

# SPL、MTD开发指南

---

文档标识: RK-KF-YF-314

发布版本: V1.1.0

日期: 2020-07-10

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

## 免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司(“本公司”, 下同)不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自所有者所有。

版权所有 © 2019 福州瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址: 福建省福州市铜盘路软件园A区18号

网址: [www.rock-chips.com](http://www.rock-chips.com)

客户服务电话: +86-4007-700-590

客户服务传真: +86-591-83951833

客户服务邮箱: [fae@rock-chips.com](mailto:fae@rock-chips.com)

## 前言

## 概述

Rockchip SDK默认采用闭源的miniloader 加载 trust 和 u-boot，所有存储设备（eMMC NAND/NOR Flash）都以block接口访问，对于想通过MTD 接口访问NAND / NOR Flash的开发者，Rockchip 提供了开源的SPL来加载trust和u-boot，并且在 u-boot中通过MTD接口访问NAND/NOR Fash 。

## 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

## 产品版本

芯片名称	内核版本
RK3308	4.4

## 修订记录

日期	版本	作者	修改说明
2019-06-20	V1.0.0	HKH	初始版本
2019-11-11	V1.0.1	HKH	增加sd卡升级说明
2020-07-08	V1.0.2	Ruby Zhang	格式修订
2020-07-10	V1.1.0	Jair Wu	新增u-boot编译说明

## 目录

### SPL、MTD开发指南

1. 编译配置
  - 1.1 U-Boot
    - 1.1.1 配置说明
    - 1.1.2 编译说明
  - 1.2 Kernel
  - 1.3 Buildroot
  - 1.4 编译脚本
  - 1.5 分区表
  - 1.6 SD 引导升级
2. 烧写说明
  - 2.1 工具
  - 2.2 启动关键Log
3. UBIFS说明
  - 3.1 UBIFS 简介
  - 3.2 UBI Layer
  - 3.3 UBIFS 应用样例
    - 3.3.1 挂载一个空的UBIFS 文件系统
    - 3.3.2 制作 UBIFS 根文件系统 UBI 镜像
    - 3.3.3 通过BuildRoot 编译 UBIFS 根文件系统 UBI 镜像
4. 附录参考

# 1. 编译配置

## 1.1 U-Boot

### 1.1.1 配置说明

defconfig 配置如下：

增加

```
CONFIG_CMD_NAND=y
CONFIG_CMD_SF=y
CONFIG_CMD_SPI=y
CONFIG_NAND_ROCKCHIP_DT=y
CONFIG_CMD_MTDPARTS=y
CONFIG_NAND=y
CONFIG_NAND_ROCKCHIP=y
CONFIG_MTD=y
CONFIG_MTD_SPI_NAND=y
CONFIG_CMD_MTD_BLK=y
CONFIG_SPL_MTD_SUPPORT=y
CONFIG_MTD_DEVICE=y
CONFIG_CMD_MTD=y
CONFIG_MTD_BLK=y
CONFIG_SPL_SPI_FLASH_SUPPORT=y
CONFIG_SPL_SPI_SUPPORT=y
CONFIG_SPL_LOAD_RKFW=y
CONFIG_SPL_NAND_SUPPORT=y
CONFIG_SPL_SYS_MALLOC_F_LEN=0x100000
CONFIG_SPI_FLASH=y
CONFIG_SF_DEFAULT_MODE=0x1
CONFIG_SF_DEFAULT_SPEED=50000000
CONFIG_SPI_FLASH_EON=y
CONFIG_SPI_FLASH_GIGADEVICE=y
CONFIG_SPI_FLASH_MACRONIX=y
CONFIG_SPI_FLASH_WINBOND=y
CONFIG_SPI_FLASH_MTD=y
CONFIG_ROCKCHIP_SFC=y
CONFIG_SYS_NAND_U_BOOT_LOCATIONS=y
CONFIG_SYS_NAND_U_BOOT_OFFS=0x8000
CONFIG_SYS_NAND_U_BOOT_OFFS_REDUND=0x10000
CONFIG_RKFW_TRUST_SECTOR=0X3000      #存储设备里面的烧写地址,以sector为单位的,1
sector=512 Bytes,即paramter.txt 里trust的起始地址
CONFIG_RKFW_U_BOOT_SECTOR=0X2000    #存储设备里面的烧写地址,以sector为单位的,1
sector=512 Bytes,即paramter.txt 里u-boot的起始地址
```

