Rockchip
瑞芯微电子

Rockchip
U-Boot 开发指南

发布版本:3.7

日期:2016.07

前言

概述

产品版本

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
2014-06-18	V1.0		初始版本
2014-09-03	V2.0		支持 RK312X
2014-10-11	V3.0		采用新架构，方便不同平台开发
2014-12-01	V3.1		RK312X ADC 检测充电动画
2014-12-11	V3.2		AudiB 支持，24Bit bmp logo 支持，DRM KeyBox 传递错误等。
2014-12-12	V3.3		支持内核显示新 logo.
2015-04-09	V3.4		支持 USB 启动和升级
2015-04-14	V3.5		RK3368 armv8 64 架构支持
2015-12-18	V3.6		支持 RK322x
2016-07-06	V3.7	CWZ	整理文档，添加 RK3366、RK3399

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

目录

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

1 Rockchip U-Boot 简介

Rockchip U-Boot 是基于开源的 UBoot 2014.10 正式版进行开发的，主要支持：

- 支持芯片：rk3288、rk3036、rk312x、rk3368、rk322x、rk3366、rk3399 等；
- 支持 Android 平台的固件启动；
- 支持 ROCKUSB 和 Google Fastboot 两种方式烧写；
- 支持 secure boot 固件签名加密保护机制；
- 支持 LVDS、EDP、MIPI、HDMI、CVBS 等显示设备；
- 支持 SDCard、Emmc、Nand Flash、U 盘等存储设备；
- 支持开机 logo 显示、充电动画显示，低电管理、电源管理；
- 支持 I2C、SPI、PMIC、CHARGE、GUAGE、USB、GPIO、PWM、DMA、GMAC、EMMC、NAND 中断等驱动；

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

2 平台架构支持

Rockchip U-Boot 基于 U-Boot 2014.10 官方版本开发的，并同步更新主分支的一些关键性更新，支持全新的配置和编译架构，方便多平台同时开发，本文档是基于 U-Boot 2014.10 的框架开发设计的。

2.1 编译配置

2.1.1 工具链配置

Rockchip U-Boot 默认使用 Google Android 系统提供的 GCC ToolChain，在 U-BOOT 根目录下的 Makefile 中指定：

```
ifeq ($(ARCHV),aarch64)

ifneq ($(wildcard ../toolchain/aarch64-linux-android-4.9),)
CROSS_COMPILE   ?= $(shell pwd)/../toolchain/aarch64-linux-android-4.9/bin/aarch64-linux-android-
endif
ifneq ($(wildcard ../prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin),)
CROSS_COMPILE   ?= $(shell pwd)/../prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin/aarch64-linux-android-
endif

else

ifneq ($(wildcard ../toolchain/arm-eabi-4.8),)
CROSS_COMPILE   ?= $(shell pwd)/../toolchain/arm-eabi-4.8/bin/arm-eabi-
endif
ifneq ($(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 ../prebuilts/gcc/linux-x86/arm/arm-eabi-4.8/bin),)
CROSS_COMPILE   ?= $(shell pwd)/../prebuilts/gcc/linux-x86/arm/arm-eabi-4.8/bin/arm-eabi-
endif
ifneq ($(wildcard ../prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin),)
CROSS_COMPILE   ?= $(shell pwd)/../prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin/arm-eabi-
endif
ifneq ($(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

endif # ARCHV=aarch64
```

注意：ARCHV 区分 64 位和 32 位的 GCC 工具链，Make 时添加参数 ARCHV=aarch64。

2.1.2 平台配置

平台配置文件位于 U-Boot 根目录下的 configs 文件夹下，其中 Rockchip 相关的以 RK 开头，并根据产品形态分为 MID 和 BOX 两种配置：

```
rk3288_defconfig
rk3126_defconfig
rk3128_defconfig
rk3368_defconfig

rk3288_box_defconfig
rk3128_box_defconfig
rk3036_box_defconfig
rk3368_box_defconfig
rk322x_box_defconfig
```

Rockchip 芯片平台的配置，主要是芯片类型，Rockchip 一些 Kconfig 的关键配置，采用 savedefconfig 模式保存。

以 rk322x_box_defconfig 为例，说明相关配置的含义

```
CONFIG_SYS_EXTRA_OPTIONS="RKCHIP_RK322X,PRODUCT_BOX,NOR
MAL_WORLD,SECOND_LEVEL_BOOTLOADER,BAUDRATE=1500000"
CONFIG_ARM=y
CONFIG_ROCKCHIP=y
CONFIG_ROCKCHIP_ARCH32=y
CONFIG_SYS_EXTRA_OPTIONS="RKCHIP_RK322X,PRODUCT_BOX,NORMAL_WO
RLD,SECOND_LEVEL_BOOTLOADER, BAUDRATE=1500000"
```

