君正®

T41 文件系统制作指南

Date:2022-07

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

Trademarks and Permissions

Disclaimer

All the deliverables and data in this folder serve only as a reference for customer development. Please read through this disclaimer carefully before you use the deliverables and data in this folder. You may use the deliverables in this folder or not. However, by using the deliverables and data in this folder, you agree to accept all the content in this disclaimer unconditional and irrevocable. If you do not find the content in this disclaimer reasonable, you shall not use the deliverables and data in this folder.

The deliverables and data in this folder are provided "AS IS" without representations, guarantees or warranties of any kind (either express or implied). To the maximum extent permitted by law, Ingenic Semiconductor Co., Ltd (Ingenic) provides the deliverables and data in this folder without implied representations, guarantees or warranties, including but not limited to implied representations, guarantees and warranties of merchantability, non-infringement, or fitness for a particular purpose. Deviation of the data provided in this folder may exist under different test environments.

Ingenic takes no liability or legal responsibility for any design and development error, incident, negligence, infringement, and loss (including but not limited to any direct, indirect, consequential, or incidental loss) caused by the use of data in this folder. Users shall be responsible for all risks and consequences caused by the use of data in this folder.

北京君正集成电路股份有限公司
地址：北京市海淀区西北旺东路 10 号院东区 14 号楼君正大厦
电话：**(86-10)56345000**
传真：**(86-10)56345001**
Http: //www.ingenic.cn

前言

概述

本文为 Ingenic T41 文件系统制作指南，方便使用者能快速在 T41 DEMB 板上搭建好开发资源环境。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
T41	

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

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修订章节
2022-07	1.0	第一次正式版本发布

目录

1 Busybox 的工具的编译使用

君正发布的 SDK 中自带 busybox 的源码包，可基于此制作自己的文件系统工具集。

1.1 配置文件介绍

图 1-1 busybox 配置文件

如图 1-1 所示，君正根据文件系统的常用工具给出了 4 种配置文件，分别是：isvp_uclibc_defconfig 、 isvp_glibc_defconfig 、 isvp_uclibc_mini_defconfig 、 isvp_glibc_mini_defconfig。其中 isvp_uclibc_defconfig 、isvp_glibc_defconfig 对文件系统下的工具支持比较完全，但是整体文件大小会比较大；isvp_uclibc_mini_defconfig 、isvp_glibc_mini_defconfig 是君正根据上面完全的配置文件裁剪出比较常用的命令。

1.2 编译方法

1.2.1 配置 flag

```
export CFLAGS="-O2 -muclibc"

export CPPFLAGS="-O2 -muclibc"

export CXXFLAGS="-O2 -muclibc"

export LDFLAGS="-O2 -muclibc"

export DSOFLAGS="-O2 -muclibc"
```

配置以上的 flag 是为了编译出 uclibc 的工具，如果编译 glibc，上面的工具可不用配置。

1.2.2 编译选项配置

（1）uclibc 工具编译

直接使用 make isvp_uclibc_mini_defconfig，生成相应的.config 配置文件，如果需要的工具在 isvp_uclibc_mini_defconfig 里面没有，可以使用 make menuconfig 去选择对应的工具。

```
make -j

make install
```

（2）glibc 工具编译

直接使用 make isvp_glibc_mini_defconfig，生成相应的.config 配置文件，如果需要的工具在 isvp_glibc_mini_defconfig 里面没有，可以使用 make menuconfig 去选择对应的工具。

```
make -j

make install
```

```
bin  linuxrc  sbin  usr
jszhang@4:/home_d/jszhang/work/isvp/opensource/busybox/_install$
```

图 1-2 busybox 生成架构

1.3 文件系统基本架构搭建

1.3.1 基本文件系统

基本的文件系统是由如下的结构组成：

```
bin  dev  etc  lib  linuxrc  media  mnt  opt  proc  root  run  sbin  sys  system  tmp  usr  var
```

图 1-3 文件系统基本组成

君正 SDK 提供的有 uclibc 和 glibc 的文件系统，包括 squashfs 模式和.tar.bz2 压缩包，用户可以直接基于.tar.bz2 的压缩包构建。

