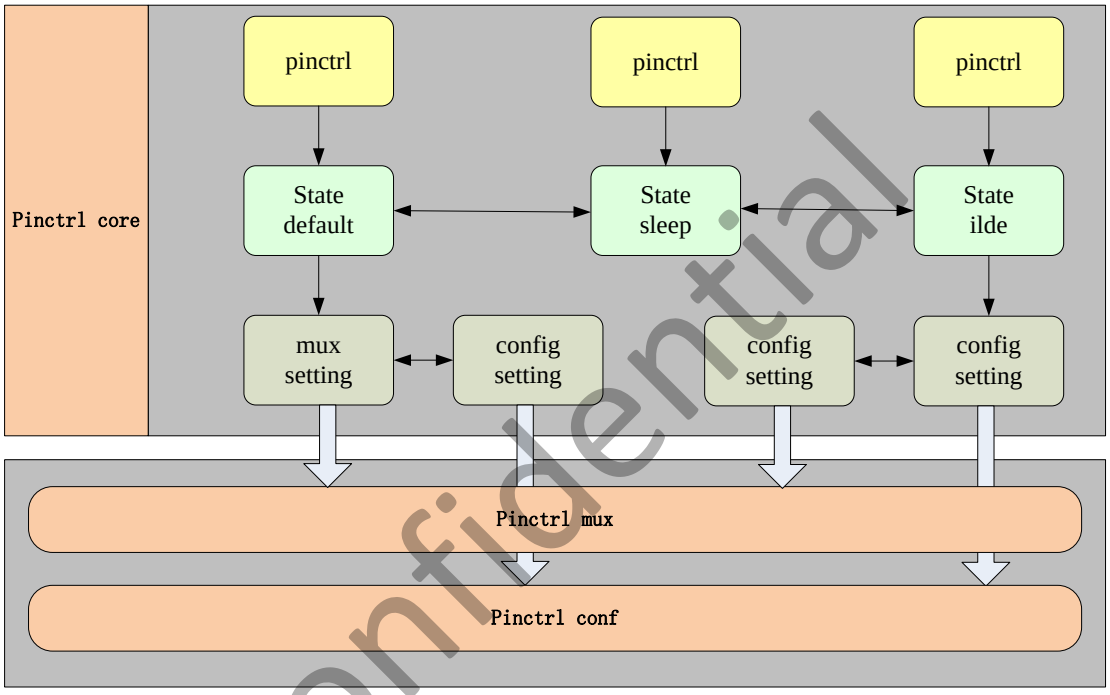
Board configuration：

pinctrl 可以透过 device tree 获取平台配置信息。SUNXI 平台中，我们通过 sys_config 获取平台配置信息。

3.2. Pinctrl/state/pin mux/pin configure 映射关系

Pinctrl common frame 主要处理 pinctrl 的状态管理与 pin 资源分配，各个功能的实现分布在 pinctrl_core.c、pinctrl_conf.c、pinctrl_mux.c 文件中。Pinctrl 的状态（state）、pin mux 与 pin configure 映射关系如上图所示。注册了 pinctrl 设备，需要填充 pinctrl_dev,同时还需要申请结构体 pinctrl 以获取 pinctrl 设备在不同状态时的句柄。

系统运行在不同的状态，pin 的配置有可能不一样，比如在默认状态，pin 的驱动力与上下拉有可能跟系统 处于 idle 状态下不一样。Pinctrl 设备在不同状态下管理 pin，需要能够正确 pin 在不同状态下的资源分配、属性配置。因此，对应于 pinctrl 的每个 state，都需要映射 pin 的 mux、config 关系。



pintctrl 模块初始化时会自动获取 sys_config 中关于 pin 脚配置的信息，并转换为 map 表信息注册给 pinctrl 系统。这些操作对于设备驱动使用都是透明的，设备驱动开发人员只需要将 sys_config 信息配置正确即可。

4. Sunxi pinctrl 接口说明

4.1.pinctrl 接口

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，所有的pin操作必须基于此pinctrl句柄;

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

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，所有的pin操作必须基于此pinctrl句柄； 与
pinctrl_get接口功能完全一样，只是devm_pinctrl_get会将申请的 pinctrl句柄做记账，
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，pinctrl子系统会通过设备名与pin配置信息匹配，
获取pin配置信息。

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pin句柄对应的pinctrl设置为指定状态；

输入： 使用pin的设备，pinctrl子系统会通过设备名与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，pinctrl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。

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 接口

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

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.

