

密级状态：绝密( ) 秘密( ) 内部( ) 公开( ☒ )

## RockChip\_Uboot 开发文档

(技术部)

文件状态：  [ ] 正在修改  [ <input checked="" type="checkbox"/> ] 正式发布	当前版本：	V1.0
	作 者：	yxj, cwz, cjf, zyw, hl 等
	完成日期：	2014-06-18
	审 核：	cwz
	完成日期：	2014-06-16

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有, 翻版必究)

## 版 本 历 史

版本号	作者	修改日期	修改说明	备注
V1.0		2014-06-18	初始版本	

## 目录

1. UBOOT 简介 .....	5
2. ROCKCHIP 平台相关代码 .....	6
3. 编译.....	7
4. CACHE 机制 .....	8
5. 中断机制.....	9
6. CLOCK 驱动 .....	10
7. I2C 驱动 .....	15
8. SPI 驱动.....	16
9. LCD 驱动 .....	17
10. 电源相关的控制.....	18
11. 电量计驱动.....	19
12. EMMC 及 NAND 的驱动 .....	21
13. FASTBOOT .....	21
13.1 进入 FASTBOOT 状态的方式.....	21
13.2 FASTBOOT 主要支持命令 .....	21
13.2.1 获取信息: .....	21
13.2.2 烧写: .....	22
13.2.3 重启: .....	22
13.2.4 解锁和锁住设备: .....	22
13.2.5 特殊命令: .....	23
13.3 FASTBOOT 解锁 .....	23
14. 固件加载.....	24

14.1	BOOT/RECOVERY 分区 .....	24
14.2	KERNEL 分区 .....	24
14.3	RESOURCE 分区 .....	24
14.4	DTB 文件.....	25
14.5	固件加载流程.....	25
<b>15.</b>	<b>BOOT_MERGER 工具 .....</b>	<b>25</b>
15.1	支持 LOADER 的打包和解包 .....	26
15.1.1	打包: .....	26
15.1.2	解包: .....	26
15.2	参数配置文件.....	26
<b>16.</b>	<b>RESOURCE_TOOL 工具 .....</b>	<b>27</b>
16.1	支持 RESOURCE 镜像的打包和解包 .....	27
16.1.1	打包: .....	27
16.1.2	解包: .....	28

## 1. Uboot 简介

RK Uboot 是基于开源的 Uboot 进行开发的，主要支持：

- 支持芯片：rk3288 等；
- Kernel 和 ramdisk 的加载启动；
- 支持 Rockusb 和 fastboot 两种方式烧写；
- 支持 secure boot 签名加密机制；
- 支持 LVDS、EDP、MIPI、HDMI 等显示设备；
- EMMC、nand flash 等存储设备；
- 支持开机 logo 显示、充电动画显示；
- I2C、SPI、pmic、charge、guage、usb、gpio、pwm 等驱动支持；

## 2. Rockchip 平台相关代码

(1)、配置文件:

```
include\configs\rk32xx.h
```

(2)、芯片架构相关文件目录:

```
arch\arm\include\asm\arch-rk32xx\
```

```
arch\arm\cpu\armv7\rk32xx\
```