该选项内的配置会被优先编译成宏定义并在相关的项前面自动添加 CONFIG_，可以在 U-BOOT 自动生成的配置文件(include/config.h)中看到生成的宏定义，会优先系统的配置文件，可以支配系统的配置文件。

RKCHIP_RK322X：定义 RK322x 的芯片类型，编译扩展为 CONFIG_ RKCHIP_RK322X;

PRODUCT_BOX：定义 BOX 产品，目前的产品形态有 MID 和 BOX 两种，编译扩展为 CONFIG_PRODUCT_BOX;

NORMAL_WORLD：定义 U-BOOT 运行在 normal world。

SECOND_LEVEL_BOOTLOADER： 定义 U-BOOT 作为二级 loader 模式，采用 NAND Flash 的项目以及安全框架驱动时，需要定义该选项。编译扩展为 CONFIG_SECOND_LEVEL_BOOTLOADER;

BAUDRATE=1500000：定义调试串口的波特率，默认是 115200，这里配置成 1.5M。编译扩展为 CONFIG_BAUDRATE=1500000。

CONFIG_ARM=y

定义 ARM 平台。

CONFIG_ROCKCHIP=y

定义 Rockchip 的平台。

CONFIG_ ROCKCHIP_ARCH32=y

定义 RK 芯片系列类型的平台，RK30 系列、RK32 系列等 32 位芯片。

CONFIG_ ROCKCHIP_ARCH64=y

定义 RK 芯片系列类型的平台，RK33 系列等 64 位芯片。

2.1.3 系统配置

系统配置文件位于 U-Boot 根目录下的 **include\configs** 文件夹下，同样以 RK 开头：

```
rk_default_config.h
rk30plat.h
rk32plat.h
rk33plat.h
```

- **rk_default_config.h**：RK 平台的公共配置，默认打开所有需要的功能。
- **rk30plat.h**：RK30 系列平台的配置，根据不同芯片进行一些细节的配置，如内存地址、简配一些功能模块的配置，RK30 系列包含 RK3036、RK3126、RK3128、RK322x 等芯片。
- **rk32plat.h**：RK32 系列平台的配置，根据不同芯片进行一些细节的配置，如内存地址、简配一些功能模块的配置，RK32 系列包含 RK3288。
- **rk33plat.h**：RK33 系列平台的配置，根据不同芯片进行一些细节的配置，如内存地址、简配一些功能模块的配置，RK33 系列包含 RK3368、RK3366、RK3399 等。

注意：这些文件中不能使用 // 作为注释，会引起 U-BOOT 解析 lds 文件时出错，如果不想定义相关模块，可以直接使用 // 或者 #undef 等。**

关键性的系统配置说明：

```
    /*
     *           UBOOT memory map
     *
     * CONFIG_SYS_TEXT_BASE is the default address which maskrom
loader UBOOT code.
     * CONFIG_RKNAND_API_ADDR is the address which maskrom loader
miniloader code.
     *
     *  kernel load address: CONFIG_SDRAM_PHY_START + 32M, size 16M,
     *  miniloader code load address: CONFIG_SDRAM_PHY_START + 48M,
size 8M,
     *  total reverse memory is CONFIG_LMB_RESERVE_MEMORY_SIZE.
     *
     *|-------------------------------------------------------------------------|
     *|START  -  KERNEL LOADER  -  NAND LOADER  -     LMB     -
UBOOT    -  END|
     *|SDRAM  -    START 32M    -   START 48M   -  START 56M  -
START 80M  - 128M|
     *|       -      kernel      -   nand code   -     fdt     -
UBOOT/ramdisk   |
     *|-------------------------------------------------------------------------|
     */
    /* rk kernel load address */
    #define CONFIG_KERNEL_LOAD_ADDR     (CONFIG_RAM_PHY_START
+ SZ_32M) /* 32M offset */

    /* rk nand api function code address */
    #define CONFIG_RKNAND_API_ADDR      (CONFIG_RAM_PHY_START
+ SZ_32M + SZ_16M) /* 48M offset */

    /* rk UBOOT reserve size */
    #define CONFIG_LMB_RESERVE_SIZE     (SZ_32M + SZ_16M +
SZ_8M) /* 56M offset */
```