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.

5. Sunxi pinctrl Demo

5.1. 设备驱动如何申请 pin

5.1.1. 通过 sys_config 配置方式申请

设备需要的pin资源可以在sys_config文件中配置，在sunxi-pinctrl初始化时会解析sys_config中的设备pin配置信息，并将设备的pin配置信息添加到sunxi-pinctrl模块中，设备驱动申请设备pin资源时只需要告诉pinctrl设备名称即可，pinctrl会自动将设备的pin资源信息绑定到该设备的pinctrl句柄中。

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 pinctrl 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;
}
```

5.1.2. 设备驱动直接申请 pin 方式

sunxi-pinctrl 可通过 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。目前 sunxi-pinctrl 平台支持配置 pin pull/driver-strength/output data/mux 四种属性。可参考以下方式使用：

```
static int sunxi_pin_resource_req(struct platform_device *pdev)
{
    struct pinctrl *pinctrl;
    script_item_u *pin_list;
    int pin_count;
    int pin_index;
    pr_warn("device [%s] pin resource request enter\n", dev_name(&pdev->dev));
    /* get pin sys_config info */
```

```
    pin_count = script_get_pio_list("lcd0", &pin_list);
    if (pin_count == 0) {
        /* "lcd0" have no pin configuration */
        return -EINVAL;
    }
    /* request pin individually */
    for (pin_index = 0; pin_index < pin_count; pin_index++) {
        struct gpio_config *pin_cfg = &(pin_list[pin_index].gpio);
        char *pin_name;
        unsigned long config;
        if (IS_SUNXI_PIN(pin_cfg->gpio)) {
            /* valid pin of sunxi-pinctrl, config pin attributes individually.*/
            pin_name = sunxi_gpio_to_name(pin_cfg->gpio);
            config=                    SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_FUNC,
pin_cfg->mul_sel);
            pin_config_set(SUNXI_PINCTRL, pin_name, config);
            if (pin_cfg->pull != GPIO_PULL_DEFAULT) {
            config = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_PUD, pin_cfg->pull);
            pin_config_set(SUNXI_PINCTRL, pin_name, config);
            }
            if (pin_cfg->drv_level != GPIO_DRVLVL_DEFAULT) {

configs=SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DRV,pin_cfg->drv_level);
            pin_config_set(SUNXI_PINCTRL, pin_name, config);
            }
            if (pin_cfg->data != GPIO_DATA_DEFAULT) {
            configs = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DAT, pin_cfg->data);
            pin_config_set(SUNXI_PINCTRL, pin_name, config);
            }
        } else if (IS_AXP_PIN(pin_cfg->gpio)) {
            /* valid pin of axp-pinctrl, config pin attributes individually. */
            pin_name = axp_gpio_to_name(pin_cfg->gpio);
            config=                    SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_FUNC,

pin_cfg->mul_sel);
            pin_config_set(AXP_PINCTRL, pin_name, config);
            if (pin_cfg->data != GPIO_DATA_DEFAULT) {
            configs = SUNXI_PINCFG_PACK(SUNXI_PINCFG_TYPE_DAT, pin_cfg->data);
            pin_config_set(AXP_PINCTRL, pin_name, config);
            }
        } else {
            pr_warn("invalid pin [%d] from sys-config\n", pin_cfg->gpio);
        }
    }
    pr_debug("device [%s] pin resource request ok\n", dev_name(&pdev->dev));
```

```
    return 0;
}
```

5.2. 设备驱动如何使用 pin 中断

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;
}
```

Declaration