SPL and MTD Developer Guide

Document ID: RK-KF-YF-314

Release version: 1.0.1

E-mail: jkand.huang@rock-chips.com

Date: 2019.11

Security Class: Publicity

Fuzhou Rockchip Electronics Co., Ltd.

Address: No.18 Building, A District, No.89 Software Boulevard, FuZhou, FuJian, PRC

Website: www.rock-chips.com

Customer service Tel.： +86-4007-700-590

Customer service Fax： +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

Rockchip SDKs use closed-source miniloader to load trust and u-boot by default. All memories (eMMC NAND or NOR Flash) are accessed through block interface. For developers who want to access NAND or NOR Flash through MTD interface, Rockchip provides open source SPL to load trust and u-boot, and access NAND or NOR Fash through MTD interface in u-boot.

Intended Audience

This document is mainly suitable for below engineers:

Field Application Engineer

Software Development Engineer

Product Version

Chipset	Kernel version
RK3308	4.4

Revision History

Date	Version	Author	Revision History
2019-06-20	V1.0.0	HKH	Initial version
2019-11-11	V1.0.1	HKH	Add SD card upgrade introduction

Contents

1. Build Configuration Changes

1.1. U-Boot

The defconfig is configured as follows：

Add:

```
CONFIG_CMD_NAND=y
CONFIG_NAND_ROCKCHIP_DT=y
CONFIG_CMD_MTDPARTS=y
CONFIG_NAND=y
CONFIG_NAND_ROCKCHIP=y
CONFIG_MTD=y
CONFIG_MTD_DEVICE=y
CONFIG_CMD_MTD=y
CONFIG_MTD_BLK=y
CONFIG_SPL_LOAD_RKFW=y
CONFIG_SPL_NAND_SUPPORT=y
CONFIG_SPL_SYS_MALLOC_F_LEN=0x100000
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     #The flashing address in memory is in sectors, 1 sector=512 Bytes, which is the start address of the trust in paramter.txt
CONFIG_RKFW_U_BOOT_SECTOR=0X2000    #The flashing address in memory is in sectors, 1 sector=512 Bytes, which is the starting address of u-boot in paramter.txt
```

Remove:

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

1.2. Kernel

Taking RK3308 EVB_V13 development board as an example, bootargs is configured to start the rootfs of ubifs, and DTS is modified as follows:

```
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 {
```

For early development and debugging, you can mount rootfs as read-write, and add rw flag to bootargs as follows:

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

The defconfig is configured as follows:

Add:

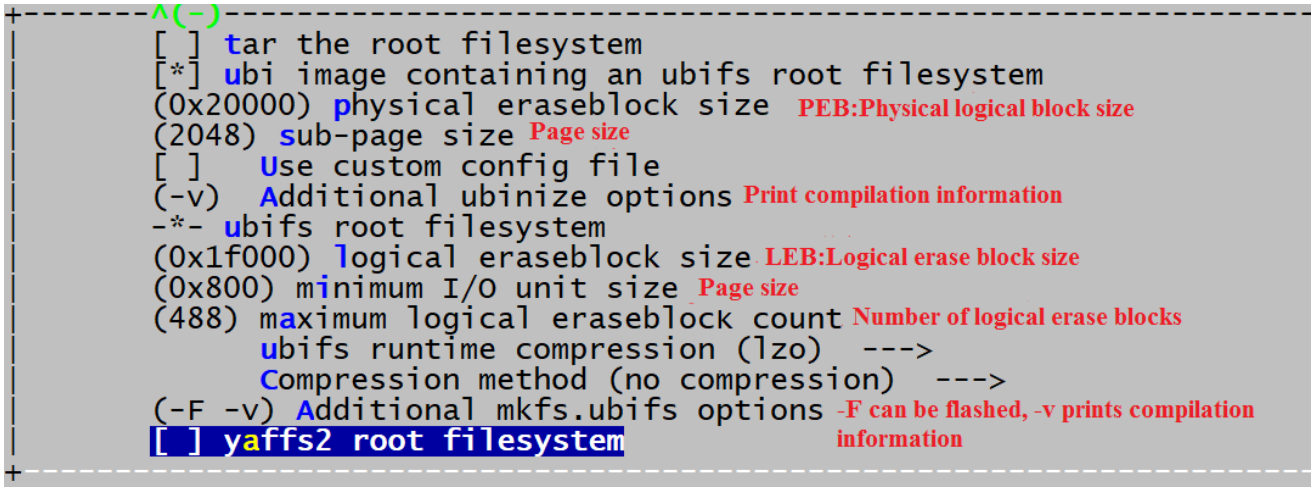
```
CONFIG_MTD=y
CONFIG_MTD_CMDLINE_PARTS=y
CONFIG_MTD_NAND=y
CONFIG_MTD_NAND_ROCKCHIP_V6=y
CONFIG_MTD_UBI=y
CONFIG_MTD_UBI_WL_THRESHOLD=1024
CONFIG_UBIFS_FS=y
CONFIG_UBIFS_FS_ADVANCED_COMPR=y
# CONFIG_UBIFS_FS_ZLIB is not set
CONFIG_CRYPTO_DEFLATE=y
```