删除

```
CONFIG_RKFLASH=y
CONFIG_RKNANDC_NAND=y
CONFIG_RKSFC_NAND=y
CONFIG_RKSFC_NOR=y
```

### 1.1.2 编译说明

64位系统:

在SDK根目录使用 `./build.sh uboot` 或进入u-boot目录下使用 `./make.sh rk3308` 即可。

32位系统:

编译32位系统时，仍需使用64位的Loader，因此需要先编译64位的u-boot，备份spl/u-boot-spl.bin，再编译32位的u-boot，打包前将spl/u-boot-spl.bin替换，具体操作如下：

```
cd u-boot
./make.sh rk3308
cp spl/u-boot-spl.bin ..
./make.sh rk3308-aarch32
cp ../u-boot-spl.bin spl/
./make.sh spl-s ../rkbin/RKBOOT/RK3308MINIALL_WO_FTL.ini
```

该步骤生成的rk3308\_loader\_wo\_ftl\_v\*.bin即可作为最终的Loader烧录至板上。

## 1.2 Kernel

以RK3308 EVB\_V13开发板为例，bootargs 配置启动ubifs的rootfs，DTS 修改如下：

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3308-evb-v13.dtsi
b/arch/arm64/boot/dts/rockchip/rk3308-evb-v13.dtsi
index 92675be..62b80e2 100644
--- a/arch/arm64/boot/dts/rockchip/rk3308-evb-v13.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3308-evb-v13.dtsi
@@ -12,7 +12,8 @@
     compatible = "rockchip,rk3308-evb-v13", "rockchip,rk3308";

     chosen {
-        bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
+        bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
 console=ttyFIQ0 root=PARTUUID=614e0000-0000 rootfstype=squashfs rootwait
 snd_aloop.index=7";
+        /*bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
 console=ttyFIQ0 root=PARTUUID=614e0000-0000 rootfstype=squashfs rootwait
 snd_aloop.index=7";*/
+        bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1
 console=ttyFIQ0 ubi.mtd=5 root=ubi0:rootfs rootfstype=ubifs rootwait
 snd_aloop.index=7";
     };

     adc-keys {
```

前期开发调试，可将rootfs 挂载为可读写，在bootargs 中加入rw 标记即可。如下：

```
bootargs = "earlycon=uart8250,mmio32,0xff0c0000 swiotlb=1 console=ttyFIQ0
ubi.mtd=5 root=ubi0:rootfs rootfstype=ubifs rw rootwait snd_aloop.index=7";
```

defconfig 配置如下:

增加

```
CONFIG_MTD=y
CONFIG_MTD_CMDLINE_PARTS=y
CONFIG_MTD_NAND=y
CONFIG_MTD_NAND_ROCKCHIP_V6=y
CONFIG_MTD_UBI=y
CONFIG_MTD_UBI_WL_THRESHOLD=1024
CONFIG_UBIFS_FS=y
CONFIG_UBIFS_FS_ADVANCED_COMPR=y
# CONFIG_UBIFS_FS_ZLIB is not set
CONFIG_CRYPTODEFLATE=y
```

删除

```
CONFIG_RK_FLASH=y
CONFIG_RK_NANDC_NAND=y
CONFIG_RK_SFC_NAND=y
CONFIG_RK_SFC_NOR=y
```

## 1.3 Buildroot

rootfs 以ubifs 为例, 参照下图配置, 具体配置参数详见ubifs

