Rockchip
瑞芯微电子

Rockchip
Pin-Ctrl 开发指南

发布版本:1.0

日期:2016.07

前言

概述

产品版本

芯片名称	内核版本
RK3399	Linux4.4

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
2016-07-25	V1.0	WDC	初始版本

目录

1 Pin-Ctrl 配置

pinctrl 部分主要包括 mux，驱动强度，上下拉配置等。

1.1 驱动文件与 DTS 配置

驱动文件所在位置：

drivers/pinctrl/pinctrl-rockchip.c

驱动 DTS 节点配置 pinctrl，通过驱动 Probe 的时候，会将“default”对应的这组 Pinctrl 配置到寄存器里面，而其他组的配置需要在代码里面解析出来，再选择切换使用。例如 HDMI pinctrl 配置与代码中使用：

```
hdmi_rk_fb: hdmi-rk-fb@ff940000 {
        status = "disabled";
        compatible = "rockchip,rk3399-hdmi";
        reg = <0x0 0xff940000 0x0 0x20000>;
        interrupts = <GIC_SPI 23 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 24 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&cru PCLK_HDMI_CTRL>,
                 <&cru HCLK_HDCP>,
                 <&cru SCLK_HDMI_CEC>,
                 <&cru PLL_VPLL>,
                 <&cru SCLK_HDMI_SFR>;
        clock-names = "pclk_hdmi",
                      "hdcp_clk_hdmi",
                      "cec_clk_hdmi",
                      "dclk_hdmi_phy",
                      "sclk_hdmi_sfr";
        resets = <&cru SRST_HDMI_CTRL>;
        reset-names = "hdmi";
        pinctrl-names = "default", "gpio";
        pinctrl-0 = <&hdmi_i2c_xfer &hdmi_cec>;
        pinctrl-1 = <&i2c3_gpio>;
        rockchip,grf = <&grf>;
        power-domains = <&power RK3399_PD_HDCP>;
};
hdmi {
        hdmi_i2c_xfer: hdmi-i2c-xfer {
                rockchip,pins =
                        <4 17 RK_FUNC_3 &pcfg_pull_none>,
                        <4 16 RK_FUNC_3 &pcfg_pull_none>;
```

```
                };

                hdmi_cec: hdmi-cec {
                        rockchip,pins =
                                <4 23 RK_FUNC_1 &pcfg_pull_none>;
                };
        };
        i2c3_gpio: i2c3_gpio {
                 rockchip,pins =
                        <4 17 RK_FUNC_GPIO &pcfg_pull_none>,
                        <4 16 RK_FUNC_GPIO &pcfg_pull_none>;
        };
```

代码中使用：

驱动解析得到 GPIO 状态，并切换成 GPIO 模式：

```
    gpio_state = pinctrl_lookup_state(hdmi_dev->dev->pins->p, "gpio");
    pinctrl_select_state(hdmi_dev->dev->pins->p, gpio_state);
```

default 状态不需要驱动解析，直接切换成 default 模式：

```
    pinctrl_select_state(hdmi_dev->dev->pins->p,
                         hdmi_dev->dev->pins->default_state);
```

1.2 Iomux 配置

Iomux 配置即切换 pin 所对应的 mux 值，其中有如下的定义，分别对应相应的寄存器值：

```
 #define RK_FUNC_GPIO    0
#define RK_FUNC_1       1
#define RK_FUNC_2       2
#define RK_FUNC_3       3
#define RK_FUNC_4       4
```

下面仍然举例 hdmii2c_xfer；首先，我们在 3399 的 trm 的 GRF 章节里面找到了 i2c3hdmi_scl 和 i2c3hdmi_sda 两个 pin 脚对应的是 gpio4c1 和 GPIO4c0，两个功能脚都是二进制‘11’作为功能值， 即使用 RK_FUNC_3；因为 GPIOA 有 8 个 pin，GPIOB 也有 8 个 pin，以此计算可得 GPIO4c1 和 GPIO4c0 为 “4 17”和 “4 16”；

3:2	RW	0x0	gpio4c1_sel GPIO4C[1] iomux select 2'b00: gpio 2'b01: i2c3hdmi_scl 2'b10: uart2dbgb_sout 2'b11: hdmii2c_scl
1:0	RW	0x0	gpio4c0_sel GPIO4C[0] iomux select 2'b00: gpio 2'b01: i2c3hdmi_sda 2'b10: uart2dbgb_sin 2'b11: hdmii2c_sda

```
        hdmi_i2c_xfer: hdmi-i2c-xfer {
                rockchip,pins =
                        <4 17 RK_FUNC_3 &pcfg_pull_none>,
                        <4 16 RK_FUNC_3 &pcfg_pull_none>;
        };
```

