Ingenic
Semiconductor

Ingenic ms800 平台使用说明

文档历史：

版本	作者	注释
1.0		

目录

1 系统简介

ilock 平台是由君正基于 X1000 芯片自主研发设计的，应用于智能门锁等市场的软件方案，使用前请详细阅读本说明文档, 本文档将对如何获取 Ingenic ms800 平台代码以及软件系统结构、编译、烧录以及应用开发做详细描述。

1.1 源码获取

A) 下面以 ubuntu 系统说明代码获取过程:

1. 获取 repo 工具: wget http://git.ingenic.cn:8082/bj/repo
2. 完后给 repo 添加可执行权限：chmod +x repo，再把 repo 的路径添加到环境变量 PATH 中
3. 安装 git 和 gitk: sudo apt-get install git gitk
4. 创建 ssh public key: ssh-keygen -t rsa ，执行这条命令之后（注意这条命令不能以 root 用户来执行），一路回车即可完成，成功后需要把~/.ssh/id_rsa.pub 文件内容提供给我们
5. 建立存放平台代码的目录: mkdir mpos; cd mpos
6. 同步初始化：
   - repo init -u ssh://usrname@58.250.243.8:29418/mirror/mpos/manifest.git
   - repo sync

1.2 目录结构

工程库源码下载结束后，通过 ls 命令查看工程库目录结构:

```
qwwang@pc:~/work/mpos$ ls
app  bootloader  build  device  docs  external  kernel  sdk  tools
```

各级目录主要包含以下内容

app: 此目录为君正提供 APP 示例。
　　用户可在此目录下使用 SDK 开发自己的 APP
bootloader: 包含君正自主研发的 x-loader 启动引导程序
build: 包含工程编译的环境配置脚本和 Makefile
device: 包含平台通用文件系统和特定设备的文件系统 patch 文件及编译配置文件
docs: 包含平台相关说明文档文件
external: 包含平台拓展模块
kernel: 包含 linux3.10 内核
sdk: 包含 sdk 源码，详细介绍请参考文档”Ingenic SDK 使用说明.pdf”
tools: 包含编译工具链、编译时 PC 所需的工具以及烧录工具

2 编译说明

2.1 全局编译

本章主要说明如何自动编译工程源码，以生成烧录镜像、sdk 共享库和应用程序

2.1.1 编译步骤

在系统顶级目录执行以下命令:

1. 配置编译环境，在工程根目录执行 source build/envsetup.sh，再执行 lunch，如下图:

```
qwwang@pc:~/work/mpos$ source build/envsetup.sh
including /home/qwwang/work/mpos/device/ms800/vendorsetup.sh
OK!!! You need to execute the 'lunch' ...
qwwang@pc:~/work/mpos$ lunch

You're building on Linux
Lunch menu... pick a combo:
     1. ms800-v10-norflash-eng
     2. ms800-v10-spinand-eng

Which would you like? [ms800-v10-norflash-eng] 2
```

说明:
ms800-v10: 硬件的版本
norflash/spinand: 存储介质的类型

2. lunch 之后显示选择菜单，下面以 nandflash 来说明，选择 2
3. 选择编译的目标设备和存储介质后，将在工程根目录生成 Makefile，可执行 make help :

```
qwwang@pc:~/work/mpos$ make help
Welcome to use Ingenic ms800 platform...
----------------------------------------------------------------
Compile cmd:
   make or make all          - will compiling all
   make build_xxxx           - will only compiling xxxx
   make clean                - will clean all except TOPDIR/out
   make distclean            - will clean all include TOPDIR/out

Envsetup info:
   TARGET_DEVICE       = ms800
   ROOTFS_TARGET_TYPE  = ubi
   LOADER_BUILD_CONFIG = ms800_nand_config
   KERNEL_BUILD_CONFIG = ms800_v10_linux_sfcnand_ubi_defconfig
----------------------------------------------------------------
```