```
+-----+
| ^(-) |
| [ ]  tar the root filesystem |
| [*]  ubi image containing an ubifs root filesystem |
|      (0x20000) physical eraseblock size PEB:物理逻辑块大小 |
|      (2048) sub-page size 页大小 |
| [ ]  Use custom config file |
| (-v) Additional ubinize options 打印编译信息 |
| -*  ubifs root filesystem |
|      (0x1f000) logical eraseblock size LEB:逻辑擦除块大小 |
|      (0x800) minimum I/O unit size 页大小 |
|      (488) maximum logical eraseblock count 逻辑擦除块数 |
|      ubifs runtime compression (lzo) ---> |
|      Compression method (no compression) ---> |
| (-F -v) Additional mkfs.ubifs options -F可烧写, -v打印编译信息 |
| [ ]  yaffs2 root filesystem |
+-----+
```

配置完成后, 使用make savedefconfig 保存Buildroot 的配置。

## 1.4 编译脚本

build.sh 增加spl 编译, 生成的 spl 文件位于u-boot/spl/u-boot-spl.bin, 如下编译脚本会自动将spl打包到 MiniloaderAll.bin 文件中。编译32位u-boot时由于存在替换操作, 因此不建议使用该脚本, 建议参考[1.1.2 编译说明](#) 手动生成。

```
diff --git a/common/build.sh b/common/build.sh
index 671decd..c4fe085 100755
--- a/common/build.sh
+++ b/common/build.sh
@@ -46,7 +46,7 @@ function build_uboot(){
    if [ -f u-boot/*_loader_.bin ]; then
        rm u-boot/*_loader_.bin
    fi
-    cd u-boot && ./make.sh $RK_UBOOT_DEFCONFIG && cd -
+    cd u-boot && ./make.sh $RK_UBOOT_DEFCONFIG && ./make.sh spl-s
    ../rkbin/RKBOOT/RK3308MINIALL_WO_FTL.ini && cd -
    if [ $? -eq 0 ]; then
        echo "====Build uboot ok!===="
    else
```

BoardConfig.mk 修改以下字段:

```
export RK_ROOTFS_TYPE=ubi
export RK_OEM_FS_TYPE=ubifs
export RK_USERDATA_FS_TYPE=ubifs
```

打包工具修改（tools 目录），不打包oem 和 userdata 如下:

```
diff --git a/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
b/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
index 92c0259..260e2fe 100755
--- a/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
+++ b/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
@@ -9,8 +9,8 @@ uboot      Image/uboot.img
boot        Image/boot.img
rootfs      Image/rootfs.img
recovery    Image/recovery.img
-oem                Image/oem.img
-userdata:grow      Image/userdata.img
+#oem                Image/oem.img
+#userdata:grow      Image/userdata.img
```

## 1.5 分区表

分区表要使用GPT表，即parameter.txt 文件中，配置如下字段

```
TYPE: GPT
```

## 1.6 SD 引导升级

SPL 方案支持sd卡升级方案，如需该功能，需要打开以下配置:

u-boot目录:

```
diff --git a/arch/arm/dts/rk3308-evb.dts b/arch/arm/dts/rk3308-evb.dts
index 3178d45..68853d6 100644
--- a/arch/arm/dts/rk3308-evb.dts
+++ b/arch/arm/dts/rk3308-evb.dts
@@ -330,7 +330,7 @@
        sd-uhs-sdr25;
        sd-uhs-sdr50;
        sd-uhs-sdr104;
-       status = "disabled";
+       status = "okay";
    };