CONFIG_KERNEL_LOAD_ADDR 配置 Kernel 加载的地址，DDR 偏移 32M 的位置。

CONFIG_RKNAND_API_ADDR 配置 Nand Flash 驱动 API 接口地址，DDR 偏移 48M 的位置。

CONFIG_LMB_RESERVE_SIZE 配置 U-BOOT Reserve 空间大小。

```
    /* rk ddr information */
    #define CONFIG_RK_MAX_DRAM_BANKS   8 /* rk ddr max banks */
    #define CONFIG_RKDDR_PARAM_ADDR
    (CONFIG_RAM_PHY_START + SZ_32M) /* rk ddr banks address and size
*/
    #define CONFIG_RKTRUST_PARAM_ADDR  (CONFIG_RAM_PHY_START
+ SZ_32M + SZ_2M) /* rk trust os banks address and size */
```

```
/* rk hdmi device information buffer (start: 128M - size: 8K) */
#define CONFIG_RKHDMI_PARAM_ADDR    CONFIG_RAM_PHY_END
```

CONFIG_RK_MAX_DRAM_BANKS 配置 DDR 驱动传递 DDR 容量信息中 DDR 的 Banks 数目。

CONFIG_RKDDR_PARAM_ADDR 配置 DDR 驱动传递容量信息的地址，偏移 32M 的位置。

CONFIG_RKTRUST_PARAM_ADDR 配置使用 ARM Trusted Firmware 时需要内核保留的空间信息。

CONFIG_RKHDMI_PARAM_ADDR 配置 HDMI 的一些信息需要内核获取的地址。

```
#define CONFIG_RAM_PHY_START      0x00000000
#define CONFIG_RAM_PHY_SIZE       SZ_128M
#define CONFIG_RAM_PHY_END        (CONFIG_RAM_PHY_START +
CONFIG_RAM_PHY_SIZE)
```

这三个宏定义了整个 U-Boot 用到的 SDRAM 的空间范围。

```
#define CONFIG_BOOTDELAY        0
#define CONFIG_BOARD_DEMO
#define CONFIG_RK_IO_TOOL
```

CONFIG_BOOTDELAY 配置 U-Boot 是否在 Shell 中启用启动延时，默认 0 代表没有延时直接启动，如果需要延时 3 秒则定义为 3。

CONFIG_RK_IO_TOOL 配置 U-Boot Shell 下运行简单的 IO Tool 工具。

CONFIG_BOARD_DEMO 配置是否开启简单的测试 Demo 程序。

```
#define CONFIG_MAX_MEM_ADDR      RKIO_IOMEMORYMAP_START
```

CONFIG_MAX_MEM_ADDR 定义了 SOC 能够访问 DDR 的最大地址物理空间，如果 DDR 的实际容量大于该值，那么该宏会起作用并最终由 DTB 传递给内核。

```
#define CONFIG_SYS_TEXT_BASE     0x00200000 /* Resersed 2M
space Runtime Firmware bin. */
```

CONFIG_SYS_TEXT_BASE 定义了 U-BOOT 运行的起始空间。

```
#define CONFIG_KERNEL_RUNNING_ADDR(CONFIG_SYS_TEXT_BASE
+ SZ_512K)
```

CONFIG_KERNEL_RUNNING_ADDR 是指定拷贝 kernel 的地址，如果定义了该宏，那么 U-BOOT 在启动的时候会直接将 kernel 拷贝到该位置并启动，否则默认将拷贝到 DDR 偏移 32M 的位置，定义该宏一般是使用没有压缩的 kernel，加快开机速度。

```
CONFIG_RK_SDCARD_BOOT_EN
CONFIG_RK_SDMMC_BOOT_EN
CONFIG_RK_SDHCI_BOOT_EN
CONFIG_RK_FLASH_BOOT_EN
CONFIG_RK_UMS_BOOT_EN
```

这几个宏定义了 SOC 支持的存储设备，define 使能，undef 关闭。

```
CONFIG_MERGER_TRUSTOS
CONFIG_RK_TOS_WITH_TA
```

CONFIG_MERGER_TRUSTOS 配置 UBOOT 是否合并 trust image。

CONFIG_RK_TOS_WITH_TA 配置 trust image 是否是带有 ta 的固件，armv7 平台会用到改宏。

```
#define CONFIG_SECUREBOOT_CRYPTO
#define CONFIG_SECUREBOOT_SHA256
```