从 make help 所获取的信息中，我们可以知道编译命令和配置信息，注意 make clean 和 make distclean 的区别，更详细请看 Makefile 文件。

4. 整体编译工程: make

```
qwwang@pc:~/work/mpos$ make
test -e /home/qwwang/work/mpos/out/ms800/system || cp -af /home/qwwang/work/mpos/device/common
/system /home/qwwang/work/mpos/out/ms800
>>>  Compiling sdk...
make ms800_sdk_defconfig -C /home/qwwang/work/mpos/sdk
make[1]: Entering directory `/home/qwwang/work/mpos/sdk'
```

成功编译后的的打印输出，如下图:

```
>>>   Building rootfs...
/home/qwwang/work/mpos/device/ms800/spinand/system_patch.sh
cp -af /home/qwwang/work/mpos/device/ms800/spinand/system.patch/* /home/qwwang/work/mpos/out/m
s800/system
/home/qwwang/work/mpos/tools/host/mkfs.ubifs -e 0x1f000 -c 2048 -m 0x800 -d /home/qwwang/work/
mpos/out/ms800/system -o /home/qwwang/work/mpos/out/ms800/image/system.ubi

#### make completed successfully (22 seconds) ####
```

说明: 从上图可以看出，全局编译最后阶段是“Building rootfs”，这一阶段包含三个步骤:

1. 调用 system_patch.sh 脚步对 out/$(TARGET_DEVICE)/system 打 patch
2. 把 system.patch/目录所有内容拷贝到 out/$(TARGET_DEVICE)/system
3. 调用 mkfs.ubifs 工具把 out/$(TARGET_DEVICE)/system 打包成 ubi 系统烧录文件

2.1.2 生成文件

编译成功后，生成文件位于 out/$(TARGET_DEVICE)目录下，执行 tree –L 2 out/ms800/，如图:

```
qwwang@pc:~/work/mpos$ tree -L 2 out/ms800/
out/ms800/
├── image
│   ├── system.ubi
│   ├── xImage
│   └── x-loader-pad-with-sleep-lib.bin
├── sdk
│   ├── app
│   ├── include -> /home/qwwang/work/mpos/sdk/include
│   ├── lib
│   └── testunit
└── system
    ├── bin
    ├── etc
    ├── lib
    ├── lib32 -> lib
    ├── linuxrc -> bin/busybox
    ├── media
    ├── mnt
    ├── opt
    ├── proc
    ├── root
    ├── run -> tmp
    ├── sbin
    ├── sys
    ├── testsuite
    ├── tmp
    ├── usr
    └── var
```

下面具体介绍编译的输出文件:

```
out/ilock/
├── image
│   ├── system.ubi  //文件系统镜像，rootfs 分区的烧录文件，对 nandflash，这个为 system.ubi
│   ├── xImage   //编译 kernel 生成文件，kernel 分区的烧录文件
│   └── x-loader-pad-with-sleep-lib.bin  //系统的引导程序，bootloader 分区的烧录文件
├── sdk
│   ├── app/      //编译 app 生成的可执行文件的输出目录，跟 app/out/一样
│   ├── include    //软连接文件，指向 sdk 供 app 开发使用的头文件
│   ├── lib/      //编译 sdk 输出的动态库，供 app 开发使用，会被自动安装到文件系统
│   └── testunit/   //sdk 各个模块的测试程序，具体使用方法，请参考源码
└── system/       //文件系统镜像就是使用相应的工具打包此目录生成的
```

2.1.3 运行测试

通过烧录工具烧录 out/ms800/image 目录下镜像文件，观察系统启动运行结果，如下:

```
X-Loader Build: May 18 2017 - 17:33:18
reset errorpc: 0x800ce474
Going to boot next stage.
Mod: Normal.
Load address:  0x80efffc0
Entry address: 0x80f00000

Jump...

Uncompressing Linux...
Ok, booting the kernel.
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.10.14-00221-ge000b1a (qwwang@pc) (gcc version 4.7.2 (Inge7
[    0.000000] bootconsole [early0] enabled
[    0.000000] CPU0 RESET ERROR PC:800CE474
[    0.000000] [<800ce474>] kfree+0x34/0x21c
[    0.000000] CPU0 revision is: 2ed1024f (Ingenic Xburst)
[    0.000000] FPU revision is: 00330000
[    0.000000] CCLK:1008MHz L2CLK:504Mhz H0CLK:200MHz H2CLK:200Mhz PCLK:100Mhz
```

...

