RK805 开发指南

发布版本：1.0

作者邮箱：chenjh@rock-chips.com

日期：2018.05

文档密级：公开资料

前言

概述

```
1 本文档主要介绍RK805的各个子模块，介绍相关概念、功能、dts配置和一些常见问题的分析定位。
```

产品版本

芯片名称	内核版本
RK805	3.10、 4.4

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
2017.05.28	V1.0	陈健洪	初稿

---

1 基础

1.1 概述

RK805 是一款高性能 PMIC，RK805集成4个大电流DCDC、3个LDO、1个RTC、可调上电时序等功能。

系统中各路电源总体分为两种：DCDC和LDO。两种电源的总体特性如下（详细资料请自行搜索）：

1. DCDC：输入输出压差大时，效率高，但是存在纹波比较大的问题，成本高，所以大压差，大电流负载时使用。一般有两种工作模式。PWM模式：纹波瞬态响应好，效率低；PFM模式：效率高，但是负载能力差。
2. LDO：输入输出压差大时，效率低，成本低，为了提高LDO的转换效率，系统上会进行相关优化如：LDO输出电压为1.1V，为了提高效率，其输入电压可以从VCCIO_3.3V的DCDC给出。所以电路上如果允许尽量将LDO接到DCDC输出回路，但是要注意上电时序。

1.2 功能

从使用者的角度看，RK805的功能概况起来可以分为4个部分：

1. regulator功能：控制各路DCDC、LDO电源状态；
2. rtc功能：提供时钟计时、定时等功能；
3. gpio功能：out1和out2两个推挽输出引脚（只能output），可当普通gpio使用；
4. pwrkey功能：检测power按键的按下/释放，可以为AP节省一个gpio。

1.3 芯片引脚功能

下面描述中，SLEEP和INT引脚需要重点关注：

1.4 重要概念

- I2C地址

  7位从机地址：0x18

- PMIC有3种工作模式

  1. PMIC normal模式

系统正常运行时PMIC处于normal模式，此时pmic_sleep为低电平。

2. PMIC sleep模式

系统休眠时需要待机功耗尽量低，PMIC会切到sleep模式减低自身功耗，这时候一般会降低某些路的输出电压，或者直接关闭输出，这可以根据实际产品需求进行配置。系统待机时AP通过I2C指令把pmic_sleep配置成sleep模式，然后拉高pmic_sleep即可让PMIC进入sleep 状态；当SoC唤醒时pmic_sleep恢复为低电平，PMIC退出休眠模式。

3. PMIC shutdown模式

当系统进入关机流程的时候，PMIC需要完成整个系统的电源下电操作。AP通过I2C指令把pmic_sleep配置成shutdown模式，然后拉高pmic_sleep即可让PMIC进入shutdown状态。

- pmic_sleep引脚

  常态为低电平，PMIC处于normal模式。当引脚拉高的时候会切换到sleep或者shutdown的模式。

- pmic_int 引脚

  常态为高电平，当有中断产生的时候变为低电平。如果中断没有被处理，则会一直维持低电平。

- out1/out2引脚

  这两个引脚可以当普通的gpio使用（推挽输出），但是只有gpio输出模式。

- pmic_pwron引脚

  pwrkey的功能需要硬件上将power按键接到这个引脚，驱动通过这个引脚来判断按下/释放。

- 各路DCDC的工作模式

  DCDC有PWM（也叫 force PWM）、PFM模式，但是PMIC有一种模式会动态切换PWM、PFM，这就是我们通常所说的AUTO模式。PMIC支持 PWM、AUTO PWM/PFM两种模式，AUTO模式效率高但是纹波瞬态响应会差。出于系统稳定性考虑，运行时都是设置为 PWM模式，系统进入休眠时会选择切换到AUTO PWM/PFM。

- DCDC3电压调节

  DCDC3这路电源比较特殊，不能通过寄存器修改电压，只能通过外部电路的分压电阻进行调节，所以如果需要修改电压请修改外围硬件，在Rockchip的方案上一般作为VCC_DDR使用。

- DCDC和LDO的运行时电压调节范围

  1. DCDC电压范围不连续：

电压范围(V)	**步进值(mV)**	**具体档位值(V)**
0.7125 ~ 1.45	12.5	0.7125、0.725、0.737.5、 .....、1.45
1.8 ~ 2.2	200	1.8、 2.0、2.2
2.3	无	2.3