Remove:

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

1.3. Buildroot

Takes ubifs as an rootfs example, refer to the configuration below, and the detailed parameters configurations, please refer to ubifs:



When finishing the configuration, use "make savedefconfig" to save Buildroot configuration.

1.4. Compile Script

Adds spl compilation to build.sh. The generated spl file is located at "u-boot/spl/u-boot-spl.bin". The following compilation script will automatically package spl into MiniloaderAll.bin file.

```
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
```

Modify the following fields in BoardConfig.mk：

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

Package tools modification (tools directory), oem and userdata are not packaged as follows

```
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. Partition Table

The partition table should use GPT table, that is, configure the following fields in parameter.txt file:

```
TYPE: GPT
```

1.6. SD Booting Upgrade

The SPL solution supports SD card upgrade solution. If you need this function, the following configuration should be opened:

U-Boot directory：

```
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-0 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 directory:

```
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);
```

Macros should be turned on in the defconfig of recovery

```
BR2_PACKAGE_MTD=y
```

When finishing compilation, use the tool SDDiskTool_v1.59 to make the card. First time to upgrade needs to generate the firmware needed to upgrade Flash in the root directory of the SD card, so it will take longer time, but it will be quicker later.

2. Upgrade Instructions

2.1. Tool

The AndroidTools supports UBI flashing. If the firmware is identified as UBI, format the partition first and then flash the partition. The tool version must be V2.6.9 or later.

2.2. Enable Key Log

SPL Log:

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

Then run "mount" command and it will mount as follows:

```
# 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 Instructions

3.1. UBIFS Introduction

Unordered Block Image File System (UBIFS) is used on solid-state memories and competes with LogFS as one of JFFS2 subsequent file systems.

3.2. UBI Layer

UBIFS includes three subsystems:

1. MTD system, providing access interfaces to various Flash chips: drivers or mtd.
2. UBI system, working on MTD, providing UBI volume: drivers or mtd or ubi.
3. UBIFS file system, working on UBI, fs or ubifs.

3.3. UBIFS Application Examples

3.3.1. Mount an Empty UBIFS File System

There are 6 partitions in current board. The partitions are as follows:

```
# 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. Format UBI partition

   Format UBI partition with the following command:

   ```
   # 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. Bind UBI to MTD partition

   Use the following command to bind UBI to MTD6 partition:

```
# 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
```

The parameter -m 6 means MTD6 partition is used. The UBI device "ubi1" can be found under "/dev/" only after the UBI is bound to JMTD partition. If a UBI volume has been created, the UBI volume "ubi1_0" can be found and accessed under "/dev/" after binding.

You will find one more device "dev/ubi1" by checking all devices "ls/dev /ubi*",

3. Create UBI Volume

A UBI volume also means a partition of a UBI device. The command to create a UBI volume is as follows:

```
# 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
```

The parameter "/dev/ubi0" is the UBI device created in the previous step.

The parameter "-N ubifs" means that the volume name created is "ubifs".

The parameter "-s SIZE" means the size of the created partition.

Note

The SIZE value should be less than the amount of space provided by the "/de/ubi1" device. You can use the command "ubinfo" to check the currently available LEBs size. As shown below, when the space provided by the current UBI device is 152.6MiB, the usable space is 147.7MiB. Therefore, you should ensure that the size of the created volume is smaller than the available LEBs.

```
# 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
```

The volume needs to be created only once. After successful creation, the volume information will be saved on the UBI device. The next time you start the volume, no need to create a volume. "ubirmvol" is used to delete a volume. When using this command to delete a volume, all information on the volume will be deleted.

4. Mount UBIFS file system

   At this point, the created volume can be mounted to the specified directory by the following command:

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

Or

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

When successfully mount, the following information will be displayed:

```
# 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
```

The following content will be displayed by checking the partition information:

```
# 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
```

The remaining space is not accurate when using UBIFS file system to display partition size. Because UBIFS file saves the compressed content of a file, the compression ratio is related to the content of the file. The remaining space may only show 2M, but in fact a 4M file can be completely saved.

3.3.2. Make a UBIFS Root File System UBI Image

- Make an image file

The mtd-ubifs tool should be use to make a UBIFS file system image by the following command:

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

Parameter -F enables “white-space-fixup”, if you use u-boot or upgrade tool, this function should be enabled.

Parameter -d rootfs_dir indicates that the root directory to be made into UBIFS image is rootfs. This parameter can also be written as -r rootfs_dir. Parameter -m 0x800 indicates that the minimum read and write unit is 2KiB. The page size of the NAND chip used here is 2KiB. The smallest read-write unit refers to FLASH device for a single read or write operation, the minimum number of bytes is 1 page for NAND devices and 1 byte for NOR devices. The parameter -o rootfs.ubifs means that the image name is rootfs.ubifs. The parameter -e 126976 means the logical erase block size.

The minimum read-write unit and logical block size can be obtained by reading MTD and UBI system information, or by calculation.

The command to read MTD information and will display as follows:

```
# 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
```

Command to read UBI information are as follow:

```
# 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
```

The parameter -c 240 indicates that there are at most 240 logical erase blocks in this file system. Calculate "240xLEB" to get the maximum usable space of this file system. LEBs=The maximum usable space of this file system /LEB. The -v parameter displays detailed information during the process of making UBIFS. The logical erase block size can be calculated by the following table:

FLASH	**Logical erase block size**
NOR	LEB=blocksize-128
NAND without subpage	LEB=blocksize - pagesize*2
NAND with subpage	LEB=blocksize - pagesize*1
blocksize	flash physical erase block size
pagesize	flash read-write page size

Note that the successfully created UBIFS root file system image is a UBI image. You can upgrade a empty UBIFS file system in kernel. The image cannot be download directly to MTD partition for use, but it can be converted into a directly downloaded file through format conversion to convert to a format which can be downloaded to MTD partition.

- Convert to a firmware that can directly flash MTD

Make UBI image conversion configuration file ubi.cfg as follows:

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

The parameter mode=ubi is a compulsory parameter. You cannot enter other values at present. It is reserved for future extended. Parameter image=out/rootfs.ubifs, means a source file. The parameter vol_id=0 indicates volume ID. The UBI image may contain multiple volumes. It is used to distinguish different volumes. The parameter vol_type=dynamic indicates that the current volume type is read-write. when it is static means read-only. Parameter vol_name=ubifs, indicates the name of the volume. Parameter vol_flags=autosize, indicates that the size of the volume is expandable.

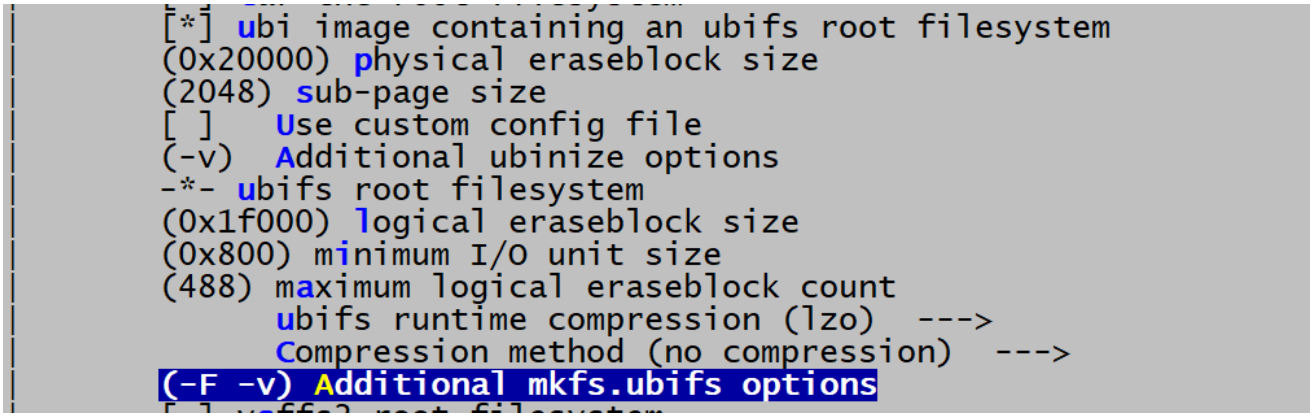
Convert UBI format

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

The parameter -o out/rootfs.img means output file. Parameter -m 2KiB, means that the minimum read and write unit is 2KiB. Parameter -p 128KiB, is flash physical erase size, not logical erase block size. ubi.cfg is configuration file. Parameter -v, shows detailed information of the making process.

3.3.3. Compile UBIFS Root File System UBI Image by Buildroot

Filesystem images --->select as shown below, where 488 is configured according to the size:



Reference documents

[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