...

```
[    1.002579] jffs2: notice: (1) jffs2_build_xattr_subsystem: complete building xattr s.
[    1.004827] VFS: Mounted root (jffs2 filesystem) on device 31:3.
[    1.006203] Freeing unused kernel memory: 204K (8061d000 - 80650000)
mknod: /dev/null: File exists
mknod: /dev/console: File exists
Starting logging: OK
Starting mdev...
Copied 20 bytes from address 0x00000000 in flash to /etc/oeminfo/sn
read sn success!
Copied 12 bytes from address 0x00001000 in flash to /etc/oeminfo/mac
read mac success!
Initializing random number generator... done.
Starting network...
```

从以上的打印信息输开始，标志着 kernel 已经启动并且 mount 文件系统完成，可以开始运行用户自定义的程序。

2.2 分段编译

分段编译可以在工程根目录下通过 make build_xxx 来进行，也可以到某个部分的目录下进行编译，但这两者是有区别：通过 make build_xxx 编译某个部分，生成的目标文件会被拷贝到 out/$(TARGET_DEVICE)/ 目录下的设定目录，另一种方式则不会拷贝目标文件。

2.2.1 单独编译某个部分

单独编译工程的某个部分首先需要配置编译环境，主要让 make 能找到交叉编译工具链，参考全局编译前两个步骤即可。

1） bootloader 单独编译，以 ms800-v10 板级 nandflash 为例:
- 目录: cd bootloader/x-loader
- 配置: make ms800_nand_config
- 编译: make
- 目标: out/x-loader-pad-with-sleep-lib.bin
- 说明: x-loader 是由君正独立自主开发的引导程序， 串口波特率为 3M

2） kernel 单独编译，以 ms800-v10 板级 nandflash 为例:
- 目录: cd kernel
- 配置: make ms800_v10_linux_sfcnand_ubi_defconfig
- 编译: make xImage –j4
- 目标: arch/mips/boot/zcompressed/xImage

3） sdk 单独编译
- 目录: cd sdk
- 编译: make ms800_sdk_defconfig; make
- 目标: out/system/和 out/examples/test_*
- 说明: sdk 是由君正基于 linux 开发的一套 API 程序，API 说明请看 SDK 的说明文档

4）app 单独编译
- 目录: cd app
- 编译: make
- 目标: out/ilock_app

注意: 以上方式编译生成的目标文件都不会拷贝到 TOPDIR/out/设定的目录下，在工程根目录使用 make xxx 编译某一部分是本文档推荐的分段编译方式。

2.2.2 编译应用程序

本节主要说明如何单独编译安装 SDK 接口库以及应用程序。
在工程顶级目录执行以下命令:
1. 设置系统环境变量 source build/source.sh; lunch （说明: 如果已经设置，此步省略）
2. 清理上次编译结果 make clean_app
3. 开始编译 make app