CONFIG_SECUREBOOT_CRYPTO 配置是否打开 crypto 硬件解密功能，现在只要芯片带有

crypto 模块，我们基本都会打开，加速开机时 boot image 的 SHA 校验

CONFIG_SECUREBOOT_SHA256 是开启 SHA256 功能，后续的一些产品安全要求越来越高，希望引入 sha256。

```
#define CONFIG_NORMAL_WORLD
#define CONFIG_SECURE_RSA_KEY_IN_RAM
#define CONFIG_SECURE_RSA_KEY_ADDR
(CONFIG_RKNAND_API_ADDR + SZ_2K)
```

Rockchip SOC 引入 ARM Trusted Fireware 之后，U-BOOT 工作在 normal word，那么 secure efuse 无法直接访问，所以需要 miniloader 在 DDR 中传递 public key 给 U-BOOT 用于固件校验。

```
CONFIG_LCD
CONFIG_RK_POWER
CONFIG_PM_SUBSYSTEM
CONFIG_RK_CLOCK
CONFIG_RK_IOMUX
CONFIG_RK_I2C
CONFIG_RK_KEY
CONFIG_RK_EFUSE
CONFIG_CMD_ROCKUSB
CONFIG_CMD_FASTBOOT
CONFIG_RK_MCU
```

配置相应的模块是否开启。

2.1.4 系统编译

32 位平台以 RK3288 为例：

```
make rk3288_defconfig
make
```

64 位平台以 RK3368 为例：

```
make rk3368_defconfig
make ARCHV=aarch64
```

Rockchip U-BOOT 提供了两个编译脚本：mkv7.sh - 编译 32 位 SOC 和 mkv8.sh - 编译 64 位 SOC。

2.2 架构文件

Rockchip SOC 架构相关文件目录：

```
arch\arm\include\asm\arch-rk32xx\
arch\arm\cpu\armv7\rk32xx\

arch\arm\include\asm\arch-rk33xx\
arch\arm\cpu\armv8\rk33xx\

board\rockchip\
```

命令相关文件目录：

```
common\
```

驱动相关文件目录：

```
drivers\
```

工具相关文件目录：

```
tools\
tools\rk_tools\
```

2.3 固件生成

Rockchip 平台 Loader 分为一级模式和二级模式，根据不同的平台配置生成相应的 Loader 固件。通过宏 CONFIG_SECOND_LEVEL_BOOTLOADER 的定义二级 Loader 模式。

2.3.1 一级 Loader 模式

U-BOOT 作为一级 Loader 模式，那么仅支持 EMMC 存储设备，编译完成后生成的镜像：

```
RK3288LoaderU-BOOT_V2.17.01.bin
```

其中 V2.17.01 是发布的版本号，rockchip 定义 U-Boot loader 的版本，其中 2.17 是根据存储版本定义的，客户务必不要修改这个版本，01 是 U-Boot 定义的小版本，用户根据实际需求在 Makefile 中修改。

2.3.2 二级 Loader 模式

U-Boot 作为二级 Loader 模式，那么固件支持所有的存储设备，该模式下，需要 MiniLoader 支持，通过宏 CONFIG_MERGER_MINILOADER 进行配置生成。同时引入 Arm Trusted Firmware 后会生成 trust image，这个通过宏 CONFIG_MERGER_TRUSTIMAGE 进行配置生成。

以 rk322x 编译生成的镜像为例：

```
RK322XMiniLoaderAll_V2.31.bin
UBOOT.img
trust.img
```

其中 V2.31 是发布的版本号，rockchip 定义 U-Boot loader 的版本，其中 2.31 是根据存储版本定义的，客户务必不要修改这个版本。

UBOOT.img 是 U-Boot 作为二级 loader 的打包。

trust.img 是 U-Boot 作为二级 loader 的打包。

RK3036、RK3126、RK3128、RK322x、RK3368、RK3366、RK3399 等采用二级 loader 模式。

3 Cache 机制

Rockchio 系列芯片 cache 接口采用 U-Boot 提供的标准接口，具体可以参考 U-Boot 的官方文档，这里做简单的介绍。

cache 相关的一些配置说明：

- CONFIG_SYS_ICACHE_OFF
  icache 开关，如果定义该宏表示关闭 icache 功能。
- CONFIG_SYS_DCACHE_OFF
  dcache 开关，如果定义该宏表示关闭 dcache 功能。注意：开启 dcache 前要配置 mmu。
- CONFIG_SYS_ L2CACHE _OFF
  L2 cache 开关。
- dcache 模式配置
  CONFIG_SYS_ARM_CACHE_WRITETHROUGH 配置 dcache writethrouch 模式；
  CONFIG_SYS_ARM_CACHE_WRITEALLOC 配置 dcache writealloc 模式；默认为 dcache writeback 模式

