# SPL、MTD开发指南

发布版本：1.0.0

作者邮箱：jkand.huang@rock-chips.com

日期：2019.06

文件密级：公开资料

---

**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 | 黄开辉 | 初始版本 |
| 2019-10-20 | V1.0.1 | 黄开辉 | 新增SPI-NADN说明 |

---

# 1、编译配置

## 1.1、u-boot

defconfig 配置如下：

参考<Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf> 第8章。

删除

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

## 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
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";
```

spi-nand机器，dts 确认sfc 节点是否有打开，如下

```
&sfc {
    status = "okay";
};
```

defconfig 配置如下：

增加

```
#spi-nand 设备
CONFIG_MTD=y
CONFIG_MTD_CMDLINE_PARTS=y
CONFIG_MTD_UBI=y
CONFIG_RK_FLASH=y
CONFIG_RK_SFC_NAND=y
CONFIG_RK_SFC_NAND_MTD=y
CONFIG_UBIFS_FS=y
CONFIG_UBIFS_FS_ADVANCED_COMPR=y

#nand 设备
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_CRYPTO_DEFLATE=y
```

nand 设备需要删除

```
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章节说明

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。

```
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
```

## 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、参考固件

百度云地址:

## 2.3、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、mount 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 "ubinfo" 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
```

mount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
054CEFDB-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=blocksize-128 |
| NAND 无子页 | LEB=blocksize - pagesize*2 |
| NAND 有子页 | LEB=blocksize - pagesize*1 |
| blocksize | flash 物理擦除块大小 |
| | 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

# 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

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