SigmaStar Camera
系统分区

REVISION HISTORY

Revision No.	Description	Date
{01}	• {Initial release}	{07/05/2018}
{02}	• {Refine}	{12/23/2019}
{03}	• {Match new partition}	{01/02/2020}
{04}	• {Remove some chip name}	{03/06/2020}

1. 目录

目录

2. 分区介绍

2.1. SPI-NOR Flash 基本分区介绍：

ALKAID 在 make image 完成后会打印所有分区的 layout：

```
MXP count max 30
NOR FLASH HAS USED 0x1000000KB
          IPL: 0x00000000-0x00010000 size:64KB
     IPL_CUST: 0x00010000-0x00020000 size:64KB
         MXPT: 0x00020000-0x00030000 size:64KB
        UBOOT: 0x00030000-0x0004F000 size:124KB
    UBOOT_ENV: 0x0004F000-0x00050000 size:4KB
         BOOT: 0x00000000-0x00050000 size:320KB
       KERNEL: 0x00050000-0x00250000 size:2048KB
       rootfs: 0x00250000-0x00650000 size:4096KB
    miservice: 0x00650000-0x00950000 size:3072KB
     customer: 0x00950000-0x01000000 size:6848KB
Available MXP record count:10
```

IPL：

CPU 上电后首先跑到的是 rom code，顾名思义代码保存在特殊的 ROM 中，且是只读的。ROM code 跑完后会读取 NOR Flash 0 地址的位置，这个位置就是 IPL 文件存放的位置，IPL 里主要功能是做一些基础的硬件初始化，例如设定当前 DDR 参数，以及 GPIO/IIC 相关等。

IPL_CUST：

IPL 初始化的是一些共有的硬件模块，IPL_CUST 中会根据当前板子的实际情况初始化客制化板子硬件的可执行的二进制文件，例如客制化的 GPIO 管教，IIC 配置。

MXPT：

分区配置相关的二进制档案。

UBOOT：

UBOOT 的二进制文件存放分区。

UBOOT_ENV：

UBOOT 的环境变量存放分区。

LOGO:

在 NVR 设备上会使用，存放的是开机 logo 相关的配置。

SPI-NOR 的 BOOT 分区中的内容是不建议改动的，若有不符合公板 release 的地方，可能会导致系统无法启动。

BOOT 的位置在 linux 在 mtd block0 的位置上，对应的设备节点/dev/mtdblock0。

BOOT 分区之后的分区，统称为 SYS 分区，SYS 分区是可以修改的，根据实际的使用情况，大致分为四个类别：

KERNEL：

存放内核的二进制文件。

ROOTFS：

如题。

miservice:

这是公板定义的一个分区，它用来存储 mi 的库、一些配置文档，文件系统默认 jiffs2。

customer:

客户自己定制的分区。

2.2. SPI-NAND Flash 基本分区介绍：

Layout 图举例：

```
BOOT: 0x140000(CIS),0x20000@0x140000(IPL0)5,0x20000(IPL_CUST0)2,0x20000(IPL_CUST1)2,0x4
SYS: 0x500000(KERNEL),0x500000(RECOVERY),0x600000(rootfs),-(UBI)
FLASH HAS USED 0x8000000KB
ChkSum        : 1043
SpareByteCnt  : 64
PageByteCnt   : 2048
BlkPageCnt    : 64
BlkCnt        : 1024
PartCnt       : 11
UnitByteCnt   : 8
Checksum ok!!
IDX:          StartBlk:           BlkCnt:     BackupBlkCnt:        PartType:
  0:     0,(0000000000)   10,(0X00140000)     0,(0000000000)   0X0020,(CIS)
  1:    10,(0X00140000)    1,(0X00020000)     5,(0X000A0000)   0X0003,(IPL)
  2:    16,(0X00200000)    1,(0X00020000)     2,(0X00040000)   0X0001,(IPL_CUST)
  3:    19,(0X00260000)    1,(0X00020000)     2,(0X00040000)   0X0001,(IPL_CUST)
  4:    22,(0X002C0000)    2,(0X00040000)     1,(0X00020000)   0X0006,(UBOOT)
  5:    25,(0X00320000)    2,(0X00040000)     1,(0X00020000)   0X0006,(UBOOT)
  6:    28,(0X00380000)    2,(0X00040000)     0,(0000000000)   0X000D,(ENV)
  7:    30,(0X003C0000)   40,(0X00500000)     0,(0000000000)   0X0012,(KERNEL)
  8:    70,(0X008C0000)   40,(0X00500000)     0,(0000000000)   0X0009,(RECOVERY)
  9:   110,(0X00DC0000)   48,(0X00600000)     0,(0000000000)   0X000F,(ROOTFS)
 10:   158,(0X013C0000)  866,(0X06C40000)     0,(0000000000)   0X0021,(UBI)
1+0 records in
```