  2. LDO电压连续：

电压范围(V)	**步进值(mV)**	**具体档位值(V)**
0.8 ~ 3.4	100	0.8、 0.9、1.0、1.1、 1.2、 3.4

1.5 上电条件和时序

1. 上电条件

   只要满足下面任意一个条件即可以实现PMIC上电：

- EN信号从低电平变高电平触发
- EN信号保持高电平，且RTC闹钟中断触发
- EN信号保持高电平，按PWRON键触发

2. 上电时序

   每款SOC平台对各路电源上电时序要求可能不一样，目前上电时序有如下情况，具体请参考最新的datasheet：

2 配置

2.1 驱动和menuconfig

3.10内核配置

RK805驱动文件（复用RK816驱动）：

```
drivers/mfd/rk816.c
drivers/input/misc/rk816-pwrkey.c
drivers/rtc/rtc-rk816.c
drivers/gpio/gpio-rk816.c
drivers/regulator/rk816-regulator.c
```

menuconfig里对应的宏配置：

```
CONFIG_MFD_RK816
CONFIG_GPIO_RK816
CONFIG_RTC_RK816
CONFIG_REGULATOR_RK816
CONFIG_INPUT_RK816_PWRKEY
```

4.4内核配置

RK805驱动文件：

```
drivers/mfd/rk808.c
drivers/input/misc/rk8xx-pwrkey.c
drivers/rtc/rtc-rk808.c
drivers/gpio/gpio-rk8xx.c
drivers/regulator/rk818-regulator.c
drivers/clk/clk-rk808.c
```

menuconfig里对应的宏配置：

```
CONFIG_MFD_RK808
CONFIG_RTC_RK808
CONFIG_GPIO_RK8XX
CONFIG_REGULATOR_RK818
CONFIG_INPUT_RK8XX_PWRKEY
CONFIG_COMMON_CLK_RK808
```

2.2 DTS配置

3.10内核配置

DTS的配置包括：I2C挂载、主体、regulator、rtc、poweroff等部分。

```
&i2c1 {
    rk805: rk805@18 {
        reg = <0x18>;
        status = "okay";
    };
};

#include "../../../arm/boot/dts/rk805.dtsi"
&rk805 {
    gpios = <&gpio2 GPIO_A6 GPIO_ACTIVE_HIGH>, <&gpio2 GPIO_D2 GPIO_ACTIVE_LOW>;
    rk805,system-power-controller;
    gpio-controller;
    #gpio-cells = <2>;

    rtc {
        status = "disabled";
    };

    regulators {
        rk805_dcdc1_reg: regulator@0 {
            regulator-name = "vdd_logic";
            regulator-min-microvolt = <700000>;
            regulator-max-microvolt = <1500000>;
            regulator-initial-mode = <0x1>;
            regulator-initial-state = <3>;
            regulator-boot-on;
            regulator-always-on;
            regulator-state-mem {
                regulator-state-mode = <0x2>;
                regulator-state-enabled;
                regulator-state-uv = <1000000>;
            };
        };
        rk805_dcdc2_reg: regulator@1 {
                    .............
        };
        rk805_dcdc3_reg: regulator@2 {
                    .............

        };
```

```
        ..............................
    };
};
```

1. I2C挂载

   整个完整的rk805节点挂在对应的i2c节点下面，并且配置status = "okay";

2. 主体部分

- 不可修改部分

```
rk805,system-power-controller：声明RK805具备管理系统下电的功能；
gpio-controller：声明RK805具有GPIO的功能；
#gpio-cells：使用者引用RK805的GPIO时需要指定的参数个数；
```

  说明：如果某个节个需要引用RK805的GPIO进行使用，引用格式如下：

  gpios = <&rk805 0 GPIO_ACTIVE_LOW>; 第一个参数： &rk805固定，不可改动； 第二个参数： 引用rk805的哪个gpio，只能是0或者1，其中0：out1， 1：out2； 第三个参数： gpio的极性。

- 可修改部分

```
gpios：指定pmic_int（第一个）和pmic_sleep（第二个）引脚；
```

3. regulator部分

- `regulator-name`: 电源名字，建议和硬件图上保持一致，使用regulator_get接口时需要匹配这个名字；
- `regulator-min-microvolt`：运行时可调节的最小电压；
- `regulator-max-microvolt`：运行时可调节的最大电压；
- `regulator-initial-mode`：运行时DCDC工作模式，一般配置为1。 1：force pwm，2：auto pwm/pfm；
- `regulator-state-mode`： 休眠时DCDC工作模式，一般配置为2。1：force pwm， 2：auto pwm/pfm；
- `regulator-initial-state`：suspend时的模式，必须配置成3；
- `regulator-boot-on`：存在这个属性时，在注册regulator的时候就会使能这路电源；
- `regulator-always-on`：存在这个属性时，运行时不允许关闭这路电源且会在注册的时候使能这路电源；
- `regulator-state-enabled`：休眠时保持上电状态，想要关闭该路电源，则改成”regulator-state-disabled”；
- `regulator-state-uv`：休眠不断电情况下的待机电压。