icache 的一些常用接口：

```
void icache_enable (void);
void icache_disable (void);
```

dcache 的一些常用接口：

```
void dcache_enable (void);
void dcache_disable (void);

void    flush_cache (unsigned long, unsigned long);
void    flush_dcache_all(void);
void    flush_dcache_range(unsigned long start, unsigned long stop);

void    invalidate_dcache_range(unsigned long start, unsigned long
stop);
void    invalidate_dcache_all(void);
void    invalidate_icache_all(void);
```

4 驱动支持

Rockchip 支持 i2c、spi、pmic、charge、guage、usb、gpio、pwm、dma、gmac、emmc、nand 中断等驱动。

4.1 中断机制

Rockhip 平台支持标准的 U-Boot 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);
```

4.2 Clock 驱动

Rockchip 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
arch\arm\cpu\armv7\rk32xx\ clock-rk3036.c
arch\arm\cpu\armv7\rk32xx\ clock-rk312x.c

arch\arm\include\asm\arch-rk33xx\clock.h
arch\arm\cpu\armv8\rk33xx\clock.c
arch\arm\cpu\armv8\rk33xx\ clock-rk3368.c
```

主要的接口函数定义见 arch\arm\include\asm\arch-rk32xx\clock.h，详细请看注释。

```
/*
 * 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

    /*
     * rkplat clock pll dump
     */
    void rkclk_dump_pll(void);

    /*
     * rkplat set sd clock src
     * 0: codec pll; 1: general pll; 2: 24M
     */
    void rkclk_set_sdclk_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。

4.3 GPIO 驱动

Rockhip 平台 gpio 接口函数在文件：drivers\gpio\rk_gpio.c 中，RK GPIO 的定义规则如下：

```
/*
 * rk gpio api define the gpio format as:
 * using 32 bit for rk gpio value,
 * the high 24bit of gpio is bank id, the low 8bit of gpio is pin number
```

```
     * eg: gpio = 0x00010008, it mean gpio1_b0, 0x00010000 is bank id of
GPIO_BANK1, 0x00000008 is GPIO_B0
     */

    /* bank and pin bit mask */
    #define RK_GPIO_BANK_MASK0xFFFFFF00
    #define RK_GPIO_BANK_OFFSET   8
    #define RK_GPIO_PIN_MASK   0x000000FF
    #define RK_GPIO_PIN_OFFSET0
```

采用 32bit 定义，高 24bit 为 bank，低 8 位为具体的 pin，如：

```
    gpio_direction_output(GPIO_BANK7 | GPIO_A4, 1);
```

具体的 bank 和 pin 在以下相关文件中定义：

```
    arch\arm\include\asm\arch-rk32xx\gpio.h
    arch\arm\include\asm\arch-rk32xx\gpio-rk3036.h
    arch\arm\include\asm\arch-rk32xx\gpio-rk312X.h
    arch\arm\include\asm\arch-rk32xx\gpio-rk3288.h

    arch\arm\include\asm\arch-rk33xx\gpio.h
    arch\arm\include\asm\arch-rk33xx\gpio-rk3368.h
```

接口函数如下：

```
    /**
     * Set gpio direction as input
     */
    int gpio_direction_input(unsigned gpio);
```

配置相应 GPIO 为输入口。

```
    /**
     * Set gpio direction as output
     */
    int gpio_direction_output(unsigned gpio, int value);
```

配置相应 GPIO 为输出口。

```
    /**
     * Get value of the specified gpio
     */
    int gpio_get_value(unsigned gpio)
```

获取 GPIO 相关 IO 的输入电平。

```
    /**
     * Set value of the specified gpio
     */
    int gpio_set_value(unsigned gpio, int value);
```

配置 GPIO 相关 IO 的输出电平。

```
    /**
     * Set gpio pull up or down mode
```

```
 */
int gpio_pull_updown(unsigned gpio, enum GPIOPullType type);
```
配置 GPIO 相关 IO 的上下拉。