    &u2phy {
```

```
-CONFIG_OF_SPL_REMOVE_PROPS="pinctrl-names clock-names interrupt-
parent assigned-clocks assigned-clock-rates assigned-clock-parents"
+CONFIG_OF_SPL_REMOVE_PROPS=""
+CONFIG_SPL_PINCTRL_GENERIC=y
+CONFIG_SPL_PINCTRL=y
```

kernel目录:

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3308.dtsi
b/arch/arm64/boot/dts/rockchip/rk3308.dtsi
index 8a98886..970fb69 100644
--- a/arch/arm64/boot/dts/rockchip/rk3308.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3308.dtsi
@@ -1166,6 +1166,8 @@
        nandc_id = <0>;
        clocks = <&cru SCLK_NANDC>, <&cru HCLK_NANDC>;
        clock-names = "clk_nandc", "hclk_nandc";
+       pinctrl-names = "default";
+       pinctrl-0 = <&flash_csn0 &flash_rdy &flash_ale &flash_cle
&flash_wrn &flash_rdn &flash_bus8>;
        status = "disabled";
    };
```

```
diff --git a/drivers/mtd/nand/rockchip_nand_v6.c
b/drivers/mtd/nand/rockchip_nand_v6.c
index 5a74427..31208ba 100644
--- a/drivers/mtd/nand/rockchip_nand_v6.c
+++ b/drivers/mtd/nand/rockchip_nand_v6.c
@@ -20,6 +20,7 @@
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/iopoll.h>
+#include <asm/io.h>

#define NANDC_V6_NUM_BANKS 4
#define NANDC_V6_DEF_TIMEOUT 20000
@@ -689,6 +690,7 @@ static int rk_nandc_probe(struct platform_device *pdev)
    int irq;
    int ret;
    int clock_frequency;
+    void __iomem *base;
```



```

        nandc = devm_kzalloc(dev, sizeof(*nandc), GFP_KERNEL);
        if (!nandc)
@@ -697,6 +699,8 @@ static int rk_nandc_probe(struct platform_device *pdev)
        nandc->dev = dev;

        r = platform_get_resource(pdev, IORESOURCE_MEM, 0);
+       base = ioremap(0xff000000, 0x10000);
+       printk("%s %x %x\n", __func__, readl(base + 0x60), readl(base + 0x68));
        nandc->regs = devm_ioremap_resource(dev, r);
        if (IS_ERR(nandc->regs))
            return PTR_ERR(nandc->regs);

```

recovery 的defconfig 需要打开宏

```
BR2_PACKAGE_MTD=y
```

编译完成后使用平台工具SDDiskTool\_v1.59进行制卡，第一次升级需要在sd卡的根目录生成升级Flash需要的固件，所以时间会比较长，后面效率则会加快。

## 2. 烧写说明

### 2.1 工具

AndroidTools 烧写工具支持UBI烧写，识别到固件为UBI，则先格式化分区，再烧写该分区，工具版本必须在V2.6.9或者之上。

### 2.2 启动关键Log

SPL Log

```

U-Boot SPL board init
U-Boot SPL 2017.09-03071-g9cb6379-dirty (Jun 28 2019 - 10:29:22)

```

启动成功之后，运行mount 命令，会有如下挂载：

```

# mount
ubi0:rootfs on / type ubifs (rw,relatime)
/dev/ubi6_0 on /oem type ubifs (rw,relatime)
/dev/ubi7_0 on /userdata type ubifs (rw,relatime)

```

## 3. UBIFS说明

### 3.1 UBIFS 简介

无序区块镜像文件系统(Unsorted Block Image File System, UBIFS)是用于固态存储设备上，并与LogFS相互竞争，作为JFFS2的后继文件系统之一。

### 3.2 UBI Layer

UBIFS 涉及三个子系统：

1. MTD系统，提供对各种Flash芯片的访问接口：drivers/mtd
2. UBI系统，工作在MTD上，提供UBI volume：drivers/mtd/ubi
3. UBIFS文件系统，工作在UBI之上，fs/ubifs

### 3.3 UBIFS 应用样例

#### 3.3.1 挂载一个空的UBIFS 文件系统

当前板子有6个分区，分区情况如下

```
# cat /proc/mtd
dev:      size  erasesize  name
mtd0: 00200000 00020000 "uboot"
mtd1: 00200000 00020000 "trust"
mtd2: 00100000 00020000 "misc"
mtd3: 00c00000 00020000 "recovery"
mtd4: 00900000 00020000 "boot"
mtd5: 04400000 00020000 "rootfs"
mtd6: 09e00000 00020000 "userdata"
```