(3)、board 相关文件目录:

```
board\rockchip\
```

(4)、命令相关文件目录:

```
common\
```

(5)、驱动相关文件目录:

```
drivers\
```

(6)、工具相关文件目录:

```
tools\
```

```
tools\rk_tools\
```

### 3. 编译

(1) 编译 uboot 之前要设置 toolchain, 在 uboot 根目录下的 Makefile 中指定

```
ifneq ($(wildcard ../arm-eabi-4.7),)
CROSS_COMPILE ?= $(shell pwd)/../arm-eabi-4.7/bin/arm-eabi-
endif
ifneq ($(wildcard ../arm-eabi-4.6),)
CROSS_COMPILE ?= $(shell pwd)/../arm-eabi-4.6/bin/arm-eabi-
endif

ifneq ($(wildcard ../toolchain/arm-eabi-4.7),)
CROSS_COMPILE ?= $(shell pwd)/../toolchain/arm-eabi-4.7/bin/arm-eabi-
endif
ifneq ($(wildcard ../toolchain/arm-eabi-4.6),)
CROSS_COMPILE ?= $(shell pwd)/../toolchain/arm-eabi-4.6/bin/arm-eabi-
endif
ifneq ($(wildcard ../toolchain/arm-eabi-4.4.3),)
CROSS_COMPILE ?= $(shell pwd)/../toolchain/arm-eabi-4.4.3/bin/arm-eabi-
endif
ifneq ($(wildcard ../toolchain/arm-eabi-4.4.0),)
CROSS_COMPILE ?= $(shell pwd)/../toolchain/arm-eabi-4.4.0/bin/arm-eabi-
endif

ifneq ($(wildcard ../prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin),)
CROSS_COMPILE ?= $(shell pwd)/../prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-eabi-
endif
ifneq ($(wildcard ../android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3),)
CROSS_COMPILE ?= $(shell pwd)/../android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
endif
ifneq ($(wildcard ../android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0),)
CROSS_COMPILE ?= $(shell pwd)/../android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-
endif
```

(2) 配置, RK uboot 的配置文件为 include/configs/rk32xx.h, 相关模块的开关通过修改该文件来进行控制

make rk32xx\_config

(3) 编译

make

编译完成后生成的镜像:

RK3288Loader\_uboot\_V2.15.bin

或者

RK3288Loader\_miniall\_V2.15.bin 和 uboot.img

其中 V2.15 是发布的版本号, 这个是 rockchip 定义 uboot loader 的版本, 客户务必不要修改这个版本。

- RK3288Loader\_uboot\_V2.15.bin 是一级 loader 模式, 只支持 emmc。
- RK3288Loader\_miniall\_V2.15.bin 和 uboot.img 组合是二级 loader 模式, 同时支持 emmc 和 nand flash, 二级 loader 模式需要在 rk32xx.h 配置文件中定义:

```
#define CONFIG_SECOND_LEVEL_BOOTLOADER
```

## 4. Cache 机制

Rockchip 系列芯片 cache 接口 采用 uboot 提供的标准接口，具体可以参考 uboot 的文档，这里做简单的介绍。

```
include\configs\rk32xx.h 中关于 cache 的配置

/*
 * cache config
 */

#define CONFIG_SYS_ICACHE_OFF

#define CONFIG_SYS_DCACHE_OFF

#define CONFIG_SYS_L2CACHE_OFF

#define CONFIG_SYS_ARM_CACHE_WRITETHROUGH

arch\arm\lib\cache-cp15.c 是一些 cache 的使能和关闭函数

arch\arm\lib\cache.c 是一些 cache 的操作相关的函数
```

一般情况下 cache 由 rockchip uboot 开发人员维护，请谨慎修改。



## 5. 中断机制

Rockchip 平台支持标准的 uboot 中断接口函数：

```
void enable_interrupts (void);

int  disable_interrupts (void);


void irq_install_handler(int irq, interrupt_handler_t *handler, void *data);

void irq_uninstall_handler(int irq);

int irq_set_irq_type(int irq, unsigned int type);

int irq_handler_enable(int irq);

int irq_handler_disable(int irq);


static inline int gpio_to_irq(unsigned gpio);
```

## 6. Clock 驱动

Rk clock 相关代码位于:

```
arch/arm/include/asm/arch-rk32xx/clock.h
arch/arm/cpu/armv7/rk32xx/clock.c
arch/arm/cpu/armv7/rk32xx/clock-rk3288.c
```