CIS：

SPI-NAND 独有的分区，保存在 flash 0 地址的位置，它包含两部分内容，一部分是 spinand info，保存 spinand 的一些基本信息，例如 block page count、block count 等，这些信息会根据 spi nand 的种类而改变，另一部分是 partinifo，保存的分区信息，这些信息都是给 rom code 读取的，rom code 通过读取正确的 spinand 参数和分区信息，从而知道 IPL 的位置把整个系统跑起来。

IPL:

IPL 分区的作用与 SPI-NOR 分区一样，但是从上图可知，IPL 在 spi-nand 上保存了六份，每个 block 一份，目的是为了防止坏块而做的备份。

IPL_CUS：

作用同 SPI-NOR，会有两个分区，IPL_CUS0, IPL_CUS1,每个分区中各三份 IPL_CUS 的 data。

UBOOT：

UBOOT 的二进制文件存放分区，会有两个分区，每个分区中一份 data。

ENV：

UBOOT 的环境变量存放分区。

LOGO:

在 NVR 设备上会使用，存放的是开机 logo 相关的配置。

以上是 BOOT 相关的分区信息，这部分无法任意修改。

KERNEL：

存放内核的二进制文件。

ROOTFS：

如题。

UBI:

UBI 的内容在上图分区表中不会显示出来,UBI 中会创建多个 ubifs 格式的子分区,客户可以根据需要自行创建。Spinand 的 miservice 分区就是放在 UBI 中。

2.3. MISERVICE 分区内容：

在 MISERVICE 分区的内容被 mount 到了根目录/config/下面，下表中加深字体的文件是必须保持路径一致的，否则有可能会导致系统无法启动。

```
└── config
    ├── board.ini                          ---->保存当前板子的信息
    ├── config_tool                        ---->mi sys 初始化 mmap 内存的应用
    ├── dump_config -> config_tool         ---->dump config 应用，config_tool 的 symbol link。
    ├── dump_mmap -> config_tool           ---->dump mmap 应用，config_tool 的 symbol link。
    ├── fbdev.ini                          ---->Framebuffer 配置，若无 fbdev 需求可删除。
    ├── iqfile
    │   ├── imx307_iqfile.bin
    │   ├── iqfile0.bin -> imx307_iqfile.bin    ---->Sensor IQ 相关的 bin
    │   ├── iqfile1.bin -> imx307_iqfile.bin
    │   ├── iqfile2.bin -> imx307_iqfile.bin
    │   ├── iqfile3.bin -> imx307_iqfile.bin
    │   └── isp_api.xml
    ├── mmap.ini                           ---->Mmap。Mmap 可以参考《Memory Layout 介绍.pdf》
    ├── model                              ---->模块相关配置，可以删除。
    │   ├── Customer_1.ini
    │   ├── Customer_2.ini
    │   └── Customer_auto_test.ini
    ├── modules                            ---->系统 KO 存放路径。
    │   └── 4.9.84
    │       ├── cifs.ko
        ......
    ├── pq                                 ---->NVR 显示 PQ 相关。
    │   ├── Bandwidth_RegTable.bin
    │   ├── Main.bin
    │   └── Main_Ex.bin
    ├── terminfo                           ---->Console 显示相关。
    │   └── v
    │       ├── vt100
        ......
    └── venc_fw                            ---->Encoder 的 firmware
        └── chagall.bin
```