```
/**
 * gpio drive strength slector
 */
int gpio_drive_slector(unsigned gpio, enum GPIODriveSlector slector);
```
配置 GPIO 相关 IO 的驱动能力。

4.4 IOMUX 驱动

Rockchip 平台 gpio 复用功能配置，驱动文件

```
arch\arm\include\asm\arch-rk32xx\iomux.h
arch\arm\cpu\armv7\rk32xx\iomux.c
arch\arm\cpu\armv7\rk32xx\iomux-rk3036.c
arch\arm\cpu\armv7\rk32xx\iomux-rk312X.c
arch\arm\cpu\armv7\rk32xx\iomux-rk3288.c

arch\arm\include\asm\arch-rk33xx\iomux.h
arch\arm\cpu\armv8\rk33xx\iomux.c
arch\arm\cpu\armv8\rk33xx\iomux-rk3368.c
void rk_iomux_config(int iomux_id);
```
ID 在 arch\arm\include\asm\arch-rk32xx\iomux.h 中定义

4.5 I2C 驱动

Rk I2C 支持标准 U-Boot 架构，详细可以参考 U-Boot 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

4.6 SPI 驱动

RK SPI 支持标准 U-Boot 架构，详细可以参考 U-Boot 文档，相关代码位于：

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

4.7 LCD 驱动

显示模块相关代码如下：

```
Drivers/video/rockchip_fb.c
Drivers/video/rockchip_fb.h
Drivers/video/rk32_lcdc.c
Drivers/video/rk3036_lcdc.c
Drivers/video/rk3368_lcdc.c
```

关于屏的电源控制：在 rockhip_fb.c 中，rk_fb_pwr_ctr_prase_dt 会解析 dts 中的所以电源控制接口，rk_fb_pwr_enable/disable 函数分别对显示模块的电源进行开关。如果屏的上电时序需要重新调整，可以修改该函数。

U-Boot 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，U-Boot 对 logo 的解析过程请参考 common/lcd.c 中的 rk_bitmap_from_resource（）函数。

开机 Logo 加载流程：

1) 从 rk_bitmap_from_resource 解析 resource 分区中的 logo；
2) 如果 resource 分区加载失败，会从 boot 分区中的 resource 中加载 logo；
3) U-Boot 的默认 logo 显示流程，所以即使 kernel 目录中不包含 logo.bmp 图片，U-Boot 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 U-Boot 的 rockchip.bmp。

4) 如果 resource 分区中存在 logo_kernel.bmp，那么 U-Boot 会加载该图片到 ddr 中并通过 commandline 通知内核加载的位置，内核显示新的 logo 图片。

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

只支持偶数分辨率的图片

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
```

5) 支持 24bit bmp 开机 logo 显示，支持内核更新一张 24bit logo 显示。

注意：对于 MID 相关的项目，不要在 U-Boot 里面打开 HDMI 相关的配置。

4.8 PMIC 驱动

如果要在 U-Boot 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 U-Boot 中已经自动兼容 Ricoh619、ACT8846、RK808 三款 PMU，对于 RK312X 项目，目前支持 ADC 电量检测和充电动画显示。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
drivers/power/pmic/pmic_rk818.c
drivers/power/fuel_gauge/fg_adc.c
```

其中 power_core.c 是 U-Boot 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。

4.9 电量计驱动

如果要在 U-Boot 中实现充电等功能，需要加入电量计（fuel_gauge）的支持。目前 RK U-Boot 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：

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 U-Boot 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和充电状态

```
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 rk32plat.h/rk30plat.h 中打开如下开关,默认该功能是关闭的

```
#define CONFIG_UBOOT_CHARGE
#define CONFIG_CMD_CHARGE_ANIM
#define CONFIG_CHARGE_DEEP_SLEEP            //采用 ricoch619 的 PMU 和 ADC 检测，不要打开这个选项
#define CONFIG_SCREEN_ON_VOL_THRESD         3550 //3.55v
#define CONFIG_SYSTEM_ON_VOL_THRESD         3650 //3.65v
```

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 U-Boot 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>;
};
```

4.10 存储驱动

U-Boot 支持 emmc、nand flash、sdcard 以及 U 盘等存储设备，其中 nand flash 驱动没有开放，接口通过 miniloader 传递给 U-Boot，所以如果需要支持 nand flash 设备需要用户自己定义 CONFIG_SECOND_LEVEL_BOOTLOADER，否则 U-Boot 只支持 emmc。

Rockchip 存储设备抽象在 storage 中，代码在 board/rockchip/common/storage/storage.c 中，相应的 API 接口主要有：