主要的接口如下:

```
/*
 * rkplat clock set pll mode
 */
void rkclk_pll_mode(int pll_id, int pll_mode);
设置 pll 的模式, slow 或者 normal

/*
 * rkplat clock set for arm and general pll
 */
void rkclk_set_pll(void);
配置 rk 芯片的相关 pll

/*
 * rkplat clock get arm pll, general pll and so on
 */
void rkclk_get_pll(void);
获取 rk 芯片配置的 pll

int rkclk_get_arm_pll(void);
```

```
int rkclk_get_general_pll(void);

int rkclk_get_codec_pll(void);

int rkclk_get_ddr_pll(void);

int rkclk_get_new_pll(void);

获取相应 pll 的频率

/*

 * rkplat clock pll dump

 */

void rkclk_dump_pll(void);

/*

 * rkplat clock set codec pll

 */

void rkclk_set_cpll_rate(uint32 pll_hz);

/*

 * rkplat set sd clock src

 * 0: codec pll; 1: general pll; 2: 24M

 */

void rkclk_set_sdcclk_src(uint32 sdid, uint32 src);

/*

 * rkplat set sd/sdmmc/emmc clock src
```

```
*/

unsigned int rkclk_get_sdclk_src_freq(uint32 sdid);

/*

* rkplat set sd clock div, from source input

*/

int rkclk_set_sdclk_div(uint32 sdid, uint32 div);

void rkclk_emmc_set_clk(int div);

/*

* rkplat get PWM clock, PWM01 from pclk_cpu, PWM23 from pclk_periph

*/

unsigned int rkclk_get_pwm_clk(uint32 id);

/*

* rkplat get I2C clock, I2c0 and i2c1 from pclk_cpu, I2c2 and i2c3 from pclk_periph

*/

unsigned int rkclk_get_i2c_clk(uint32 i2c_bus_id);

/*

* rkplat get spi clock, spi0 and spi1 from pclk_periph

*/
```

```

unsigned int rkclk_get_spi_clk(uint32 spi_bus);

/*
 * rkplat lcdc aclk config
 * lcdc_id (lcdc id select) : 0 - lcdc0, 1 - lcdc1
 * pll_sel (lcdc aclk source pll select) : 0 - codec pll, 1 - general pll
 * div (lcdc aclk div from pll) : 0x00 - 0x1f
 */
int rkclk_lcdc_aclk_set(uint32 lcdc_id, uint32 pll_sel, uint32 div);

/*
 * rkplat lcdc dclk config
 * lcdc_id (lcdc id select) : 0 - lcdc0, 1 - lcdc1
 * pll_sel (lcdc dclk source pll select) : 0 - codec pll, 1 - general pll
 * div (lcdc dclk div from pll) : 0x00 - 0xff
 */
int rkclk_lcdc_dclk_set(uint32 lcdc_id, uint32 pll_sel, uint32 div);

/*
 * rkplat lcdc dclk and aclk parent pll source
 * lcdc_id (lcdc id select) : 0 - lcdc0, 1 - lcdc1
 * dclk_hz: dclk rate
 * return dclk rate

```

```
*/  
  
int rkclk_lcdc_clk_set(uint32 lcdc_id, uint32 dclk_hz);  
  
  
  
  
/*  
  
* rkplat pll select by clock  
  
* clock: device request freq HZ  
  
* return value:  
  
* high 16bit: 0 - codec pll, 1 - general pll  
  
* low 16bit : div  
  
*/  
  
uint32 rkclk_select_pll_source(uint32 clock, uint32 even);
```

一般情况下，无须修改 clock 的内容，采用我们建议的配置就可以。

## 7. I2C 驱动

Rk I2C 支持标准 uboot 架构，详细可以参考 uboot 文档，相关代码位于：

```
drivers\i2c\rk_i2c.c
```

注意：目前 i2c 读写最大长度为 32 字节，后续会完善驱动。

```
int i2c_set_bus_num(unsigned bus_idx)
```

设置即将操作的 i2c 总线，这个要最先配置。

```
void i2c_init(int speed, int unused)
```

初始化对应总线的 i2c

```
int i2c_set_bus_speed(unsigned int speed)
```

配置你需要的 i2c 总线频率

```
int i2c_probe(uchar chip)
```

侦测指定的 i2c 地址的设备是否存在

```
int i2c_read(uchar chip, uint addr, int alen, uchar *buf, int len)
```

i2c 读取操作

```
int i2c_write(uchar chip, uint addr, int alen, uchar *buf, int len)
```

I2C 写入操作

## 8. SPI 驱动

Rk SPI 支持标准 uboot 架构，详细可以参考 uboot 文档，相关代码位于：

```
drivers\spi\rk_spi.c
```



## 9. LCD 驱动

关于屏的电源控制：在 rk32xx.c 中 rk\_fb\_init 函数，会解析 dts 中的 lcd\_en 和 lcd\_cs 两个 gpio 配置和 delay 时长，如果屏的上电时序需要重新调整，可以修改该函数。如果产品板子还有更多电源控制的 gpio 或者需要 I2C 控制的 ldo，都可以加在这里。