基本的文件系统是由 busybox 和 buildroot 生成，如果基本文件系统结构已经存在，可以只替换里面的工具和相关的库就能够制作文件系统。

其中 bin/ 、sbin/、usr/下面存放生成的工具，并且链接到 linuxrc 上，用户自己编译的工具也可以软链接上去。

Lib/下面是由工具链下取来的相关的 C 库和工具库，建议用户不要修改，可以去对应的工具链下复制需要的库放到这里。

其他的文件一般都是脚本和配置文件，可以延续使用之前的，用户根据需求可以添加自己的用户目录和脚本。目前的启动脚本存放在/etc/init.d/rcS。

1.3.2 文件系统搭建

直接使用君正提供的基本文件系统压缩包开发，用户可以把 Busybox 生成的相关文件替换到压缩包，然后制作成文件系统就可以使用这些工具。

/lib 库下的内容，用户可以根据需求去添加，建议不要随意减少，以免造成应用程序不能运行的问题。

注意：建议替换之前先删除里面同名的文件，以避免制作的文件系统中存在工具冲突。

2 文件系统制作

常用的文件系统有下面几种方式，用户可根据需求去使用对应的方法。

2.1 Squash 文件系统制作方法

mksquashfs root-uclibc-1.1 root-uclibc-1.1.squashfs -comp xz
参数简介：
通常注意参数”-b”以及”-comp”（-comp 必须配置，-b 可选）
-comp COMPRESSION 压缩格式： gzip (默认), lzo, xz
-b BLOCK_SIZE 块大小，默认 128KB

2.2 Jffs2 文件系统制作方法

mkfs.jffs2 -o root-uclibc-1.1.jffs2 -r root-uclibc-1.1 -e 0x10000 -s 0x1000 -n -l -X zlib --pad=0x10000
参数简介：
-o, --output=FILE 指定生成文件的文件名.(default: stdout)
-r, -d, --root=DIR 指定需要制作的文件夹目录名.(默认：当前文件夹)
-e, --eraseblock=SIZE 设定擦除块的大小为(默认: 64KB)
-s, --pagesize=SIZE 节点页大小(默认: 4KB)
-n, --no-cleanmarkers 指明不添加清楚标记（nand flash 有自己的校检块，存放相关的信息。） 如果挂载后会出现类似：CLEANMARKER node found at 0x0042c000 has totlen 0xc != normal 0x0 的警告，则加上-n 就会消失。
-l, --little-endian 指定使用小端格式
-X, --enable-compressor=COMPRESSOR_NAME 指压缩格式
-p, --pad[=SIZE] 通常用 16 进制表示输出文件系统大小，不足部分用 0xff 补充

2.3 UBI 文件系统制作方法

2.3.1 mkfs.ubifs 制作 UBI 文件系统（不带卷集信息）

mkfs.ubifs -e 0x1f000 -c 568 -m 2048 -d config/ -o config.ubifs -v
参数简介：

-e 0x1f000 ： 表示逻辑擦除块（LEB） 的 size。
-c 568 ： 最多逻辑可擦除块数目为 568(568*128KB=71MB),这个可根据 ubi volume 数量来设置， 实际上是设置此卷的最大容量。
-m 2048 : flash 最小的读写单元， 一般为 flash 的页大小。
-d config/ ： 需要制作成 UBI 文件系统的源文件。
-o config.ubifs ： 制作出来的镜像名称为 config.ubifs。
-v ： 显示制作的详细过程。
备注：制作成功的 UBI 镜像， 是没有创建 volume 的， 不能直接烧录使用， 需要进行创建 volume 步骤（挂载时需要配合 ubiattach ,ubimkvol 命令使用）。
简单的脚本挂载（mtd3）：

```
ubiattach -m 3 -d 3 /dev/ubi_ctrl

avlSize=$((((0x$(cat /proc/mtd | grep "mtd3" | cut -d ' ' -f

2)/128/1024)-20)*124*1024))

ubimkvol /dev/ubi3 -N config -s $avlSize

mount -t ubifs /dev/ubi3_0 /mnt
```

2.3.2 ubinize 制作 UBI 文件系统（带有卷集信息）

ubinize -o config.img -m 2048 -p 128KiB -s 2048 ubinize.cfg -v
参数简介：
-o config.img ： 制作出的镜像名称是 config.img。
-m 2048 ： flash 最小的读写单元， 一般为 flash 的页大小。
-p 128KiB ： 表示物理擦除块（PEB） 的 size， 一般为块大小。
-s 2048 ： UBI 头部信息的最小输入输出单元， 一般与最小读写单元(-m 参数)大小一样。
ubinize.cfg ： 制作带 volume UBI 格式的镜像所需要的配置文件。
-v ： 显示制作过程中的详细参数。

ubinize.cfg 制作：

```
$ touch ubinize.cfg

$ vim ubinize.cfg
```

新建脚本

```
[ubifs]

mode=ubi

image=config.ubifs

vol_id=0

#vol_size=64MB
```

```
vol_type=dynamic

vol_name=config

vol_flags=autoresize

vol_alignment=1
```