如果硬件原理图上与所给参考代码定义 pin 引脚不同或者 mux 值不同时候，如何修改。

假设现在有某个具体的产品是使用 i2c2 来连接 HDMI 作通讯功能，通过在该产品的 rk3399-xxx.dts 引用覆盖来实现，下面为示例：

```
&hdmi_rk_fb {
        status = "okay";
        pinctrl-names = "default", "gpio";
        pinctrl-0 = <&i2c2_xfer &hdmi_cec>;
        pinctrl-1 = <&i2c2_gpio>;
};
```

1.3 驱动强度配置

驱动强度配置，即配置所对应的驱动强度电流值，分别对应相应的寄存器值，与 mux 用法类似，以下为示例：

```
                pcfg_pull_up_2ma: pcfg-pull-up-2ma {
                        bias-pull-up;
                        drive-strength = <2>;
                };

                pcfg_pull_down_12ma: pcfg-pull-down-12ma {
                        bias-pull-down;
                        drive-strength = <12>;
                };

                pcfg_pull_none_13ma: pcfg-pull-none-13ma {
```

```
                        bias-disable;
                        drive-strength = <13>;
                };

                gmac {
                        rgmii_pins: rgmii-pins {
                                rockchip,pins =
                                        /* mac_txd1 */
                                        <3 5 RK_FUNC_1
&pcfg_pull_none_13ma>,
                                        /* mac_txd0 */
                                        <3 4 RK_FUNC_1
&pcfg_pull_none_13ma>,
                                        /* mac_rxd3 */
                                        <3 3 RK_FUNC_1
&pcfg_pull_none>,
                                        /* mac_rxd2 */
                                        <3 2 RK_FUNC_1
&pcfg_pull_none>,
                                        /* mac_txd3 */
                                        <3 1 RK_FUNC_1
&pcfg_pull_none_13ma>,
                                        /* mac_txd2 */
                                        <3 0 RK_FUNC_1
&pcfg_pull_none_13ma>;
                        };
        };
```

如果想增加或减少驱动强度，但是与所给参考代码定义的驱动强度不同时候，如何修改。同样类似 mux 的修改，在产品的 dts 文件里面引用之后，修改覆盖。

每一个 pin 具有自己所在对应的驱动电流强度范围，所以配置的时候要选择其有效可配电流值，如果不是该 pin 对应的有效电流值，配置将会出错，无法生效。

1.4 上下拉配置

上下拉配置，即配置芯片 pad 内部上拉，下拉或者都不配置，分别对应相应的寄存器值，与 mux 用法类似，以下为示例：

```
                pcfg_pull_up: pcfg-pull-up {
                        bias-pull-up;
                };

                pcfg_pull_down: pcfg-pull-down {
                        bias-pull-down;
```

```
                };

                pcfg_pull_none: pcfg-pull-none {
                        bias-disable;
                };

                pcfg_pull_up_8ma: pcfg-pull-up-8ma {
                        bias-pull-up;
                        drive-strength = <8>;
                };

                uart1 {
                        uart1_xfer: uart1-xfer {
                                rockchip,pins =
                                        <3 12 RK_FUNC_2 &pcfg_pull_up>,
                                        <3 13 RK_FUNC_2
&pcfg_pull_none>;
                        };
                };
```

如果想配置成上拉，下拉或者都不配置，但是与所给参考代码定义的配置不同时候，如何修改。同样类似 MUX 的修改，在产品的 DTS 文件里面引用之后，修改覆盖。

1.5 常见问题

a) 如果 dts 配置正确了，但是读寄存器配置不对，请确认该驱动是否有调用到 probe 函数；

b) 如果 dts 配置正确了，但是读寄存器配置不对，且 a 已正确，请确认是否有 pinctrl 的错误 log 出现，例如：

c) 如果 dts 配置正确了，但是读寄存器配置不对，且 a，b 都已正确，有可能被其他模块使用，一般都是切成 GPIO，请搜索 dts 文件，确认是否有其他模块使用该 pin 脚；如果 dts 没有检索出来，可以在 drivers/pinctrl/pinctrl-rockchip.c 的 rockchip_set_mux()函数中加入打印，例如现在要找 GPIO1_B7 被哪个驱动所用，可以加入下面补丁，通过 dump_stack()看是否为正确调用：

```
    diff --git a/drivers/pinctrl/pinctrl-rockchip.c
b/drivers/pinctrl/pinctrl-rockchip.c
    index c6c04ac..c1dd0bd 100644
    --- a/drivers/pinctrl/pinctrl-rockchip.c
    +++ b/drivers/pinctrl/pinctrl-rockchip.c
    @@ -661,6 +661,11 @@ static int rockchip_set_mux(struct
rockchip_pin_bank *bank, int pin, int mux)
            dev_dbg(info->dev, "setting mux of GPIO%d-%d to %d\n",
                                                bank->bank_num, pin,
```

```
mux);

    +       if ((bank->bank_num == 1) && (pin == 15)) {
    +               printk("setting mux of GPIO%d-%d to %d\n",
bank->bank_num, pin, mux);
    +               dump_stack();
    +       }
    +
          regmap = (bank->iomux[iomux_num].type & IOMUX_SOURCE_PMU)
                                        ? info->regmap_pmu : info->regmap_base
```

