



ROM 分区配置指南

文档版本
发布日期

V1.1
2020-09-30

版权所有 © 紫光展锐科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

Unisoc Confidential For hiar

紫光展锐科技有限公司



前言

概述

本文初步介绍 Android 11.0、Android 10.0、Android 9.0 UNISOC ROM 空间的分配情况以及分区的方法。

读者对象


本文档主要适用于需修改分区配置相关的工程师。

缩略语

缩略语	英文全名	中文解释
MTD	Memory Technology Device	内存技术设备
UBI	Unsorted Block Images	未分类块镜像
UBIFS	Unsorted Block Image File System	无序区块镜像文件系统
EMMC	Embedded Multi Media Card	嵌入式多媒体控制器

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-08-07	第一次正式发布。

文档版本	发布日期	修改说明
V1.1	2020-09-30	增加 Android 10.0、Android 9.0 的内容描述。

关键字

ROM 分区、EMMC、UBI。

Unisoc Confidential For hiar

目 录

1 分区结构概述.....	1
1.1 ROM 分区列表	1
1.2 EMMC 分区介绍	5
2 分区操作.....	9
2.1 增加分区	9
2.2 修改分区大小	10
2.3 删除分区	11
3 分区生效流程.....	12
4 EMMC 相关代码概述	17
4.1 Device.....	17
4.2 SPL	19
4.3 Uboot.....	19
4.4 Kernel	23

Unisoc Confidential For hiar

图目录

图 3-1 下载协议握手过程	12
----------------------	----

Unisoc Confidential For hiar

表目录

表 1-1 Android 10.0、Android 9.0 ROM 分区列表.....	1
表 1-2 Android 11.0 ROM 分区列表	3

Unisoc Confidential For hiar

1 分区结构概述

本章节主要描述展锐平台 ROM 空间划分情况以及分区格式、分区大小、分区功能，并介绍了 EMMC 的分区。

1.1 ROM 分区列表

表 1-1 是展锐 Android 10.0、Android 9.0 平台 ROM 空间划分情况以及分区格式、分区大小和分区功能的初步描述。

表1-1 Android 10.0、Android 9.0 ROM 分区列表

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
1	prodnv	sparse format	size="5"	开机后系统中的 productinfo 分区，保存 adc 校准参数、eng.db 数据库。
2	Miscdata	RAW	size="1"	保存 ota、recovery 时的一些数据。
3	recovery	sparse format	size="40"	存放 recovery.img，恢复出厂设置。
4	misc	RW	size="1"	保存杂项数据，如系统关闭开关上的窗体中的设置相关。
5	trustos	RAW	size="6"	存放 tos-sign.bin
6	Trustos_bak	RAW	size="6"	trustos 的备份，防止 trustos 破坏导致系统无法开机。
7	sml	RAW	size="1"	安全世界和非安全世界切换，即 Android 和 TOS 之间的切换、源管理功能、核上下电、睡眠等。
8	Sml_bak	RAW	size="1"	sml 的备份，防止 sml 破坏导致系统无法开机。
9	uboot	RAW	size="1"	存放 ubootloader img。
10	Uboot_bak	RAW	size="1"	ubootloader img 的备份，防止 sml 破坏导致系统无法开机。
11	Uboot_log	RAW	size="4"	存放 uboot log。
12	logo	RAW	size="6"	存放开机 logo 图片。
13	fbootlogo	RAW	size="6"	存放 fastboot 模式的 logo 图片。

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
14	L_fixnv1	RAW	size="2"	存放 pubcp_nvitem.bin, 射频参数相关。
15	L_fixnv2	RAW	size="2"	fixnv1 的备份, 防止 fixnv 破坏导致系统无法开机。
16	L_runtimenv1	RAW	size="2"	运行时由 modem 生成, 是 fixnv 的一份复制。
17	L_runtimenv2	RAW	size="2"	L_runtimenv1 的备份, 起到掉电保护的作用。
18	Gpsgl	RAW	size="1"	存放 gnssmodem.bin。
19	Gpsbd	RAW	size="1"	存放 gnssbdmodem.bin。
20	Wcnmodem	RAW	size="10"	存放 cm4_v2.bin, 是 Connectivity 芯片的协议栈相关。
21	persist	sparse format	size="2"	存放 persist.img。
22	L_modem	RAW	size="25"	存放 pubcp_modem.dat, 通信协议栈相关。
23	L_deltanv	RAW	size="25"	存放 pubcp_mininfo_ncache_deltanv.bin。
24	L_gdsp	RAW	size="10"	存放 pubcp_DM_DSP.bin, 数字处理等。
25	L_ldsp	RAW	size="20"	存放 pubcp_LTEA_DSP.bin。
26	Pm_sys	RAW	size="1"	存放 cm4.bin。
27	Teecfg	RAW	size="1"	安全相关。
28	Teecfg_bak	RAW	size="1"	Teecfg 的备份分区, 起到掉电保护作用。
29	boot	sparse format	size="35"	存放 boot.img, Kernel 驱动相关。
30	dtbo	sparse format	size="8"	存放 dtbo.img。
31	super	sparse format	size="4100"	存放 SUPER.img, Android 系统相关。
32	cache	sparse format	size="150"	存放 cache.img, 在 CTS 测试, 恢复出厂设置是需要使用。
33	socko	sparse format	size="75"	存放 socko.img。
34	odmko	sparse format	size="25"	存放 odmko.img。