参数介绍：
mode=ubi ： 这是个强制参数， 目前只能是 ubi
image=config.ubifs ： 指定 mkfs.ubifs 制作 UBI 文件系统制作的 UBI 镜像， 作为源文件。
vol_id=0 ： 指定卷的 id 为 0。
vol_size=64MB ： 指定该卷的 size 为 64MB， 使用该参数的时候要注意当设置了 vol_flags=autoresize， 这个参数将不起作用， 设置成了自动分配 size。 手动指定 size 的时候， 要注意， 参数的大小不能超过当前分区的最大 size。
vol_type=dynamic ： dynamic： 当前卷为动态卷， 是可读可写的。
static： 表示卷为静态卷， 是只读的。
vol_name=config ： 指定当前卷的卷名为 config。
vol_flags=autoresize ： 指定当前卷为自动分配 size， 当有一个文件系统有两个卷的时候只能有一个为 autoresize。
vol_alignment=1 ： 指定对齐方式， 默认为 1。
备注：通过 ubinize 制作的 ubi 文件系统可以直接烧录使用。

2.4 Yaffs2 文件系统制作

进入 isvp/resource/tools_t41/bin/yaffs2_utils 目录：

1. 注意 Makefile 中内核路径配置

```
$(BASE_LINKS):

    ln -s ../../../../opensource/kernel-4.4.94/fs/yaffs2/$@ $@
```

2. 使用方法：

```
make clean

make
```

3. 生成 mkyaffs2image 后执行：

```
./mkyaffs2image 源文件 目的文件
```

例如：./mkyaffs2image root-uclibc-toolchain720 root-uclibc-toolchain720.bin
注意：如果自己下载的 yaffs2 源码工具，需要做如图 2-1 修改。

```
// Adjust these to match your NAND LAYOUT:
#define chunkSize (2048 - 16)
#define spareSize 16
#define pagesPerBlock 64



typedef struct
{
    dev_t dev;
    ino_t ino;
    int   obj;
} objItem;

struct yaffs_dev dummy_dev = {.swap_endian = 0};


static objItem obj_list[MAX_OBJECTS];
ne_d/tyu/isvp/prebuilt/tools_t40/bin/yaffs2_utils/mkyaffs2image.c[POS=51,1][8%]29/05/21 - 11:24
```

图 2-1 yaffs2 修改

Yaff2 有在 NAND 中会写入 tags 信息，默认使用的是页内 tags，分配在每一页的最后 16bit。

1. 修改参数信息

```
#define chunkSize (2048 - 16) //表示页内有效数据段大小

#define spareSize 16          //表示页内额外数据段，tags

#define pagesPerBlock 64      //NAND 中的块数量
```

```
static void shuffle_oob(char *spareData, struct yaffs_packed_tags2 *pt)
{
    //assert(sizeof(*pt) <= spareSize);
    // NAND LAYOUT: For non-trivial OOB orderings, here would be a good place to shuffle.
    //memcpy(spareData, pt, sizeof(*pt));
    memcpy(spareData, &pt->t, sizeof(pt->t));
}
```

图 2-2 shuffle_oob 信息

2. 如图 2-2 所示，修改 shuffle_oob 函数，不需要额外的 ECC 数据。

3 文件系统烧录

squash 和 yaffs2 文件系统可直接烧录。不带卷的 ubi 文件系统不可直接烧录，需要在 uboot 中通过 ubi 命令来创建一个卷；带卷的 ubi 文件系统可以直接烧录。

以 uboot 与 kernel 烧录完成为前提，分区以 SDK 代码默认分区为例。

3.1 nor flash（sf 命令）

```
isvp_t41# mw.b 0x80600000 0xff 0x200000
isvp_t41# tftpboot 0x80600000 root-uclibc-toolchain720.squashfs
isvp_t41# sf0 probe;sf0 erase 0x2c0000 0x200000;sf0 write 0x80600000
0x2c0000 0x200000
```