说明：

如果regulator-min-microvolt和regulator-max-microvolt的电压相等，则在注册这个regulator的时候系统框架默认会把这个电压设置下去并使能这路电源，不需要使用者干预。

如果regulator-boot-on或者regulator-always-on存在，则系统框架在注册这路regulator的时候默认会进行enable，此时的这路regulator的电压有2种情况：如果regulator-min-microvolt和regulator-max-microvolt的电压相等，则系统框架会把这路电压设置为当前这个电压值；如果regulator-min-microvolt和regulator-max-microvolt的电压不相等，则此时的电压是PMIC的本身的硬件默认上电电压。

4. rtc部分

如果不想使能RTC的功能（如box产品上），则需要像上面那样增加节点，显式指明为status = "disabled"。如果需要使能的的话则可以把整个RTC节点去掉或者设置状态为status = "okay"即可。

5. poweroff部分

```
gpio_poweroff {
    compatible = "gpio-poweroff";
    gpios = <&gpio2 GPIO_D2 GPIO_ACTIVE_HIGH>;
    status = "okay";
};
```

因为RK805支持拉高pmic_sleep引脚进行整个PMIC的下电，所以需要在根节点下增加这个节点。其中gpios是可改部分，用于指明pmic_sleep引脚。

4.4内核配置

DTS的配置包括：i2c挂载、主体、rtc、pwrkey、gpio、regulator等部分。

```
&pinctrl {
    pmic {
        pmic_int_l: pmic-int-l {
        rockchip,pins =
            <2 6 RK_FUNC_GPIO &pcfg_pull_up>;   /* gpio2_a6 */
        };
    };
};

&i2c1 {
    status = "okay";
    rk805: rk805@18 {
        compatible = "rockchip,rk805";
        status = "okay";
        reg = <0x18>;
        interrupt-parent = <&gpio2>;
        interrupts = <6 IRQ_TYPE_LEVEL_LOW>;
        pinctrl-names = "default";
        pinctrl-0 = <&pmic_int_l>;
        rockchip,system-power-controller;
        wakeup-source;
        gpio-controller;
        #gpio-cells = <2>;
        rtc {
            status = "disabled";
        };
        pwrkey {
            status = "disabled";
        };
        gpio {
            status = "okay";
        };
        regulators {
            compatible = "rk805-regulator";
            status = "okay";
            #address-cells = <1>;
            #size-cells = <0>;
            vdd_logic: RK805_DCDC1@0 {

                regulator-compatible = "RK805_DCDC1";
```

```
                regulator-name = "vdd_logic";
                regulator-min-microvolt = <712500>;
                regulator-max-microvolt = <1450000>;
                regulator-initial-mode = <0x1>;
                regulator-ramp-delay = <12500>;
                regulator-boot-on;
                regulator-always-on;
                regulator-state-mem {
                    regulator-mode = <0x2>;
                    regulator-on-in-suspend;
                    regulator-suspend-microvolt = <1000000>;
                };
            };

            vdd_arm: RK805_DCDC2@1 {
                ................
            };
            vcc_ddr: RK805_DCDC3@2 {
                ................
            };
            ..............................
        };
    };
};
```

1. i2c挂载

整个完整的rk805节点挂在对应的i2c节点下面，并且配置status = "okay";

2. 主体部分

- 不可修改：

```
compatible = "rockchip,rk805";
reg = <0x18>;
rockchip,system-power-controller;
wakeup-source;
gpio-controller;
#gpio-cells = <2>;
```

- 可修改（按照pinctrl规则）

```
interrupt-parent：pmic_int隶属于哪个gpio；
interrupts：pmic_int在interrupt-parent的gpio上的引脚索引编号和极性；
pinctrl-names：不修改，固定为 "default"；
pinctrl-0：引用pinctrl里定义好的pmic_int引脚；
```