Uboot 使用的 logo 存放在 kernel 的根目录下，编译时会打包进 resource.img 文件中，Uboot 对 logo 的解析过程请参考 common/lcd.c 中的 rk\_bitmap\_from\_resource ( ) 函数。如果 rk\_bitmap\_from\_resource 解析 resource 中的 logo 失败，会继续执行 uboot 的默认 logo 显示流程，所以即使 kernel 目录中不包含 logo.bmp 图片，uboot 也会在屏中间显示一张如下所示的图：



这张名为 rockchip.bmp 的图保存在 /u-boot/tools/logos 目录下，为了尽量减小 uboot.bin 的 size，这张图做得很小（只有 200x500,8bit 的 bmp 图片），如果想替换开机 logo，建议修改 kernel 中的 logo.bmp 而不是修改 uboot 的 rockchip.bmp。

另外需要强调的是，uboot 对 bmp 的支持比较弱，目前知道有如下限制：

- (1) 只支持偶数分辨率的图片
- (2) 所有的 bmp 图片建议用如下命令进行处理后，将处理后的 logo\_rle8.bmp 用于显示

```
convert -compress rle -colors 256 logo.bmp logo_rle8.bmp
```

## 10. 电源相关的控制

如果要在 Uboot 里面实现开机 LOGO，充电动画等功能，则需要对系统进行电源相关的控制。这些功能在 PMU 驱动中实现。

目前 RK uboot 中已经自动兼容 Ricoh619、ACT8846、RK808 三款 PMU，相关代码如下：

```
drivers/power/power_core.c  
  
drivers/power/power_rockchip.c  
  
drivers/power/pmic/pmic_act8846.c  
  
drivers/power/pmic/pmic_ricoh619.c  
  
drivers/power/pmic/pmic_rk808.c
```

其中 power\_core.c 是 uboot power 子系统的核心代码，提供对 pmic、charger、fuel gauge 进行管理的接口。Power\_rockchip.c 是对 Rockchip 平台 pmic、charger、fuel gauge 进行兼容的框架层代码。向上提供统一接口供系统调用，向下对各种电源 IC 进行管理。其他的为各个 PMIC 驱动。

系统启动的时候，在 rk32xx.c 文件中 board\_late\_init(void)中通过 pmic\_init 系统调用，对 PMIC 进行基本的初始化。

## 11. 电量计驱动

如果要在 uboot 中实现充电等功能，需要加入电量计（fuel\_gauge）的支持。目前 RK uboot 中支持 Ricoh619、cw201x 两款电量计：

```
drivers/power/fuel_gauge/fg_cw201x.c
```

```
drivers/power/fuel_gauge/fg_cw201x.c
```

系统启动的时候，在 rk32xx.c board\_late\_init 中通过调用 fg\_init 接口，对电量计进行初始化。

fg\_init 在 power\_rockchip.c 中实现：

```
int Fg_init(unsigned char bus)
{
    int ret;
    #if defined(CONFIG_POWER_FG_CW201X)
    ret = fg_cw201x_init(bus);
    if (ret >= 0) {
        printf("fg:cw201x\n");
        return 0;
    }
    #endif
    return 0;
}
```

对于一款电量计驱动，需要实现如下重要接口：

- (1) fg\_xxx\_init(): 该函数进行基本的 fuel gauge 初始化和注册：

```
int fg_cw201x_init(unsigned char bus)
{
    static const char name[] = "CW201X_FG";
    int ret;
    if (!cw.p) {
        ret = cw201x_parse_dt(gd->fdt_blob);
        if (ret < 0)
            return ret;
    }

    cw.p->name = name;
    cw.p->interface = PMIC_I2C;
    cw.p->fg = &cw201x_fg_ops;
    return 0;
}
```

name 用于对 PMIC 的查找（在 uboot 的 power 系统中，pmic、charger、fuel\_gauge 统一抽象为 pmic 设备）。所以这里的名称还要在 power\_rockchip.c 的 fg\_names 中注册，以供系统查找。

```
static const char * const fg_names[] = {
    "CW201X_FG",
    "RICOH619_FG",
};
```