- mw.b 0x80600000 0xff 0x200000

清 ddr，将 ddr 地址 0x80600000 开始，大小为 0x200000 的内容置 1。

- tftpboot 0x80600000 root-uclibc-toolchain720.squashfs

通过 tftpboot 下载文件系统到 ddr 地址的 0x80600000 上。

注意：也可利用 SD 卡将文件系统加载到 DDR 上。

- sf0 probe;sf0 erase 0x2c0000 0x200000;sf0 write 0x80600000 0x2c0000 0x200000

将下载到 ddr 上的文件系统写入到 flash 中。

1. sf0 probe

   匹配 nor flash。

2. sf0 erase 0x2c0000 0x200000

   从 nor flash 的 0x2c0000 位置开始擦除，擦除大小为 0x200000。

3. sf0 write 0x80600000 0x2c0000 0x200000

   从 ddr 地址 0x80600000 开始，大小为 0x200000 的内容写入到 nor flash 的 0x2c0000 位置。

3.2 nand flash（nand 命令）

注意：nand 命令不需要执行 probe。

3.2.1 yaffs2 文件系统或者带卷的 ubi 文件系统烧录方法

```
isvp_t41# mw.b 0x80600000 0xff 0x2000000
isvp_t41# tftpboot 0x80600000 rootfs720.ubifs
isvp_t41# nand erase 0x400000 0x2000000;nand write 0x80600000 0x400000
0x2000000
```

- mw.b 0x80600000 0xff 0x2000000

清 ddr，将从 ddr 地址 0x80600000 开始，大小为 0x200000 的内容置 1。

- tftpboot 0x80600000 rootfs720.ubifs

通过 tftp 下载文件系统到 ddr 地址为 0x80600000 上。

注意：也可利用 SD 卡将文件系统加载到 DDR 上。

- nand erase 0x400000 0x2000000

从 nand flash 的 0x400000 位置开始擦除，擦除大小为 0x2000000。

- nand write 0x80600000 0x400000 0x2000000

ddr 地址从 0x80600000 开始，大小为 0x2000000 的内容写入到 nand flash 的 0x400000 位置。

3.2.2 不带卷的 ubi 文件系统烧录方法

```
isvp_t41# mw.b 0x80600000 0xff 0x2000000
isvp_t41# tftpboot 0x80600000 rootfs720.ubifs
isvp_t41# nand erase 0x400000 0x2000000
isvp_t41# mtdparts default
isvp_t41# mtdparts
isvp_t41# ubi part root
isvp_t41# ubi create rootfs
isvp_t41# ubi write 0x80600000 rootfs 0x1000000
```