```
int32 StorageInit(void);
```

```
uint16 StorageGetBootMedia(void);

int StorageReadLba(uint32 LBA, void *pbuf, uint32 nSec);
int StorageWriteLba(uint32 LBA, void *pbuf, uint32 nSec, uint16 mode);

int StorageReadPba(uint32 PBA, void *pbuf, uint32 nSec);
int StorageWritePba(uint32 PBA, void *pbuf, uint32 nSec);
```

5 Google Fastboot

Google 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）。

5.1 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）

5.2 Fastboot 主要支持命令

5.2.1 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
```

5.2.2 镜像烧写

```
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）
```

5.3 重启

```
fastboot oem recovery                                    重启进 recovery
```

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

```
    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
```

5.3.1 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
```

5.3.2 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
```

5.4 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

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

6 固件加载

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

6.1 Boot/Recovery 分区

Boot 和 recovery 的固件分为两种形式：

1、 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。

2、 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。

6.2 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。

6.3 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   ..............
```

U-Boot 支持将 kernel 所需的 dtb 打包在 resource 分区。

6.4 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。U-Boot 假定 kernel 启动必须加载 dtb 文件。

6.5 固件加载流程

U-Boot 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。

Copyright 2016 @Fuzhou Rockchip Electronics Co., Ltd.

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

7 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 U-Boot 源码内：

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

7.1 支持 Loader 的打包和解包

7.1.1 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 U-Boot 源码内（tools/rk_tools/RKBOOT））如：

```
./tools/boot_merger ./tools/rk_tools/RKBOOT/RK3288.ini
out:RK3288Loader_UBOOT.bin
fix opt:RK3288Loader_UBOOT_V2.15.bin
merge success(RK3288Loader_UBOOT_V2.15.bin)
```

7.1.2 解包：

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

7.2 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_20
140404.bin
[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
_20140404.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
```

8 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 U-Boot 源码内（tools/resource_tool/）

8.1 支持 resource 镜像的打包和解包

8.1.1 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 successed!
```

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 successed!
```

```
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
```

8.2 解包

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

9 Trust_merger 工具

Trust_merger 是用于打包 bl30、bl31 bin、bl32 bin 等文件，生成烧写工具需要的 TrustImage 格式的 linux 版本工具。其源码位于 U-Boot 源码内：

```
UBOOT# ls ./tools/trust_merger.*
./tools/trust_merger.c    ./tools/trust_merger.h
```

支持 trust 的打包和解包

打包：

```
./tools/trust_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 U-Boot 源码内（tools/rk_tools/RKTRUST））如：

```
./tools/trust_merger ./tools/rk_tools/RKTRUST/RK3368.ini
out:trust.img
merge success(trust.img)
```

9.1 解包：

```
./tools/trust_merger --unpack <trust.img>
```

9.2 参数配置文件

以 3368 的配置文件为例：

```
[VERSION]
MAJOR=0                    ----主版本号
MINOR=1                    ----次版本号
[BL30_OPTION]                  ----bl30，目前设置为 mcu bin
SEC=1                      ----存在 BL30 bin
PATH=tools/rk_tools/bin/rk33/rk3368bl30_v2.00.bin    ----指定 bin 路径
ADDR=0xff8c0000            ----固件 DDR 中的加载和运行地址
[BL31_OPTION]                  ----bl31，目前设置为多核和电源管理相关的 bin
SEC=1                      ----存在 BL31 bin
PATH=tools/rk_tools/bin/rk33/rk3368bl31-20150401-v0.1.bin----指定 bin 路径
ADDR=0x00008000                ----固件 DDR 中的加载和运行地址
[BL32_OPTION]
SEC=0                      ----不存在 BL31 bin
[BL33_OPTION]
SEC=0                      ----不存在 BL31 bin
[OUTPUT]
PATH=trust.img [OUTPUT]  ----输出固件名字
```

10 SDCard 和U 盘启动升级

Rockchip 设备支持 RK 制作工具定制的 sdcard、u 盘等的启动和升级功能。

10.1 SDCard 启动和升级配置

SDCard 升级和启动功能由宏 CONFIG_RK_SDCARD_BOOT_EN 配置，rk_default_config.h 默认关闭，如果需要使用该功能，请在 rkxxplat.h 相关平台配置文件中定义打开。

UBoot 识别到升级的 sdcard，串口会打印信息：