Fg 的 ops 接口，主要用于获取电池的电量 and 充电状态

```
static struct power_fg cw201x_fg_ops = {
    .fg_battery_check = cw201x_check_battery,
    .fg_battery_update = cw201x_update_battery,
};
```

其中 fg\_battery\_update 接口用于更新电池的电量、电压等信息，fg\_battery\_check 接口用于获取电池是否充电等状态。

为了实现充电动画，需要在 rk32xx.h 中打开如下开关

```
#define CONFIG_UBOOT_CHARGE

#define CONFIG_CMD_CHARGE_ANIM

#define CONFIG_CHARGE_DEEP_SLEEP

#define CONFIG_SCREEN_ON_VOL_THRESD          3550

#define CONFIG_SYSTEM_ON_VOL_THRESD          3650
```

其中 CONFIG\_SCREEN\_ON\_VOL\_THRESD 是系统点亮屏幕的电压门限，低于这个电压，禁止系统亮屏。CONFIG\_SYSTEM\_ON\_VOL\_THRESD 是系统正常启动的电压门限，低于这个电压，禁止 uboot 启动内核。这两个电压可以根据具体的产品设计灵活调整。

对这两个门限的判断在 power\_rockchip.c 中实现：

```
/*system on thresd*/
int is_power_low(void)
{
    int ret;
    struct battery battery;
    memset(&battery, 0, sizeof(battery));
    ret = get_power_bat_status(&battery);
    if (ret < 0)
        return 0;
    return (battery.voltage_uV < CONFIG_SYSTEM_ON_VOL_THRESD) ? 1:0;
}

/*screen on thresd*/
int is_power_extreme_low(void)
{
    int ret;
    struct battery battery;
    memset(&battery, 0, sizeof(battery));
    ret = get_power_bat_status(&battery);
    if (ret < 0)
        return 0;
    return (battery.voltage_uV < CONFIG_SCREEN_ON_VOL_THRESD) ? 1:0;
}
```

另外，要显示充电动画和开机 logo 还要在 dts 里面把 uboot-logo-on 属性置 1:

```
&fb {
    rockchip,disp-mode = <DUAL>;
    rockchip,uboot-logo-on = <1>;
};
```

## 12.emmc 及 nand 的驱动

- (1) uboot 支持 emmc 及 nand flash, 想要 uboot 同时支持 emmc 和 nand flash, 并且开机时自动识别, 必须在 rk32xx.h 中定义 CONFIG\_SECOND\_LEVEL\_BOOTLOADER, 否则 uboot 只支持 emmc。
- (2) 存储模块的代码位置在 board/rockchip/common/storage/storage.c 中, 初始化入口为 int32 StorageInit(void)。如果碰到初始化失败的情况, 可注意排查下硬件的焊接情况。

## 13.Fastboot

Fastboot 是 loader 提供的一种类似 rockusb/adb 的交互模式。Fastboot 交互中, 使用的 PC 工具源码位于 android 源码中 (system/core/fastboot/), 分为 windows 版本和 linux 版本 (**windows 端使用的设备驱动与 adb 相同**)。

### 13.1 进入 fastboot 状态的方式

- 1、开机中 loader 启动阶段按键进入 (**3288sdk 板为 vol-键**):

```
checkKey((uint32 *)&boot_rockusb, (uint32 *)&boot_recovery, (uint32 *)&boot_fastboot);

....

} else if(boot_fastboot && (vbus!=0)){

    printf("fastboot key pressed.\n");

    frt = FASTBOOT_REBOOT_FASTBOOT;

}
```

- 2、带有 fastboot 参数的 reboot 命令 (reboot fastboot, 通过 PMU\_SYS\_REG0 寄存器传递)

### 13.2 Fastboot 主要支持命令

#### 13.2.1 获取信息:

fastboot getvar version

获得版本

fastboot getvar version-bootloader	获得版本
fastboot getvar unlocked	获得解锁情况
fastboot getvar secure	获得锁住情况（与 unlock 相反）
fastboot getvar product	获得产品信息
fastboot getvar serialno	获得序列号
fastboot getvar partition-type:<partition_name>	获得指定分区类型
fastboot getvar partition-size:<partition_name>	获得指定分区大小
fastboot getvar partition-offset:<partition_name>	获得指定分区偏移