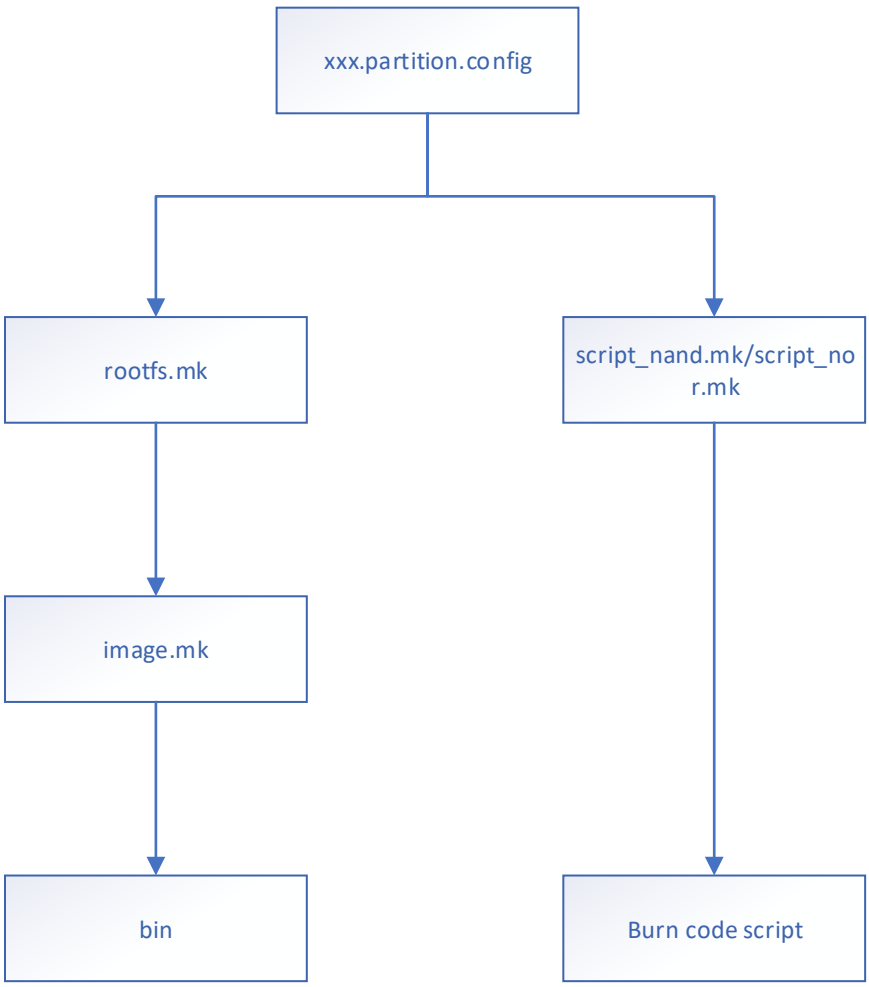
3. 分区变更：

3.1. 分区变更前注意事项

无论是 spi-nor flash 上的分区，还是 spi-nand flash 上的分区。都可归为两类，一类是不建议改动的 boot，一类是可以变动的 sys，这一章节介绍的分区变更都是在 sys 这部分上，包括如何配置脚本变更分区顺序，如何打包分区，如何生成烧录脚本，若客户有自己的一套打包和烧录的流程，可以不用关心这一章，需要特别注意的是在上一章中 miservice 分区中有特别标注出来的文件需要放到指定的路径下系统才能跑起来，所以在制作分区时务必在根目录下创建/config/，并把相关的文件放到此路径下。

3.2. 分区变更的脚本代码架构介绍

分区变更分为分区制作、打包和分区烧录脚本制作这两部分，下面流程图的两个分支，左边走的是分区制作、打包流程，右边走的是分区烧录脚本制作的流程，分区烧录脚本仅支持 tftp 升级脚本，暂不支持 usb 升级脚本制作。



xxx.partition.config：

这是总的分区配置信息脚本，所有与分区调整相关的代码都在这个 config 文件中实现，这个 config 文件会有很多份，每个不同的 chip、spinand 分区还是 spinor 分区都会有各自的分区 config 文件。

通过 build config，就可以找到该项目使用的分区 config 文件是什么，例如某项目使用的 build config：

project/configs/ipc/$(CHIP)/nor.glibc-squashfs.009a.64.qfn88

此配置脚本中，变量 IMAGE_CONFIG 可以找到对应使用的分区配置脚本。

此项目的分区配置脚本存放的路径在： project/mage/configs/$(CHIP)/nor.squashfs.partition.config

可以看出，这是 spinor flash 分区配置脚本文件。

在分区 config 文件中用变量定义了分区 BOOT 的部分和 SYS 的部分：

```
Spinor：
    mxp$(BOOTTAB) = ”xxx”
    mxp$(SYSTAB) = ”xxx”
Spinand:
    cis$(BOOTTAB) = xxx
    cis $(SYSTAB) = xxx
```

在 config 脚本文件中，每个分区由一组变量表示，以上 mxp 和 cis 也是分区的一组变量之一，只不过这两个分区很特殊，存放的是一组规定的分区配置 raw data，这个是一个特殊的分区，特殊的分区由特别的指令生成，在后面章节会有介绍。

rootfs.mk:

这是 ROOTFS 和 miservice 分区制作脚本，项目使用的是：

project/mage/configs/$(CHIP)/rootfs.mk

script_nand.mk/script_nor.mk:

这是制作 spi-nand 和 spi-nor 烧录脚本的脚本，使用的是：

project/mage/configs/$(CHIP)/script_nor.mk

project/mage/configs/$(CHIP)/script_nand.mk

image.mk:

分区打包脚本，路径在 project/mage/image.mk。

3.3. 分区位置变更

3.3.1 Spinor flash

改变 mxp$(SYSTAB)中所定义的分区前后位置即可，语法格式与 linux uboot 中使用的 mtdpart 是一样的。

mxp$(SYSTAB) =
"$(kernel$(PATSIZE))(KERNEL),$(rootfs$(PATSIZE))(rootfs),$(miservice$(PATSIZE))(miservice),$(customer$(PATSIZE))(customer)"

其中 kernel 和 rootfs 的分区信息由逗号 ’,’ 隔开。

3.3.2 Spinand flash

改变 cis $(SYSTAB)中所定义的分区前后位置即可，语法格式与 linux uboot 中使用的 mtdpart 是一样的，这个变量的值也会保存到 uboot 中的环境变量中。

cis$(SYSTAB) = $(kernel$(MTDPART)),$(rootfs$(MTDPART)),-(UBI)