#### 1. 格式化UBI分区

使用以下命令格式化UBI分区

```
# ubiformat /dev/mtd6
ubiformat: mtd6 (nand), size 165675008 bytes (158.0 MiB), 1264 eraseblocks of
131072 bytes (128.0 KiB), min. I/O size 2048 bytes
libscan: scanning eraseblock 1263 -- 100 % complete
ubiformat: 1260 eraseblocks are supposedly empty
ubiformat: 4 bad eraseblocks found, numbers: 1260, 1261, 1262, 1263
ubiformat: formatting eraseblock 462 -- 36 % complete   1 Jan 08:00:21
ntpd[377]: Listen normally on 4 wlan0 169.254.47.142:123
ubiformat: formatting eraseblock 1263 -- 100 % complete
```

#### 2. 绑定UBI到MTD分区

绑定UBI到MTD6分区，使用以下命令

```
# ubiattach /dev/ubi_ctrl -m 6
[ 91.687268] ubi1: attaching mtd6
[ 92.117692] ubi1: scanning is finished
[ 92.126771] ubi1: attached mtd6 (name "userdata", size 158 MiB)
[ 92.126838] ubi1: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
[ 92.126864] ubi1: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
[ 92.126888] ubi1: VID header offset: 2048 (aligned 2048), data offset:
4096
[ 92.126912] ubi1: good PEBs: 1260, bad PEBs: 4, corrupted PEBs: 0
[ 92.126935] ubi1: user volume: 0, internal volumes: UBI device number 1,
total 12601 LEBs (159989760 bytes, 152.6 MiB), available 1220 LEBs (15491,0720
bytes, 147.7 MiB), LEB size 126976 bytes (124.0 KiB)
max. volumes count: 128
[ 92.126973] ubi1: max/mean erase counter: 0/0, WL threshold: 1024, image
seq# uence number: 864764485
[ 92.127000] ubi1: available PEBs: 1220, total reserved PEBs: 40, PEBs
reserved for bad PEB handling: 36
[ 92.127128] ubi1: background thread "ubi_bgt1d" started, PID 465
```

参数-m 6 表示使用MTD6 分区。只有绑定了UBI 到 JMTD分区以后，才能在/dev/下找到ubi 设备 "ubi1"，如果曾经创建过UBI卷，那么绑定以后才能在/dev/下找到并访问ubi卷“ubi1\_0”

查看所有设备 ls /dev/ubi\*，将发现多了一个设备“dev/ubi1”

### 3. 创建UBI卷

UBI 卷可以理解为UBI 设备的分区，创建UBI卷命令如下：

```
# ubimkvol /dev/ubi1 -N ubifs -s 147.7MiB
Volume ID 0, size 1157 LEBs (146911232 bytes, 140.1 MiB), LEB size 126976
bytes (124.0 KiB), dynamic, name "ubifs", alignment 1
```

参数 "/dev/ubi0" 是上一步骤创建的UBI设备

参数 "-N ubifs" 表示创建的卷名为"ubifs"

参数 "-s SIZE" 表示创建的分区大小

#### 说明

SIZE值应小于"/dev/ubi1" 设备能提供的空间大小。

可以使用命令 "ubinfs" 查看当前可使用的LEBs大小。如下所示，当前UBI设备提供的空间为152.6MiB时，可使用的空间大小为147.7MiB。所以，应该保证所创建的卷的SIZE值小于可使用的LEBs空间大小。

```
# ubinfo /dev/ubi1
ubi1
Volumes count:                0
Logical eraseblock size:      126976 bytes, 124.0 KiB
Total amount of logical eraseblocks: 1260 (159989760 bytes, 152.6 MiB)
Amount of available logical eraseblocks: 1220 (154910720 bytes, 147.7 MiB)
Maximum count of volumes      128
Count of bad physical eraseblocks: 4
Count of reserved physical eraseblocks: 36
Current maximum erase counter value: 3
Minimum input/output unit size: 2048 bytes
Character device major/minor: 249:0
```