- mw.b 0x80600000 0xff 0x2000000

清 ddr，将从 ddr 地址 0x80600000 开始，大小为 0x2000000 的内容置 1。

- tftpboot 0x80600000 rootfs720.ubifs

通过 tftp 下载文件系统到 ddr 地址为 0x80600000 上。

注意：也可利用 SD 卡将文件系统加载到 DDR 上。

- nand erase 0x400000 0x2000000

从 nand flash 的 0x400000 位置开始擦除，擦除大小为 0x2000000。

- mtdparts default

载入默认分区表，环境变量中 printenv 出现 mtdparts 参数。

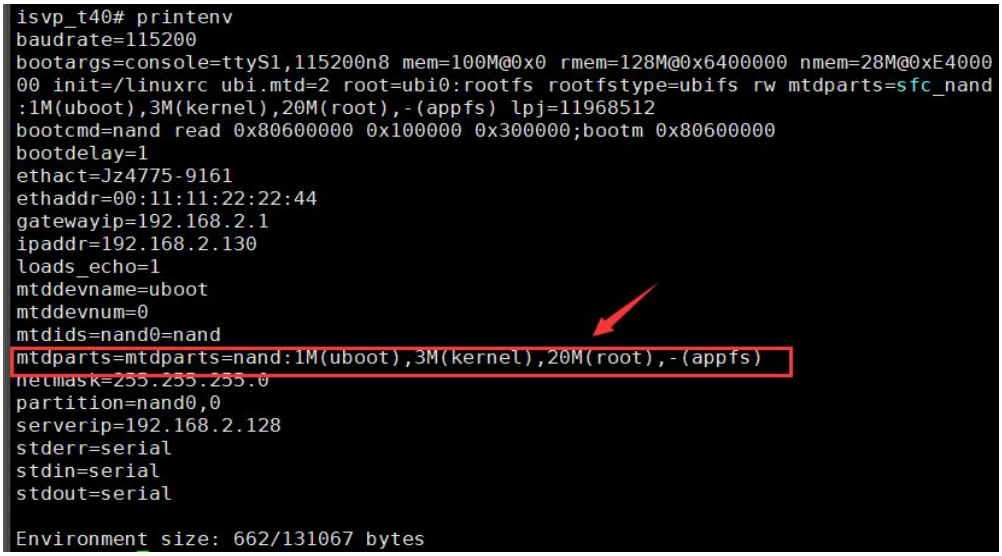

图 3-1　printenv 信息

- mtdparts

查看分区表信息（注意：这个位置的要与 bootargs 分区信息保持一致）

图 3-2　分区表信息

下例为增加 config 分区

1. 增加 config 分区：

```
isvp_t40# mtdparts default
isvp_t40# mtdparts

device nand0 <nand>, # parts = 4
 #: name                size            offset          mask_flags
 0: uboot               0x00100000      0x00000000      0
 1: kernel              0x00300000      0x00100000      0
 2: root                0x01400000      0x00400000      0
 3: appfs               0x0e800000      0x01800000      0

active partition: nand0,0 - (uboot) 0x00100000 @ 0x00000000

defaults:
mtdids  : nand0=nand
mtdparts: mtdparts=nand:1M(uboot),3M(kernel),20M(root),-(appfs)
```

```
isvp_t41# setenv mtdparts

'mtdparts=nand:1M(uboot),3M(kernel),20M(root),15M(config),-(appfs)'
```

2. 设置 bootargs 分区信息

```
isvp_t41# setenv bootargs 'console=ttyS1,115200n8 mem=100M@0x0

rmem=128M@0x6400000 nmem=28M@0xE400000 init=/linuxrc ubi.mtd=2

root=ubi0:rootfs rootfstype=ubifs rw

mtdparts=sfc_nand:1M(uboot),3M(kernel),20M(root),15M(config),-(appfs)

lpj=11968512'
```

3. 保存环境变量

```
isvp_t41# saveenv
```

- ubi part root

将指定的 root 分区绑定到 UBI 上，作用和 ubiattach 类似。

```
isvp_t40# ubi part root
UBI: attaching mtd1 to ubi0
UBI: physical eraseblock size:   131072 bytes (128 KiB)
UBI: logical eraseblock size:    126976 bytes
UBI: smallest flash I/O unit:    2048
UBI: VID header offset:          2048 (aligned 2048)
UBI: data offset:                4096
UBI: empty MTD device detected
UBI: create volume table (copy #1)
UBI: create volume table (copy #2)
UBI: attached mtd1 to ubi0
UBI: MTD device name:            "mtd=2"
UBI: MTD device size:            20 MiB
UBI: number of good PEBs:        160
UBI: number of bad PEBs:         0
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     0
UBI: available PEBs:             154
UBI: total number of reserved PEBs: 6
UBI: number of PEBs reserved for bad PEB handling: 2
UBI: max/mean erase counter: 1/0
```

图 3-3 ubi part 参数

- ubi create rootfs <size> <type>

创建一个 rootfs (volume name)，不设置 size 和 type，默认创建一个动态卷。

```
isvp_t40# ubi create rootfs
No size specified -> Using max size (19554304)
Creating dynamic volume rootfs of size 19554304
```

1. <size>

就是该分区的最大 size，如需设置这个值需要注意下，参考图 3-3 中的两个参数：

```
UBI: logical eraseblock size:    126976 bytes

UBI: available PEBs:             154
```

size 的大小不能超过[logical eraseblock size] * [available PEBs]，即

126976 bytes * 154 = 19554304 bytes = 19096 KB ≈ 18.6MB

2. <type>

s(static),表示该 volume 为静态卷，挂载后为只读的一个卷；

d[dynamic],表示是动态卷，挂载后是可读可写。

- ubi write <ddr addr> <volue name> <ubi_img size>

前面的步骤中，我们已将文件系统数据加载到 ddr 0x80600000 地址上，这一步将 ddr 上文件系统写入创建的 volume 中 ,作用和 ubiupdatevol 类似。

```
isvp_t40# ubi write 0x80600000 rootfs 0x1000000
16777216 bytes written to volume rootfs
isvp_t40#
```