d) RK3399 中的 uart_dbg 和 pwm 有定义多组 pin 脚，除了正常的 iomux 设置外，其中 uart_dbg 还需要通过 GRF_SOC_CON7 寄存器的 bit10~bit11 选择; pwm3 需要通过 PMUGRF_SOC_CON0 寄存器的 bit5 来选择，目前这些配置都在 uboot fireware 里面做。

GRF_SOC_CON7
Address: Operational Base + offset (0x0e21c)
SoC control register 7

13	RO	0x0	reserved
12	RW	0x1	grf_con_force_jtag
11:10	RW	0x0	grf_uart_dbg_sel

PMUGRF_SOC_CON0
Address: Operational Base + offset (0x00180)
SoC control register 0

5	RW	0x1	pwm3_sel Use 2 optional IOs for pwm3. 0: pwm3a 1: pwm3b

2 GPIO 使用

PINCTRL 除了 MUX 值配置以外，最常使用的就是 GPIO，下面介绍 GPIO 使用方式；

2.1 DTS 配置与代码使用

以下面的声卡 ES8316 为例，在 DTS 定义了两个 GPIO，分别是 GPIO0_B3 和 GPIO4_D4:

```
        es8316: es8316@10 {
                #sound-dai-cells = <0>;
                compatible = "everest,es8316";
                reg = <0x10>;
                clocks = <&cru SCLK_I2S_8CH_OUT>;
                clock-names = "mclk";
                spk-con-gpio = <&gpio0 11 GPIO_ACTIVE_HIGH>;
                hp-det-gpio = <&gpio4 28 GPIO_ACTIVE_LOW>;
        };
```

代码中使用，spk-con-gpio 申请为 gpio 输出，另外一个 hp-det-gpio 申请为输入：

```
    es8316->spk_ctl_gpio = of_get_named_gpio_flags(np, "spk-con-gpio", 0,
&flags);
    ret = devm_gpio_request_one(&i2c->dev, es8316->spk_ctl_gpio,
GPIOF_DIR_OUT, NULL);

    es8316->hp_det_gpio = of_get_named_gpio_flags(np, "hp-det-gpio", 0,
&flags);
    ret = devm_gpio_request_one(&i2c->dev, es8316->hp_det_gpio, GPIOF_IN,
"hp det");
```

2.2 GPIO 中断

参考下面例子，用法简单了，网上例子也很多：

```
    mpu6500@68 {
                status = "disabled";
                compatible = "invensense,mpu6500";
                pinctrl-names = "default";
                pinctrl-0 = <&mpu6500_irq_gpio>;
                reg = <0x68>;
                irq-gpio = <&gpio1 4 IRQ_TYPE_EDGE_RISING>;
                mpu-int_config = <0x10>;
                mpu-level_shifter = <0>;
                mpu-orientation = <0 1 0 1 0 0 0 0 1>;
```

```
                    orientation-x= <1>;
                    orientation-y= <1>;
                    orientation-z= <1>;
                    mpu-debug = <1>;
            };
    gpio_pin = of_get_named_gpio_flags(np, "irq-gpio", 0, (enum of_gpio_flags
*)&irq_flags);
    gpio_request(gpio_pin, "mpu6500");
    client->irq = gpio_to_irq(gpio_pin);
    ret = request _irq(client->irq, inv_irq_handler, irq_flags | IRQF_SHARED,
"inv_irq", st);
```

2.3 GPIO 常见问题

1. 当使用 GPIO request 时候，会将该 PIN 的 MUX 值强制切换为 GPIO，所以使用该 pin 脚为 GPIO 功能的时候确保该 pin 脚没有被其他模块所使用；
2. 如果用 IO 命令读某个 GPIO 的寄存器，读出来的值是异常, 0x00000000 或 0xffffffff 等，请确认该 GPIO 的 CLK 是不是被关了，打开 clk，应该可以读到正确的寄存器；
3. 测量该 PIN 脚的电压不对时，如果排除了外部因素，可以确认下该 pin 所在的 io 电压源是否正确，以及 IO-Domain 配置是否正确。
4. 如果使用该 GPIO 时，不会动态的切换输入输出，建议在开始时就设置好 GPIO 输出方向，后面拉高拉低时使用 gpio_set_value()接口，而不建议使用 gpio_direction_output(), 因为 gpio_direction_output 接口里面有 mutex 锁，对中断上下文调用会有错误异常，且相比 gpio_set_value，gpio_direction_output 所做事情更多，浪费。