### 13.2.2 烧写:

- fastboot flash <partition\_name> <filename> 烧写固件

（如：fastboot flash system system.img。

烧写 parameter/loader 时，需指定分区名为“parameter”/“loader”）

- fastboot update <filename> 烧写升级包

（升级包通过在 android 源码中 make updatepackage 生成）

### 13.2.3 重启:

fastboot oem recovery	重启进 recovery
fastboot oem recovery:wipe_data	重启恢复出厂设置
fastboot reboot	重启
fastboot reboot-bootloader	重启进入 rockusb 烧写模式
fastboot continue	重启

### 13.2.4 解锁和锁住设备:

- fastboot oem unlock 解锁
- fastboot oem unlock\_accept 确认解锁

(需要在 fastboot oem unlock 命令后, 5 秒内输入)

fastboot oem lock

## 锁住设备

### 13.2.5 特殊命令:

```
fastboot boot <kernel> [ <ramdisk> ]
```

## 临时从指定固件启动

(kernel 目前支持 Image/zImage, 需要将 dtb 存于 kernel 末尾, 或者 resource 分区中)

fastboot oem log

## 获取串口 log 信息

```
fastboot oem ucmd <uboot cmds>
```

## 运行 uboot 命令

## 13.3 Fastboot 解锁

**fastboot 锁住状态下，不允许烧写及执行 oem 命令，初始状态为锁住。**

解锁流程大致如下：

- 1、执行 `fastboot oem unlock`
- 2、5 秒内继续执行 `fastboot oem unlock_accept`
- 3、机器会重启进入 `recovery` 恢复出厂设置
- 4、再次进入 `fastboot`，则 `fastboot getvar unlocked` 应该返回"yes"（设备已解锁）

如果设备进入 fastboot 状态后，fastboot 命令提示未发现设备，则需要在命令中加入-i 参数指定设备 vid，例如 fastboot -i 0x2207 getvar unlocked

## 14. 固件加载

固件加载涉及到 boot、recovery、kernel、resource 分区以及 dtb 文件。

### 14.1 boot/recovery 分区

Boot 和 recovery 的固件分为两种形式：

#### (A)、android 标准格式

标准固件格式将 ramdisk 和 kernel 打包在一起，镜像文件的魔数为"ANDROID!"：

00000000	41 4E 44 52	4F 49 44 21	24 10 74 00	00 80 40 60	ANDROID!\$.t...@`
00000010	F9 31 CD 00	00 00 00 62	00 00 00 00	00 00 F0 60	.1.....b.....`

标准格式可以带有签名、checksum 等信息，以及 dtb 文件等额外数据。打包固件时，recovery 镜像默认为标准格式，而标准格式的 boot 镜像则需要通过 ./mkimage.sh ota 方式生成。

#### (B)、RK 格式

Rk 格式的镜像单独打包一个文件 (ramdisk/kernel)，镜像文件的魔数为"KRNL"：

00000000	4B 52 4E 4C	42 97 0F 00	1F 8B 08 00	00 00 00 00	KRNLB.....
00000010	00 03 A4 BC	0B 78 53 55	D6 37 BE 4F	4E D2 A4 69	.....xSU.7.ON..i

打包生成的 kernel.img、默认打包方式生成的 boot.img 均为 Rk 格式。

### 14.2 kernel 分区

Kernel 分区包含 kernel 信息。如果启动时，加载的 boot/recovery 分区自身带有 kernel (android 标准格式)，则忽略 kernel 分区，优先使用其自身包含的 kernel。

### 14.3 resource 分区

Resource 镜像格式是为了简单存取多个资源文件设计的简易镜像格式，其魔数为"RSCE"：



00000000	52 53 43 45	00 00 00 00	01 01 01 00	01 00 00 00	RSCE.....
00000010	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

Uboot 支持将 kernel 所需的 dtb 打包在 resource 分区。

## 14.4 Dtb 文件

Dtb 文件是新版本 kernel 的 dts 配置文件的二进制化文件。

目前 dtb 文件可以存放于 android 标准格式的 boot/recovery 分区中，也可以存放于 resource 分区。

uboot 假定 kernel 启动必须加载 dtb 文件。

## 14.5 固件加载流程

Uboot 加载固件流程为：

- 1、加载需要启动的 boot/recovery 分区的 ramdisk 内容
- 2、加载启动分区的 kernel 内容。如果失败（为 Rk 格式），则继续加载 kernel 分区
- 3、加载启动分区的 dtb 文件。如果失败，则继续尝试从 resource 分区加载。