其中 kernel 和 rootfs 的分区信息由逗号 ’,’ 隔开。

3.4. 增加修改删除 jiffs2 分区

在 spinor 的 flash 上一般用 jiffs2 作为可读写的分区的文件系统，如下定义了 customer 分区的相关配置：

```
customer$(RESOUCE)    = $(OUTPUTDIR)/customer
customer$(FSTYPE)     = jffs2
customer$(PATSIZE)    = 0x6B0000
customer$(MOUNTTG)    = /customer
customer$(MOUNTPT)    = mtd:customer
customer$(OPTIONS)    = rw
```

customer$(RESOUCE)：

表示分区的资源文件放置的位置，rootfs.mk 和 image.mk 分别会往此路径中添加必须的文件，以及把文件夹打包成 bin 文件。

customer$(FSTYPE)：

打包的文件系统类型。

customer$(PATSIZE)：

分区大小。

customer$(MOUNTTG)、customer$(MOUNTPT)、customer$(OPTIONS)：

分区在板子起来后需要 mount 的路径以及参数。rootfs.mk 中会处理这些参数。

3.4.1 修改

若要修改分区，则改动以上变量值即可。

3.4.2 增加分区

1、假设增加一个分区名为 aaa，然后配置其大小等信息：

```
aaa$(RESOUCE)    = $(OUTPUTDIR)/aaa
aaa$(FSTYPE)     = jffs2
aaa$(PATSIZE)    = 0x6B0000
aaa$(MOUNTTG)    = /aaa
aaa$(MOUNTPT)    = mtd:aaa
aaa$(OPTIONS)    = rw
```

2、分区相关的变量配置完成之后，在 mxp$(SYSTAB) = ”xxx”中指定其放置的位置。

3、在变量“IMAGE_LIST”后追加“aaa”。

4、在表示需要 mount 的节点的变量“USR_MOUNT_BLOCKS”后追加“aaa”。

3.4.3 删除分区

按照增加分区的操作，反过来执行。

3.4.4 烧录脚本

系统在 script_nor.mk 中会根据分区的配置自动生成。

3.5. 增加修改删除 squashfs 分区

Squashfs 的分区变更与 jiffs2 分区变更大同小异，主要在 xxx$(FSTYPE)上有不同，除去 rootfs 这个比较特别的分区之外，miservice 分区制作成 squashfs：

```
miservice$(RESOUCE)    = $(OUTPUTDIR)/tvconfig/config
miservice$(FSTYPE)     = squashfs
miservice$(PATSIZE)    = 0x180000
miservice$(MOUNTTG)    = /config
miservice$(MOUNTPT)    = /dev/mtdblock3
miservice$(OPTIONS)    = ro
```

需要特别注意的是，若使用者用的是 spinand 的分区配置脚本，那么分区位置放在：

cis$(SYSTAB) = xxx 中。

与 jiffs2 分区一样，系统在 script_nor.mk/script_nand.mk 中会根据分区的配置自动生成烧录脚本。

3.6. 增加修改删除 ubifs 分区

UBIFS 分区一般应用在 spinand 上作为可读写的分区文件系统。

UBIFS 的所有的分区都属于 UBI 的 mtd block 中的一个子分区。一个普通的 UBI 分区变量设置如下：

```
miservice$(RESOUCE)    = $(OUTPUTDIR)/miservice/config
miservice$(FSTYPE)     = ubifs
miservice$(PATSIZE)    = 0xA00000
miservice$(MOUNTTG)   = /config
miservice$(MOUNTPT)   = ubi0:miservice
miservice$(OPTIONS)    = rw
```

与 jiffs2 分区变更方法不同的是，ubifs 分区不需要添加 mtdpart 讯息，也就是无需在 cis$(SYSTAB) = xxx 中添加分区位置。

3.7. UBI 内部各个 volume 的 size 设置

UBI 中的各个分区严格来说叫做 volume（卷）而不是 partition（比如 RFS, APP 等，volume 比 partition 低一个级别），不管 UBI 中有多少 volume，整个 UBI 在 MTD 中都只是一个 partition。

Volume 的信息是由 UBI 管理的，和 ENV 无关，在 NAND 上有特定区域保存其信息，这是和 mtd partition 不同的地方。

3.7.1 volume size 的下限

由于 UBIFS 的需求，一个 volume（如 customer）至少需要 22 个 block，对于 2KB page size 的 NAND，就是将近 3MB，我们通常规定这个 size 不小于 4MB。这是对 volume size 下限的约束。

这个约束通常不会被违反，但某些大容量的 NAND，block size 有可能达到 2MB，这时 22 个 block 就有 44MB 了，不注意的话，就有可能违背约束了。如果不满足这一点，mkfs.ubifs 会报出 block 数不足的 error。

3.7.2 UBI 的预留量计算

对于 volume size 上限没有规定，但是各个 volume 的 size 和应该比前面算出的 UBI size 要小，小的差额也就是所谓的 UBI 分区预留量。预留量的计算公式是：

total block num * 2% + ubi block num * 1% + 4 + ubi block num * 2 / (page num of a block)