```
SDCard Update.
```

如果是启动的 sdcard，则会打印信息：

```
SDCard Boot.
```

10.2 U 盘启动和升级配置

U 盘升级和启动功能由宏 CONFIG_RK_UMS_BOOT_EN 配置，rk_default_config.h 默认关闭，如果需要使用该功能，请在 rkxxplat.h 相关平台配置文件中定义打开。

UBoot 识别到升级的 U 盘，串口会打印信息：

```
UMS Update.
```

如果是启动的 sdcard，则会打印信息：

```
UMS Boot.
```

10.2.1 功能配置

UMS 相关的宏定义在相应的 rkxxplat.h 中

```
CONFIG_RK_UMS_BOOT_EN
```

配置 U-Boot 是否支持 U 盘启动和升级功能，define 打开 undef 关闭。

```
RKUSB_UMS_BOOT_FROM_DWC2_OTG
RKUSB_UMS_BOOT_FROM_DWC2_HOST
RKUSB_UMS_BOOT_FROM_EHCI_HOST1
RKUSB_UMS_BOOT_FROM_EHCI_HOST2
RKUSB_UMS_BOOT_FROM_EHCI_HOST3
```

五选一，由于 U-Boot USB 框架的限制，只能开启一个 USB 控制器打开 U 盘启动功能。

这里以 rk32plat.h 相关的代码为例：

```
#ifdef CONFIG_RK_UMS_BOOT_EN
/*
 * USB Host support, default no using
 * Please first select USB host controller if you want to use UMS Boot
 * Up to one USB host controller could be selected to enable for booting
 * from USB Mass Storage device.
 *
 * PLS define a host controler from:
 *  RKUSB_UMS_BOOT_FROM_DWC2_OTG
 *  RKUSB_UMS_BOOT_FROM_EHCI_HOST1
 *  RKUSB_UMS_BOOT_FROM_DWC2_HOST
 *
```

```
     * First define the host controller here
     */


    /* Check UMS Boot Host define */
    #define RKUSB_UMS_BOOT_CNT
(defined(RKUSB_UMS_BOOT_FROM_DWC2_OTG) + \
                    defined(RKUSB_UMS_BOOT_FROM_EHCI_HOST1) + \
                    defined(RKUSB_UMS_BOOT_FROM_DWC2_HOST))

    #if (RKUSB_UMS_BOOT_CNT == 0)
        #error "PLS Select a USB host controller!"
    #elif (RKUSB_UMS_BOOT_CNT > 1)
        #error "Only one USB host controller can be selected!"
    #else
        #define CONFIG_CMD_USB
        #define CONFIG_USB_STORAGE
        #define CONFIG_PARTITIONS
    #endif


    /*
     * USB Host support, default no using
     * please first check plat if you want to using usb host
     */
    #if defined(RKUSB_UMS_BOOT_FROM_EHCI_HOST1)
        #define CONFIG_USB_EHCI
        #define CONFIG_USB_EHCI_RK
    #elif defined(RKUSB_UMS_BOOT_FROM_DWC2_HOST) ||
defined(RKUSB_UMS_BOOT_FROM_DWC2_OTG)
        #define CONFIG_USB_DWC_HCD
    #endif
    #endif /* CONFIG_RK_UMS_BOOT_EN */
```

当使能 CONFIG_RK_UMS_BOOT_EN 后，必须定义相应的 HOST 端口（根据具体的芯片定义），没有定义或者定义过多，编译的时候都会报错提示。

10.2.2 控制器配置表

board\rockchip\common\mediaboot\UMSBoot.c 根据具体使用的芯片平台和控制器填写 rkusb_hcd 配置表，USB Mass_storage 会根据配置来进行初始化

```
    struct rkusb_hcd_cfg {
        bool enable;   //预留给扩展功能使用，默认值为 True 不需要修改
        void* regbase;   //寄存器基地址，由 UBOOT 维护人员填写
        int gpio_vbus;   //USB 供电控制 GPIO，由用户根据电路配置填写
        char *name;    //控制器名称，由 UBOOT 维护人员填写
        void (*hw_init)(void);    //初始化回调函数，由 UBOOT 维护人员填写
        void (*hw_deinit)(void);   //反初始化回调函数，由 UBOOT 维护人员填写
```

```
    };
```