详细请见下图:

```
qwwang@pc:~/work/mpos$ make app
test -e /home/qwwang/work/mpos/out/ms800/system || cp -af /home/qwwang/work/mpos/device/common
/system /home/qwwang/work/mpos/out/ms800
>>>  Compiling sdk...
make ms800_sdk_defconfig -C /home/qwwang/work/mpos/sdk
make[1]: Entering directory `/home/qwwang/work/mpos/sdk'
#
# ms800_sdk_defconfig is written to .config
#
make[1]: Leaving directory `/home/qwwang/work/mpos/sdk'
make -j8 -C /home/qwwang/work/mpos/sdk
make[1]: Entering directory `/home/qwwang/work/mpos/sdk'
make[2]: Entering directory `/home/qwwang/work/mpos/sdk/examples'
  LINK  test_tm1620
  LINK  test_timer
...

>>>  Compiling app...
make -C /home/qwwang/work/mpos/app
make[1]: Entering directory `/home/qwwang/work/mpos/app'
mips-linux-gnu-gcc /home/qwwang/work/mpos/app/target/*.o -EL -Os -g -std=gnu11 -mhard-float -L
/home/qwwang/work/mpos/app/lib -L/home/qwwang/work/mpos/app/../sdk/lib/ms800 -Wl,-Bstatic -Wl,
-Bdynamic -lpthread -lrt -lm -lingenic -lavutil -lswscale -o /home/qwwang/work/mpos/app/out/ms
800_app

...
 Target: "/home/qwwang/work/mpos/app/out/ms800_app" is ready

make[1]: Leaving directory `/home/qwwang/work/mpos/app'
test -e /home/qwwang/work/mpos/out/ms800/sdk/app || mkdir -p /home/qwwang/work/mpos/out/ms800/
sdk/app
cp -f /home/qwwang/work/mpos/app/out/ms800_app /home/qwwang/work/mpos/out/ms800/sdk/app
cp -f /home/qwwang/work/mpos/app/out/ms800_app /home/qwwang/work/mpos/out/ms800/system/usr/bin

#### make app successfully (1 seconds) ####
```

具体包含以下过程:

1 自动编译 sdk 接口库源码
2 自动安装 sdk 共享库到文件系统 usr/lib 目录下
3 自动安装 sdk 共享库、头文件以及测试用例到 out/ms800/sdk 相应的目录下
4 自动安装 sdk 共享库、头文件到 app 相应的目录下，以供给应用开发
5 自动编译 app 且将生成 app 拷贝到 out/ms800/sdk/app 以及文件系统 usr/bin 目录下

通过以上过程，会得到 sdk 和 app 的编译目标文件，并且被安装自动安装到文件系统目录中，再执行 make build_rootfs 将可以得到包含 sdk 共享库和 app 应用的文件系统镜像

2.2.3 制作文件系统镜像

本节主要说明如何单独生成文件系统镜像。
在工程顶级目录执行以下命令:
1. 设置系统环境变量 source build/source.sh; lunch （说明: 如果已经设置，此步省略）
2. 开始编译 make build_rootfs
详细请见下图:

这条命令的执行过程，通过上图的打印信息已经很清除的表示出来，最终生成 out/ilock/image/system.ubi（不同类型的存储介质，制作镜像的工具和命令以及镜像名都不一样）。

```
qwwang@pc:~/work/mpos$ make build_rootfs
test -e /home/qwwang/work/mpos/out/ms800/system || cp -af /home/qwwang/work/mpos/device/common
/system /home/qwwang/work/mpos/out/ms800
>>>   Building rootfs...
/home/qwwang/work/mpos/device/ms800/spinand/system_patch.sh
cp -af /home/qwwang/work/mpos/device/ms800/spinand/system.patch/* /home/qwwang/work/mpos/out/m
s800/system
/home/qwwang/work/mpos/tools/host/mkfs.ubifs -e 0x1f000 -c 2048 -m 0x800 -d /home/qwwang/work/
mpos/out/ms800/system -o /home/qwwang/work/mpos/out/ms800/image/system.ubi

#### make build_rootfs successfully (1 seconds) ####
```

3 烧录说明

本章将简单说明烧录工具的使用和各个分区及其烧录文件。

3.1 烧录配置

目前最新的烧录工具在 tools/burntools/目录下，如下图:

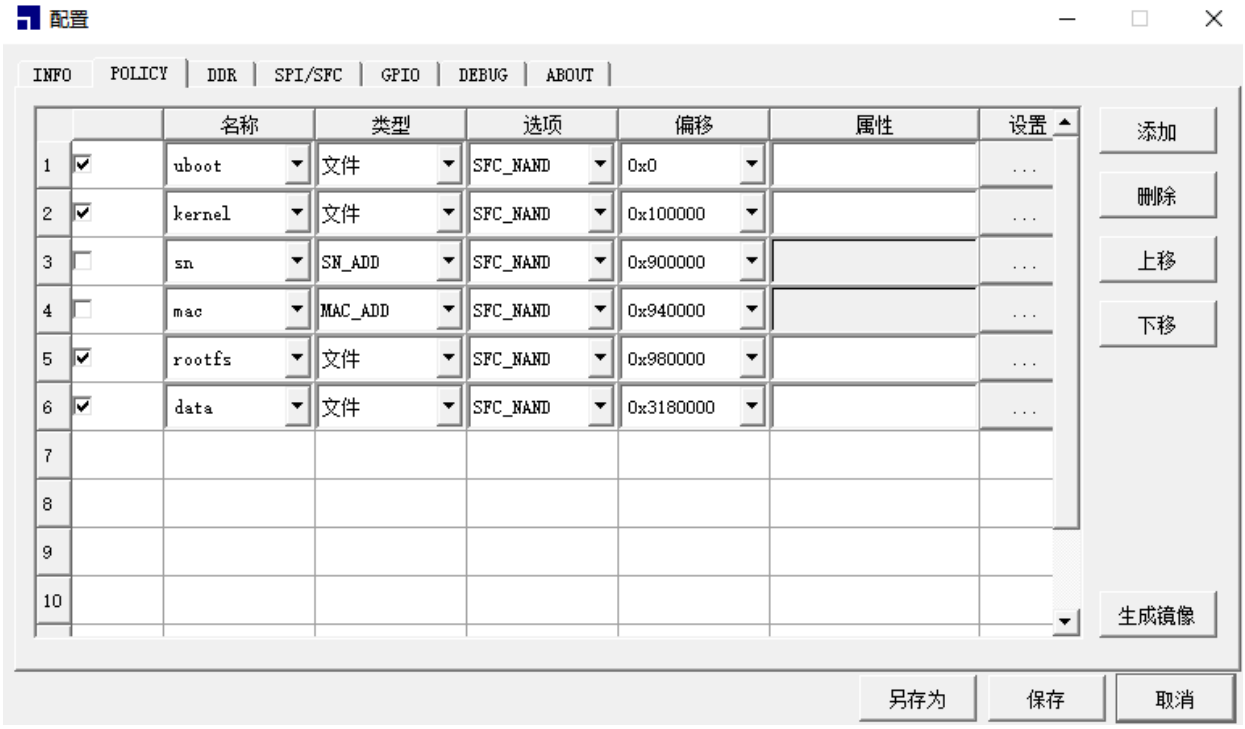
```
qwwang@pc:~/work/ilock/tools/burntools$ ls -l
总用量 59660
-rw-rw-r--  1 qwwang qwwang 46388952 Jul 25 11:59 cloner-2.2.0-ubuntu_release.tar.gz
-rw-rw-r--  1 qwwang qwwang 11172117 Jul 25 11:59 cloner-2.2.0-windows_release.zip
drwxrwxr-x 10 qwwang qwwang     4096 May 18 10:31 ClonerAllVersions
-rw-rw-r--  1 qwwang qwwang   617125 Jun 22 18:32 USBCloner烧录工具快速上手指南.pdf
-rw-rw-r--  1 qwwang qwwang  2903845 Jun 22 18:32 USBCloner烧录工具说明文档.pdf
```

烧录工具详细的使用方法请参考文档《USBCloner 烧录工具快速上手指南.pdf》和《USBCloner 烧录工具说明文档.pdf》，默认配置文件是 x1000_sfc_nand_lpddr_linux.cfg (注意: 烧录工具版本和配置文件名可能会更新，对 norflash 的板，配置文件选 x1000_sfc_nor_quad_lpddr_linux.cfg), 如下图:

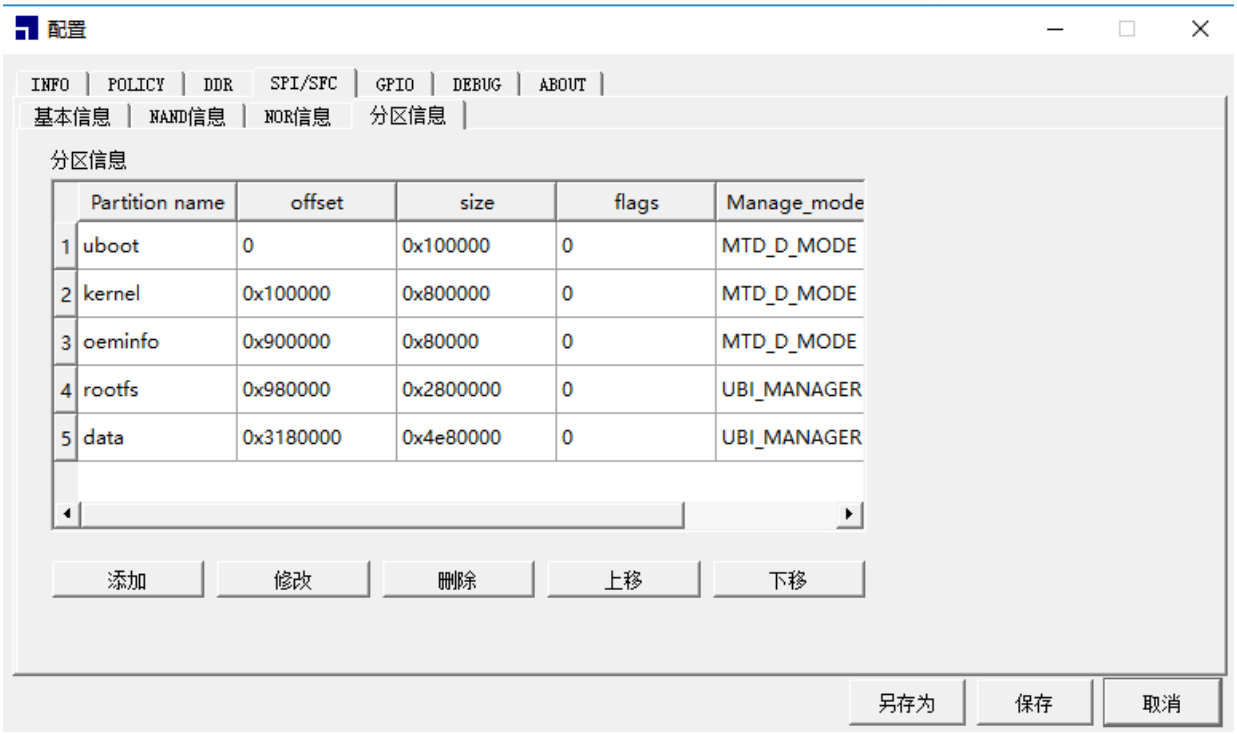
默认是一下 5 个分区:

1. uboot 是 bootloader 分区，烧录镜像 x-loader-pad-with-sleep-lib.bin

2. bootimg 是 kernel 分区，烧录镜像 xImage
3. sn 和 mac 同属于 oeminfo 分区
4. rootfs 是文件系统分区, 烧录镜像 system.ubi 或是 system.jffs2, 根据所选文件系统类型来决定
5. data 是用户数据分区，用户可以根据需要决定这个分区的用途或者去掉这个分区, 如下图:



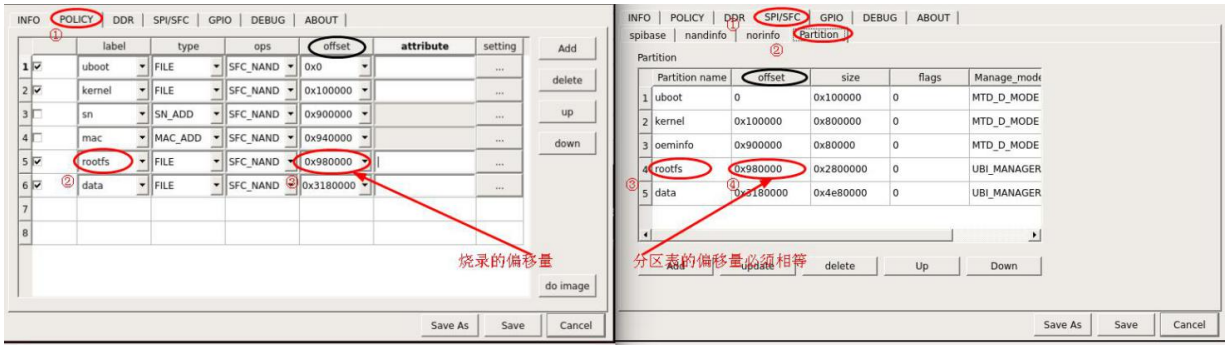
分区信息:



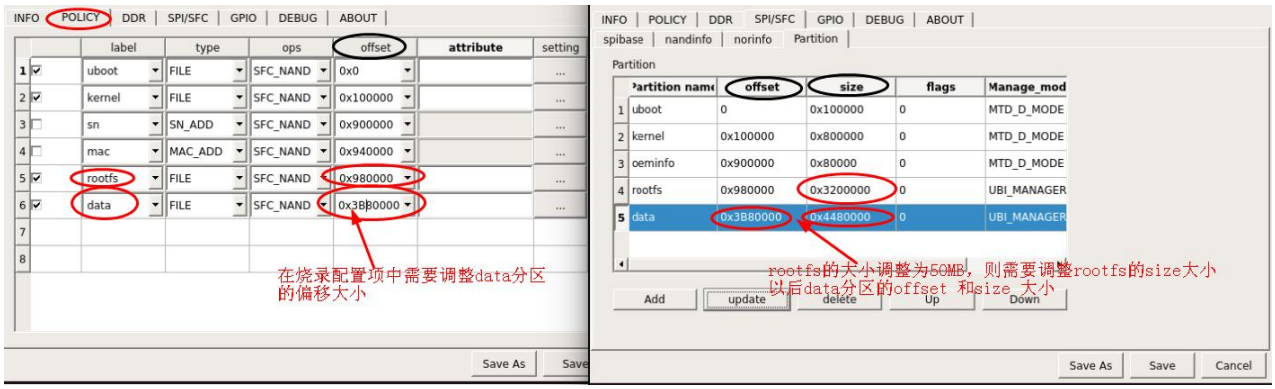
说明:

1. 系统的分区表是通过烧录工具写入到存储介质特定的位置上，kernel 启动是会读出分区。

如果需更改分区表的大小、添加或删除分区，注意 POLICY 和 SPI/SFC-→Partition 中的
offset 必须一致，并且 x-loader kernel 分区的偏移量、大小以及给 kernel 的参数也需要做相
应的修改，否则会导致系统启动失败。



例如：如果想将 rootfs 分区增大(为 50MB)，则需要调整 rootfs 之后所有分区的起始位置和 size 大小。如下：



2. 制作 UBI 文件系统实用工具 mkfs.ubifs.
测试示例如下：

```
mkfs.ubifs -e 0x1f000 -c 2048 -m 0x800 -d $SRC_PATCH -o $TARGET_UBI_SYSTEM

-m, --min-io-size=SIZE   minimum I/O unit size
-e, --leb-size=SIZE      logical erase block size
-c, --max-leb-cnt=COUNT  maximum logical erase block count
-o, --output=FILE        output to FILE
```

3. 启动自动挂载 data 分区文件
A 在文件系统中增加 data 目录
B 在 /etc/fstab 文件中添加如下一行，data 分区挂在在 data 目录下

```
/dev/ubi1_0      /data       ubifs   defaults        0     0
```