卷是需要创建一次，创建完成，卷的信息将被保存在UBI设备上，下一次启动，不需要再创建卷。"ubirmvol"，用于删除卷，使用该命令删除卷，卷上所有的信息将被删除。

#### 4. 挂载UBIFS 文件系统

此时就可以将创建的卷挂载到指定的目录去了，命令如下：

```
mount -t ubifs /dev/ubi1_0 /mnt/
```

或者

```
mount -t ubifs ubi1:ubifs /mnt
```

挂载成功信息将显示如下：

```
# mount -t ubifs /dev/ubi1_0 /userdata/
[ 503.693331] UBIFS (ubi1:0): default file-system created
[ 503.694376] UBIFS (ubi1:0): background thread "ubifs_bgt1_0" started, PID 477
[ 503.730377] UBIFS (ubi1:0): UBIFS: mounted UBI device 1, volume 0, name "ubifs"
[ 503.730454] UBIFS (ubi1:0): LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes: 2048 bytes/2048 bytes
[ 503.730489] UBIFS (ubi1:0): FS size: 152752128 bytes (145 MiB, 1203 LEBs), journal size 7618560 bytes (7 MiB, 60 LEBs)
[ 503.730582] UBIFS (ubi1:0): reserved for root: 495# 2683 bytes (4836 KiB)
[ 503.730619] UBIFS (ubi1:0): media format: w4/r0 (latest is w4/r0), UUID 054CEFD8-A535-4680-A69F-EFC0352731EB, small LPT model
```

查看分区信息，将显示如下内容：

```
# df
Filesystem            1K-blocks      Used Available Use% Mounted on
ubi0:rootfs            54128        54128          0 100% /
devtmpfs               122976          0    122976   0% /dev
tmpfs                  123136          0    123136   0% /dev/shm
tmpfs                  123136         68    123068   0% /tmp
tmpfs                  123136         96    123040   0% /run
/dev/ubi1_0            137276         24    132416   0% /userdata
```

UBIFS 文件系统显示的分区大小，剩余空间并不准确。因为UBIFS文件保存的是文件压缩后的内容，压缩比率与文件内容相关。可能剩余空间显示只有2M，但是可以将一个4M的文件完整保存。

### 3.3.2 制作 UBIFS 根文件系统 UBI 镜像

制作镜像文件

制作UBIFS文件系统镜像，需要使用mtd-ubifs工具，命令如下：

```
mkfs.ubifs -F -d rootfs_dir -e 126976 -c 240 -m 0x800 -v -o rootfs.ubifs
```

参数 -F 使能 “white-space-fixup”，如果是通过u-boot或者烧写工具烧写需要使能此功能

参数 -d rootfs\_dir 表示将要被制作作为UBIFS 镜像的根目录为 rootfs，这个参数也可以写成 -r rootfs\_dir

参数 -m 0x800 表示最小读写单元是2KiB。这里使用的NAND芯片页大小为2KiB。最小读写单元是指FLASH器件

一次读写操作，最小操作的字节数，对NAND 器件，是页大小，对NOR 器件，是1个字节。

参数 -o rootfs.ubifs 表示制作出来的镜像名称为 rootfs.ubifs

参数 -e 126976 表示逻辑擦除块大小

最小读写单元和逻辑块大小可以通过读MTD 和UBI 系统信息获得，也可以通过计算获得。

读MTD 信息命令以及显示内容如下：

```
# mtdinfo /dev/mtd6
mtd6
Name:                userdata
Type:                nand
Eraseblock size:     131072 bytes, 128.0 KiB
Amount of eraseblocks: 1264 (165675008 bytes, 158.0 MiB)
Minimum input/output unit size: 2048 bytes
Sub-page size:       2048 bytes
OOB size:            64 bytes
Character device major/minor: 90:12
Bad blocks are allowed: true
Device is writable:  true
```