3. rtc、pwrkey、gpio

如果menuconfig选中了这几个模块，但是实际又不需要使能这几个驱动，那么可以在dts里增加rtc、pwrkey、gpio节点，并且显式指明状态为status = "disabled"，这样就不会使能驱动，但是开机信息会有错误log报出，可以忽略；如果要使能驱动，则可以去掉相应的节点，或者设置状态为status = "okay"。

4. regulator

- `regulator-compatible`：驱动注册时需要匹配的名字，不能改动，否则会加载失败；
- `regulator-name`：电源的名字，建议和硬件图上保持一致，使用regulator_get接口时需要匹配这个名字；
- `regulator-min-microvolt`：运行时可以调节的最小电压；
- `regulator-max-microvolt`：运行时可以调节的最大电压；
- `regulator-initial-mode`：运行时DCDC的工作模式，一般配置为1。 1：force pwm，2：auto pwm/pfm；
- `regulator-mode`：休眠时DCDC的工作模式，一般配置为2。1：force pwm， 2：auto pwm/pfm；
- `regulator-initial-state`：suspend时的模式，必须配置成3；
- `regulator-boot-on`：存在这个属性时，在注册regulator的时候就会使能这路电源；
- `regulator-always-on`：存在这个属性时，表示运行时不允许关闭这路电源且会在注册的时候使能这路电源；
- `regulator-ramp-delay`：DCDC的电压上升时间，固定配置为12500；
- `regulator-on-in-suspend`：休眠时保持上电状态，想要关闭该路电源，则改成”regulator-off-in-suspend”；
- `regulator-suspend-microvolt`：休眠不断电情况下的待机电压。

2.3 函数接口

如下几个接口基本可以满足日常使用，包括regulator开、关、电压设置、电压获取等：

1. 获取regulator：

```
struct regulator *regulator_get(struct device *dev, const char *id)
```

dev默认填写NULL即可，id对应dts里的regulator-name属性。

2. 释放regulator

```
void regulator_put(struct regulator *regulator)
```

3. 打开regulator

```
int regulator_enable(struct regulator *regulator)
```

4. 关闭regulator

```
int regulator_disable(struct regulator *regulator)
```

5. 获取regulator电压

```
int regulator_get_voltage(struct regulator *regulator)
```

6. 设置regulator电压

```
int regulator_set_voltage(struct regulator *regulator, int min_uV, int max_uV)
```

传入的参数时保证 min_uV = max_uV，由调用者保证。

7. 范例

```
struct regulator *rdev_logic;

rdev_logic = regulator_get(NULL, "vdd_logic");          // 获取vdd_logic
regulator_enable(rdev_logic);                           // 使能vdd_logic
regulator_set_voltage(rdev_logic, 1100000, 1100000);    // 设置电压1.1v
regulator_disable(rdev_logic);                          // 关闭vdd_logic
regulator_put(rdev_logic);                              // 释放vdd_logic
```

3 Debug

3.10内核

因为PMIC涉及的驱动在使用逻辑上都不复杂，重点都体现在最后的寄存器设置上。所以目前常用的debug方式就是直接查看rk805的寄存器，通过如下节点：

```
/sys/rk816/rk816_test
```

读寄存器：

```
echo r [addr]  > /sys/rk816/rk816_test
```

写寄存器：

```
echo w [addr] [value] > /sys/rk816/rk816_test
```

范例：

```
echo r 0x2f > /sys/rk816/rk816_test         // 读取0x2f寄存器的值，为0x9b
```

```
1|shell@rk3228h:/ # echo r 0x2f >  /sys/rk816/rk816_test
[  283.091704]  [2:              sh:  618] ------zhangqing: get cmd = r
[  283.091766]  [2:              sh:  618] CMD : r 2f
[  283.091914]  [2:              sh:  618]
[  283.091914]  [2:              sh:  618] 2f 9b
```

```
echo w 0x2f 0x9c > /sys/rk816/rk816_test    // 设置0x2f寄存器的值为0x9c
```

一般写操作执行完之后最好再读一遍确认是否写成功。

```
1|shell@rk3228h:/ # echo r 0x2f >  /sys/rk816/rk816_test
[  283.091704]  [2:              sh:  618] ------zhangqing: get cmd = r
[  283.091766]  [2:              sh:  618] CMD : r 2f
[  283.091914]  [2:              sh:  618]
[  283.091914]  [2:              sh:  618] 2f 9b
```

4.4内核

命令格式同3.10内核一样，只是节点路径不同，4.4内核上的debug节点路径是：

```
/sys/rk8xx/rk8xx_dbg
```