Dtb 文件（fdt）和 ramdisk 将被加载到 uboot 动态申请的内存中。Kernel 则被加载到内存 32M 偏移处运行。

## 15.boot\_merger 工具

boot\_merger 是用于打包 loader、ddr bin、usb plug bin 等文件，生成烧写工具需要的 loader 格式的 linux 版本工具。其源码位于 uboot 源码内：

```
uboot# ls ./tools/boot_merger.*
./tools/boot_merger.c  ./tools/boot_merger.h
```

## 15.1 支持 loader 的打包和解包

### 15.1.1 打包:

```
./tools/boot_merger [--pack] <config.ini>
```

打包需要传递描述打包参数的 ini 配置文件路径。

(目前使用的配置文件均存放于 uboot 源码内 (tools/rk\_tools/RKB00T)) 如:

```
./tools/boot_merger ./tools/rk_tools/RKB00T/RK3288.ini
out:RK3288Loader_uboot.bin
fix opt:RK3288Loader_uboot_V2.15.bin
merge success(RK3288Loader_uboot_V2.15.bin)
```

### 15.1.2 解包:

```
./tools/boot_merger --unpack <loader.bin>
```

## 15.2 参数配置文件

以 3288 的配置文件为例:

[CHIP\_NAME]

NAME=RK320A -----芯片名称: "RK" 加上与 maskrom 约定的 4B 芯片型号

[VERSION]

MAJOR=2 -----主版本号

MINOR=15 -----次版本号

[CODE471\_OPTION] -----code471, 目前设置为 ddr bin

NUM=1

Path1=tools/rk\_tools/32\_LPDDR2\_300MHz\_LPDDR3\_300MHz\_DDR3\_300MHz\_20140404.b  
in

[CODE472\_OPTION] -----code472, 目前设置为 usbplug bin

```
NUM=1

Path1=tools/rk_tools/rk32xxusbplug.bin

[LOADER_OPTION]

NUM=2

LOADER1=FlashData      ----flash data, 目前设置为 ddr bin
LOADER2=FlashBoot      ----flash boot, 目前设置为 uboot bin

FlashData=tools/rk_tools/32_LPDDR2_300MHz_LPDDR3_300MHz_DDR3_300MHz_201404
04.bin

FlashBoot=u-boot.bin

[OUTPUT]                ----输出路径, 目前文件名会自动添加版本号

PATH=RK3288Loader_uboot.bin
```

## 16.Resource\_tool 工具

resource\_tool 是用于打包任意资源文件, 生成 resource 镜像的 linux 工具。  
其源码位于 uboot 源码内 (tools/resource\_tool/)

### 16.1 支持 resource 镜像的打包和解包

#### 16.1.1 打包:

```
./tools/resource_tool [--pack] [--image=<resource.img>] <file list>
```

如:

```
uboot/tools/resource_tool/resources# ../resource_tool `find . -type f`
Pack to resource.img succeeded!
```

pack\_resource.sh 脚本可以新增资源文件到现有的镜像:

```
./pack_resource <resources dir> <old image> <dst image> <resource_tool path>
```

如:

```
uboot# sudo ./tools/resource_tool/pack_resource.sh  
tools/resource_tool/resources/ ../kernel/resource.img resource.img  
tools/resource_tool/resource_tool  
Pack tools/resource_tool/resources/ & ../kernel/resource.img to  
resource.img ...
```

Unpacking old image(../kernel/resource.img):

rk-kernel.dtb

Pack to resource.img succeeded!

Packed resources:

```
rk-kernel.dtb charge_anim_desc.txt images/battery_4.bmp  
images/battery_0.bmp images/battery_1.bmp images/battery_2.bmp  
images/battery_3.bmp images/battery_5.bmp images/battery_fail.bmp  
resource.img
```

### 16.1.2 解包:

```
./tools/resource_tool --unpack [--image=<resource.img>] [output dir]
```