可以看到预留量是由四个部分组成的：

1）total block num * 2%：即 block 总数的 2%，为出厂坏块做的预留。2G bit NAND 为 2048 * 2% = 40；

2）ubi block num * 1%：即 block 总数的 1%，为 NAND 使用中可能新产生的坏块做的预留；如果 ubi 分区占用了 2000 个 block，则 2000 * 1% = 20；

3）“4”：UBI 用作一些特定用途，固定值；

4）ubi block num * 2 / (page num of a block)：由于 UBI 将每个 block 的前两个 page 用于存放自己的内部管理数据，因此这两个 page 也是算入预留量的。

如果 ubi 分区占用了 2000 个 block，一个 block 有 64 个 page，则这一部分空间有 2000 * 2 / 64 = 63 个 block，以一个 block 为 128KB 来计，则约为 8MB。

以上 4 个部分中，第 1 和 2 部分同 NAND 的容量成比例关系，第 3 部分和 NAND 容量无关，是固定的，第 4 部分和 UBI 分区的容量以及 NAND 结构都有关系。

需要注意的是，这里直接算出来的都是 block 数，而用户需要的是字节数，因此需要知道一个 block 的大小来转换一下。目前最常用 NAND 的 block size 是 128KB，但这个是依 NAND 型号有可能变化的，以所用 NAND spec

为准，不能教条。

常见的 1G bit，2G bit，4G bit NAND 的预留量算出来为 9MB，17MB，34MB，可以不用自己再去计算。对于最新的 MBoot，预留量如果不足，ubi create 后面的几个 volume 时会报出空间不足的 error 作为提醒。

ubi create 时如果返回 error -28， 就表示空间不足；如果返回 error -17，则表示重复创建 volume，即同一个名字的 volume 不能 ubi create 两次。如果某个 volume 因为空间不足而创建失败，则只能考虑在不同 volume 之间挪出空间或是换用更大容量的 NAND。

3.8. 特殊分区处理

KERNEL、uboot、IPL、logo 等分区属于特殊分区，这些分区没有特别的文件系统，所以无法在 Makefile 脚本中做集中处理，必须 by case 去处理。因此添加一个特殊的分区需要清楚了解如下两个步骤：

3.8.1 KERNEL 分区处理

image.mk 中需要写上分区打包的脚本命令。例如 KERNEL 分区打包命令：

```
kernel_nofsimage:
    @echo [[$@]]
    cp -rvf $($(patsubst %_nofsimage,%,$@)$(RESOUCE)) (IMAGEDIR)/$(patsubst %_nofsimage,%,$@)
```

在 script_nor.mk/ script _nand.mk 中写上分区烧录脚本生成的命令。

例如制作 KERNEL 分区烧录脚本的脚本命令：

```
kernel_$(FLASH_TYPE)__script:
    @echo "aaaabbbaaaaaaa $@"
    @echo "# <- this is for comment / total file size must be less than 4KB" >
$(SCRIPTDIR)/[[kernel.es
    @echo tftp $(TFTPDOWNLOADADDR) kernel >> $(SCRIPTDIR)/[[kernel.es
    @echo nand erase.part KERNEL >> $(SCRIPTDIR)/[[kernel.es
    @echo nand write.e $(TFTPDOWNLOADADDR) KERNEL \$${filesize} >>
$(SCRIPTDIR)/[[kernel.es
    @echo nand erase.part RECOVERY >> $(SCRIPTDIR)/[[kernel.es
    @echo nand write.e $(TFTPDOWNLOADADDR) RECOVERY \$${filesize} >>
$(SCRIPTDIR)/[[kernel.es
    @echo "% <- this is end of file symbol" >> $(SCRIPTDIR)/[[kernel.es
    @echo kernel-image done!!!
```

3.8.2 带备份的分区处理

SPINAND 的分区中 IPL、IPL_CUST、UBOOT 这些分区一般会有好几个备份，例如 IPL 在烧录时，每个 IPL 的 data 会在每个 block 开始各占一份，一个 block 有 128kb 这么大，一般 IPL 的 data 不超过 20kb，不会超过 128kb，因此假设 IPL 的数据有六个备份，那么整个 IPL 分区会有 6 个 block 这么大，那么在分区打包的时候会用到 dd 命令把 IPL 分区进行打包，使输出的 bin 文件能够与 IPL 分区完全映射，六份 ipl 的 data 各自分布在每个 block 的开头。