4 应用程序

4.1 运行效果

应用程序源码默认路径位于工程根目录的 app 目录下，其中已经包含了默认的 app 测试 demo 程序，建议用户在此目录下开发自己的应用。

4.2 应用变更

用户在 app 目录下部署安排自己的应用程序，每次变更内容后，请注意可能需要根据修改情况自行更新 **Makefile** 重新编译。

新编译出来的 app 应用程序可以通过 adb 上传到板子上，再到板子运行该程序，即可看程序的运行效果。另一种方法是在工程跟目录执行 make app，再执行 make build_rootfs 得到包含新 app 应用程序的文件系统镜像，重新烧录文件系统分区即可。

5 ADB 调试

5.1 使用说明

adb 调试功能用在产品开发调试阶段，主要目的是减少反复烧录的过程，提高程序的开发调试的效率。如何使用 adb 调试功能？以 ubuntu 系统为例:

1. 安装 adb 工具: sudo apt-get install android-tools-adb，（如果已经安装忽略此步，使用虚拟机的需要在 windows 系统下安装 adb）
2. 查看板子串口的系统启动信息，如果有下图的信息，说明支持 adb 并启动成功:

```
Starting adb...
[    1.524409] android_usb: already disabled
/ # -v- Handling main() -v-
-v- init_transport_registration:658 -v-
install_listener('tcp:5037','*smartsocket*')
-v- usb_init:118 -v-
-v- jdwp_control_init:458 -v-
[    1.550200] adb_open
[    1.550377] adb_bind_config
[    1.804156] android_work: sent uevent USB_STATE=CONNECTED
[    1.804529] android_usb gadget: high-speed config #1: android
[    1.812112] android_work: sent uevent USB_STATE=CONFIGURED
```

以调试 app 应用程序说明 adb 使用过程，首先 adb 上传新 ilock_app 应用和其依赖库

1) adb push app/out/ilock_app /usr/bin
2) adb push lib/libingenic.so /usr/lib （如果需要）
3) 完后到串口终端运行刚才的上传的程序

如何把板子上的文件或程序下载到本地 PC 机？更多 adb 的使用方法找度娘！

6 文件系统

本章将简单介绍整个编译系统制作文件系统镜像的过程以及怎样向文件系统添加用户自定义的文件或程序。

6.1 制作过程

以 spinand flash 为例，介绍编译系统制作文件系统镜像的过程，如下流程:

1. 在源码根目录执行 make 命令时, 如果 out/$(TARGET_DEVICE)/system 目录不存在，首先将 device/common/system 整个目录拷贝到 out/$(TARGET_DEVICE)目录下;
2. 然后将 sdk 编译的动态库和 app 编译的应用程序 demo 拷贝到 out/$(TARGET_DEVICE)/system 的相应目录;
3. 在制作系统镜像之前, 调用 device/$(TARGET_DEVICE)/spinand/system_patch.sh 脚本对 out/$(TARGET_DEVICE)/system 目录打补丁;
4. 接着把 device/$(TARGET_DEVICE)/spinand/system.patch/ 目录下的所有内容拷贝到 out/$(TARGET_DEVICE)/system 目录下;
5. 最后使用 tools/host/mkfs.ubifs 工具把 out/$(TARGET_DEVICE)/system 目录的内容打包成文件系统镜像;

说明:

norflash 的文件系统制作流程也是一样的，不同在于用 device/$(TARGET_DEVICE)/norflash 目录下文件打补丁。

6.2 定制文件系统

了解文件系统镜像制作过程后，在定制文件系统时，用户可以修改 system_patch.sh 脚本和修改、添加或删除 system.patch 目录下的内容。例如让系统启动后自动后台运行 ms800_app 应用程序，可以在 system.patch/etc/profile 脚本中添加一行 /usr/bin/ms800_app &，然后执行 make build_rootfs 重新制作文件系统镜像，最后烧录新的文件系统镜像即可。