读取UBI 信息命令

```
# ubinfo /dev/ubi1
ubi1
Volumes count:                0
Logical eraseblock size:      126976 bytes, 124.0 KiB
Total amount of logical eraseblocks: 1260 (159989760 bytes, 152.6 MiB)
Amount of available logical eraseblocks: 1220 (154910720 bytes, 147.7 MiB)
Maximum count of volumes      128
Count of bad physical eraseblocks: 4
Count of reserved physical eraseblocks: 36
Current maximum erase counter value: 7
Minimum input/output unit size: 2048 bytes
Character device major/minor: 249:0
```

参数 `-c 240` 表示此文件系统最多使用240个逻辑擦除块。计算"240xLEB"得到此文件系统的最大可使用空间。LEBs=此文件系统的最大可使用空间/LEB。

参数 `-v` 显示制作UBIFS 过程中的详细信息。

逻辑擦除块大小可以通过计算得到，计算方法如下表：

FLASH	逻辑擦除块大小
NOR	$LEB = \text{blocksize} - 128$
NAND 无子页	$LEB = \text{blocksize} - \text{pagesize} * 2$
NAND 有子页	$LEB = \text{blocksize} - \text{pagesize} * 1$
blocksize	flash 物理擦除块大小
pagesize	flash 读写页大小

注意，制作成功的UBIFS根文件系统镜像为UBI 镜像，可以在内核下对空UBIFS文件系统进行升级操作，该镜像不能直接烧录到MTD分区上使用，但是可以通过格式转换，转换成能直接烧录MTD分区上个格式。

转换为可直接烧写MTD 的固件

制作UBI 镜像转换配置文件 `ubi.cfg`，如下

```
[ubifs-volumn]
mode=ubi
image=out/rootfs.ubifs
vol_id=0
vol_type=dynamic
vol_alignment=1
vol_name=ubifs
vol_flags=autoresize
```

参数`mode=ubi`，是强制参数，当前不能输入别的值，保留为以后扩展功能

参数`image=out/rootfs.ubifs`, 此文件为源文件

参数`vol_id=0`，表示卷的ID，UBI镜像可能包含多个卷，这个用来区别不同的卷。

参数`vol_type=dynamic`，表示当前卷类型是可读写的。只读为`static`

参数`vol_name=ubifs`，卷的名称

参数`vol_flags=autosize`，表示卷的大小是可扩展的

转换UBI格式

```
#ubinize -o out/rootfs.img -m 2KiB -p 128KiB ubi.cfg -v
```

参数 `-o out/rootfs.img` 表示输出文件

参数 `-m 2KiB`，最小读写单元为2KiB

参数 `-p 128KiB`，flash物理擦除大小，不是逻辑擦除块大小

`ubi.cfg` 配置文件

参数 `-v`，显示制作过程的详细信息

### 3.3.3 通过BuildRoot 编译 UBIFS 根文件系统 UBI 镜像

Filesystem images ---> 选择如下图，其中488 根据大小来配置

```
+-----+
| ^(-)                                     |
| [ ] tar the root filesystem              |
| [*] ubi image containing an ubifs root  |
|      filesystem                         |
| (0x20000) physical eraseblock size      |
| (2048) sub-page size                    |
| [ ] Use custom config file              |
| (-v) Additional ubinize options         |
| -* ubifs root filesystem                |
| (0x1f000) logical eraseblock size        |
| (0x800) minimum I/O unit size           |
| (488) maximum logical eraseblock count  |
|      ubifs runtime compression (lzo)   --->|
|      Compression method (no compression) --->|
| (-F -v) Additional mkfs.ubifs options |
| [ ] yaffs2 root filesystem              |
+-----+
```

## 4. 附录参考

[1] UBI FAQ: <http://www.linux-mtd.infradead.org/faq/ubi.html>

[2] UBIFS FAQ: [http://www.linux-mtd.infradead.org/faq/ubifs.html#L\\_lebsz\\_mismatch](http://www.linux-mtd.infradead.org/faq/ubifs.html#L_lebsz_mismatch)

[3] MTD FAQ: <http://www.linux-mtd.infradead.org/faq/general.html>