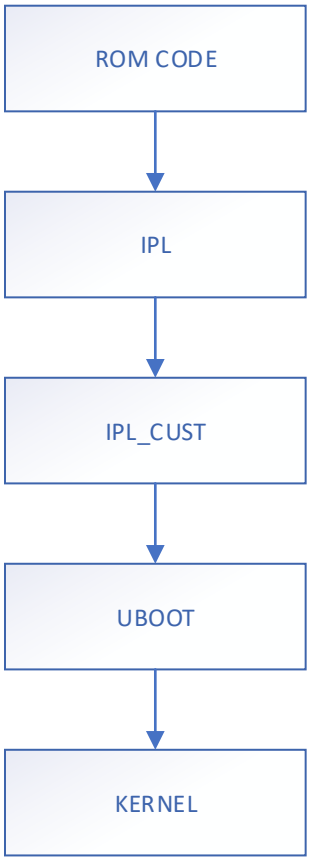
若要设置带备份的分区，可以按照如下代码设定：

```
ipl$(RESOUCE) = $(PROJ_ROOT)/board/$(CHIP)/boot/ipl/IPL.bin
ipl$(DATASIZE) = 0x20000
ipl$(COPIES) = 6
ipl$(BKCOUNT) = 5
ipl$(PATSIZE) = $(call multiplyhex, $(ipl$(COPIES)), $(ipl$(DATASIZE)))
ipl$(MTDPART) = $(ipl$(DATASIZE))@$(cis$(CISSIZE))(IPL0)$(ipl$(BKCOUNT))
```

'ipl$(COPIES)'表示 IPL 的 data 会有多少个备份。
'ipl$(DATASIZE)'表示一份 data 的大小，ipl 的 bin 文件若不超过 1 个 block 的大小则按照 1 个 block 的大小来算。
'ipl$(PATSIZE)'是 IPL 实际分区的大小，makefile 中做了一个乘法计算，就是备份的个数乘以 data 大小。
'ipl$(BKCOUNT)'是 PNI 数据结构中特有的一种数据，在系统真正跑起来时，若没有坏块的情况下，系统默认使用的是 ipl 的第一份 data，但是第一份 data 所在的 block 产生坏块，则 ROM code 会偏移一个 block 去找 ipl，这个 BKCOUT 变量就是告诉 ROM code 可以偏移 block 的最大个数。
'ipl$(MTDPART)是给'pnigenerator 用作解析分区配置的变量，它会传递给 cis$(BOOTTAB)或者 cis$(SYSTAB)，若没有 BKCOUNT 的情况下，语法格式符合 uboot mtaparts 的标准，而在没有定义变量 MTDPARTS 的情况下，uboot 在执行 write cis 命令时会解析 pni 的值，并且转换成 mtdparts 字串保存在环境变量中。

3.8.3 MXP 与 PARTINFO

ARM 启动后会先进到一段固定的 ROM CODE 中：
如图所示：



ROM code 启动后会按照固定的地址启动到 IPL，IPL 会先去找一份原始的分区配置 raw data，这部分就是 MXP 或 PARTINFO。

MXP 与 PARTINFO 这两份 RAW DATA 分别保存的是 SPINOR/SPINAND 分区的配置信息，arm 启动执行到 IPL 时由于在此时还没有 mtdpart 的概念，所以会去读 MXP 或 PARTINFO 中的内容才能找到 IPL_CUST 或者 uboot，因此这两个分区所在的地址是固定的。

3.9. SPINOR 的 MXP

SPINOR 的分区修改全部汇总到 MXP_SF.bin，这个 bin 是由一个 tool 通过传入不同的参数动态生成。在 ALKAID 的编译打包环境中 image.mk 负责分区的打包步骤，通过 shell 脚本生成 MXP_SF.bin。

如同所示：

```
/home/malloc.peng/ALKAID/project/image/makefiletools/bin/mxpgenerator "0x10000(IPL),0x10000(IPL_CUST),0x10000(MXPT),0x1F000(UBOOT),0x1000(UBOOT_ENV)" "0x200000(KERNEL),0x400000(root
lloc.peng/ALKAID/project/image/output/images/boot/MXP_SF.bin
:[156]
BOOT: 0x10000(IPL),0x10000(IPL_CUST),0x10000(MXPT),0x1F000(UBOOT),0x1000(UBOOT_ENV)
SYS: 0x200000(KERNEL),0x400000(rootfs),0x300000(miservice),0x6B0000(customer)
MXP count max 30
NOR FLASH HAS USED 0x1000000KB
        IPL: 0x00000000-0x00010000 size:64KB
    IPL_CUST: 0x00010000-0x00020000 size:64KB
       MXPT: 0x00020000-0x00030000 size:64KB
      UBOOT: 0x00030000-0x0004F000 size:124KB
   UBOOT_ENV: 0x0004F000-0x00050000 size:4KB
       BOOT: 0x00000000-0x00050000 size:320KB
     KERNEL: 0x00050000-0x00250000 size:2048KB
     rootfs: 0x00250000-0x00650000 size:4096KB
   miservice: 0x00650000-0x00950000 size:3072KB
    customer: 0x00950000-0x01000000 size:6848KB
Available MXP record count:10
[[ipl_nofsimage]]
```

工具 mxpgenerator:执行命令：

mxpgenerator "0x10000(IPL),0x10000(IPL_CUST),0x10000(MXPT),0x1F000(UBOOT),0x1000(UBOOT_ENV)" "0x200000(KERNEL),0x400000(rootfs),0x300000(miservice),0x6B0000(customer)" /home/malloc.peng/ALKAID/project/image/output/images/boot/MXP_SF.bin

mxpgenerator 有三个参数 mxpgenerator [v0] [v1] [v2]:

v0 传入的是 BOOT 部分的分区配置，v1 传入的是 sys 分区的部分，v2 则是输出文件路径。

mxpgenerator 工具的源码路径：

project/image/makefiletools/src/mxpgenerator/mxpgenerator.c

利用服务器上的 gcc 编译：

gcc mxpgenerator.c –o mxpgenerator

3.10. SPINAND 的 PARTINFO

PARTINFO.pni 通过工具 pnigenerator 生成 PARTINFO.pni，保存在 CIS 分区中，其中包括一些 spinand 基本的讯息。

```
BOOT: 0x140000(CIS),0x20000@0x140000(IPL0)5,0x20000(IPL_CUST0)2,0x20000(IPL_CUST1)2,0x4
SYS: 0x500000(KERNEL),0x500000(RECOVERY),0x600000(rootfs),-(UBI)
FLASH HAS USED 0x8000000KB
ChkSum       : 1043
SpareByteCnt : 64
PageByteCnt  : 2048
BlkPageCnt   : 64
BlkCnt       : 1024
PartCnt      : 11
UnitByteCnt  : 8
Checksum ok!!
IDX:          StartBlk:          BlkCnt:      BackupBlkCnt:        PartType:
  0:     0,(0000000000)   10,(0X00140000)     0,(0000000000)   0X0020,(CIS)
  1:    10,(0X00140000)    1,(0X00020000)     5,(0X000A0000)   0X0003,(IPL)
  2:    16,(0X00200000)    1,(0X00020000)     2,(0X00040000)   0X0001,(IPL_CUST)
  3:    19,(0X00260000)    1,(0X00020000)     2,(0X00040000)   0X0001,(IPL_CUST)
  4:    22,(0X002C0000)    2,(0X00040000)     1,(0X00020000)   0X0006,(UBOOT)
  5:    25,(0X00320000)    2,(0X00040000)     1,(0X00020000)   0X0006,(UBOOT)
  6:    28,(0X00380000)    2,(0X00040000)     0,(0000000000)   0X000D,(ENV)
  7:    30,(0X003C0000)   40,(0X00500000)     0,(0000000000)   0X0012,(KERNEL)
  8:    70,(0X008C0000)   40,(0X00500000)     0,(0000000000)   0X0009,(RECOVERY)
  9:   110,(0X00DC0000)   48,(0X00600000)     0,(0000000000)   0X000F,(ROOTFS)
 10:   158,(0X013C0000)  866,(0X06C40000)     0,(0000000000)   0X0021,(UBI)
1+0 records in
```

执行命令：

pnigenerator -s 64 -p 2048 -b 64 -k 1024 -u 8 -l 0x20000 -t "0x60000(CIS),0x20000@0x60000(IPL0),0x20000(IPL1),0x20000(IPL2),0x20000(IPL_CUST0),0x20000(IPL_CUST1),0x20000(IPL_CUST2),0x60000(UBOOT0),0x60000(UBOOT1),0x20000(ENV)" -y "0x500000(KERNEL),0x500000(RECOVERY),0x600000(rootfs),-(UBI)" -o /home/malloc.peng/ALKAID/project/image/output/images/boot/PARTINFO.pni

pnigenerator [options]

参数含义：

-s Spare byte count.
-p Page byte count.
-b Block page count.
-k Block count.
-u Unit byte count.
-l Block size.
-t BOOT Part
-y SYS Part
-r Dump pni file
-o Output file path.

pnigenerator 工具的源码路径：

project/image/makefiletools/src/pnigenerator/pnigenerator.c

利用服务器上的 gcc 编译：

gcc pnigenerator.c –o pnigenerator

PARTINFO.bin 中保存 BOOT part 的分区信息就已经足够，其他的分区可以追加到 uboot 的 mtdparts 中。

4. ONE BIN 介绍

ONE BIN 功能是基于 ALKAID 的打包脚本通过 dd 命令使 BOOT PART 分区合并成一个 bin，它可以支持 spinand 和 spinor 的两种分区打包方式。

在 project/image/boot.mk 中有 dd 合并 bin 的具体做法。

4.1. Spinor

参考脚本：

project/image/configs/i6b0/nor.squashfs.partition.boot.config

与普通的打包脚本对比：

1、 增加了需要合并的分区名。

BOOT_IMAGE_LIST = ipl ipl_cust mxp uboot

2、 合并的分区从 IMAGE_LIST 中剔除，并把合并后的分区名 ’boot’ 加上。

IMAGE_LIST = boot kernel rootfs miservice customer

3、 在 config 文件中添加 sstar 分区。

boot$(RESOUCE) = $(IMAGEDIR)/boot/sstar.bin

boot$(PATSIZE) = $(shell printf 0x%x $(shell stat -c "%s" $(sstar$(RESOUCE))))

4、 同时在 image.mk 和 script_nor.mk 中需要添加 sstar 的 image 生成脚本和 tftp 烧录脚本：

image.mk:

```
boot_nofsimage: $(BOOT_TARGET_NOFSIMAGE) $(BOOT_TARGET_FSIMAGE)
@echo [[$@]]
dd if=/dev/zero of=$(boot$(RESOUCE)) bs=1 count=0
```

script_nor.mk:

```
uboot_spi_nor__script:
    @echo "# <- this is for comment / total file size must be less than 4KB" > $(SCRIPTDIR)/[[uboot
    @echo mxp r.info UBOOT >> $(SCRIPTDIR)/[[uboot
    @echo sf probe 0 >> $(SCRIPTDIR)/[[uboot
    @echo sf erase \$${sf_part_start} \$${sf_part_size} >> $(SCRIPTDIR)/[[uboot
    @echo tftp $(TFTPDOWNLOADADDR) boot/$(notdir $(uboot$(RESOUCE))) >> $(SCRIPTDIR)/[[uboot
    @echo sf write $(TFTPDOWNLOADADDR) \$${sf_part_start} \$${filesize} >> $(SCRIPTDIR)/[[uboot
    @echo "% <- this is end of file symbol" >> $(SCRIPTDIR)/[[uboot
```

4.2. Spinand

参考脚本：

project/image/configs/i6b0/spinand.squashfs.partition.boot.config

与普通的打包脚本对比：

1、 增加了需要合并的分区名。

BOOT_IMAGE_LIST = ipl ipl_cust uboot

2、 合并的分区从 IMAGE_LIST 中剔除，并把合并后的分区名 ’boot’ 加上。

IMAGE_LIST = cis boot kernel rootfs miservice customer

3、 MTDPART 设定更新：

MTDPARTS = "mtdparts=nand0:$(boot$(MTDPART)),$(cis$(SYSTAB))"

4、 确认分区多份 copy

```
ipl_cust$(RESOUCE) = $(PROJ_ROOT)/board/$(CHIP)/boot/ipl-dualos/IPL_CUST.bin
ipl_cust$(DATASIZE) = 0x20000
ipl_cust$(COPIES) = 3
ipl_cust$(BKCOUNT) = 2
ipl_cust$(PATSIZE) = $(call multiplyhex, $(ipl_cust$(COPIES)), $(ipl_cust$(DATASIZE)))
ipl_cust$(PATCOUNT) = 2

ipl_cust$(MTDPART) =

$(ipl cust$(DATASIZE))(IPL_CUST0)$(ipl_cust$(BKCOUNT)),$(ipl_cust$(DATASIZE))(IPL_CUST1)$(ipl_

cust$(BKCOUNT))
```

请注意 PATSIZE 要根据当前的 data 大小和一个分区中所包含的 data 个数而定。

例如 ipl_cust 的 data 数量有六份，因此 COPIES = 6。 Data 各自分散在两个分区中（IPL_CUST0/IPC_CUST1），每个分区占三份 data，每一份 data 的 size 为 0x20000 因此，PATSIZE 为 0x20000 * 3 = 0x60000。，

5、 在 config 文件中添加 boot 分区。

```
boot$(RESOUCE) = $(IMAGEDIR)/boot.bin
boot$(PATSIZE) = $(shell printf 0x%x $(shell stat -c "%s" $(boot$(RESOUCE))))
boot$(MTDPART) = $(boot$(PATSIZE))@$(cis$(PATSIZE))(BOOT),0x40000(ENV)
```

6、 同时在 image.mk 和 script_nand.mk 中需要添加 boot 的 image 生成脚本和 tftp 烧录脚本：

7、 ipl/uboot 进行了合并后，对于系统的 mtd device 有减少，分区在 mount 的时候请注意/dev/mtdblock 后 block id 是否对应。

image.mk:

```
boot_nofsimage: $(BOOT_TARGET_NOFSIMAGE) $(BOOT_TARGET_FSIMAGE)
@echo [[$@]]
dd if=/dev/zero of=$(boot$(RESOUCE)) bs=1 count=0
```

script_nand.mk:

```
boot_$(FLASH_TYPE)__script:
@echo "# <- this is for comment / total file size must be less than 4KB" > $(SCRIPTDIR)/[[boot.es
@echo tftp $(TFTPDOWNLOADADDR) boot.bin >> $(SCRIPTDIR)/[[boot.es
@echo nand erase.part BOOT >> $(SCRIPTDIR)/[[boot.es
@echo nand write.e $(TFTPDOWNLOADADDR) BOOT \$${filesize} >> $(SCRIPTDIR)/[[boot.es
@echo "% <- this is end of file symbol" >> $(SCRIPTDIR)/[[boot.es
```