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
35	vbmeta	sparse format	size="1"	存放 vbmeta-sign.img。
36	Vbmeta_bak	sparse format	size="1"	vbmeta 的备份分区，起到掉电保护作用。
37	Metadata	RAW	size="16"	Erase misc section operation。
38	Sysdumpdb	RAW	size="10"	Erase misc section operation。
39	Vbmeta_system	sparse format	size="1"	存放 vbmeta_system.img。
40	Vbmeta_vendor	sparse format	size="1"	存放 vbmeta_vendor.img。
41	userdata	sparse format	剩余大小	存放 userdata.img，包含用户的数据。

表 1-2 是展锐 Android 11.0 平台 ROM 空间划分情况以及分区格式、分区大小和分区功能的初步描述。

表1-2 Android 11.0 ROM 分区列表

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
1	prodnv	sparse format	size="10"	开机后系统中的 productinfo 分区，保存 adc 校准参数、eng.db 数据库。
2	Miscdata	RAW	size="1"	保存 ota、recovery 时的数据。
3	misc	RAW	size="1"	保存杂项数据，如系统关闭开关上的窗体中的设置相关。
4	Trustos_a	RAW	size="6"	存放 tos-sign.bin。
5	Trustos_b	RAW	size="6"	trustos 的 b 分区，防止 trustos 破坏导致系统无法开机。
6	Sml_a	RAW	size="1"	安全世界和非安全世界切换，即 Android 和 TOS 之间的切换、源管理功能、核上下电，睡眠等。
7	Sml_b	RAW	size="1"	sml 的 b 分区，防止 sml 破坏导致系统无法开机。
8	Uboot_a	RAW	size="1"	存放 ubootloader img。
9	Uboot_b	RAW	size="1"	ubootloader img 的 b 分区，防止 uboot 破坏导致系统无法开机。
10	Uboot_log	RAW	size="4"	存放 uboot log。
11	logo	RAW	size="8"	存放开机 logo 图片。

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
12	fbootlogo	RAW	size="8"	存放 fastboot 模式的 logo 图片。
13	L_fixnv1_a	RAW	size="2"	存放 pubcp_nvitem.bin, 射频参数相关。
14	L_fixnv1_b	RAW	size="2"	L_fixnv1_a 的 b 分区。
15	L_fixnv2_a	RAW	size="2"	fixnv1 的备份, 防止 fixnv 破坏导致系统无法开机。
16	L_fixnv2_b	RAW	size="2"	L_fixnv2_a 的 b 分区。
17	L_runtimenv1	RAW	size="2"	运行时由 modem 生成, 是 fixnv 的一份复制。
18	L_runtimenv2	RAW	size="2"	L_runtimenv1 的备份, 起到掉电保护的作用。
19	Gnssmodem_a	RAW	size="1"	存放 gnssmodem.bin。
20	Gnssmodem_b	RAW	size="1"	存放 gnssmodem.bin 的 B 分区。
21	Wcnmodem_a	RAW	size="10"	存放 cm4_v2.bin, 是 Connectivity 芯片的协议栈相关。
22	Wcnmodem_b	RAW	size="10"	存放 cm4_v2.bin 的 B 分区。
23	persist	sparse format	size="2"	存放 persist.img。
24	L_modem_a	RAW	size="25"	存放 pubcp_modem.dat, 通信协议栈相关。
25	L_modem_b	RAW	size="25"	存放 pubcp_modem.dat 的 B 分区。
26	L_deltanv_a	RAW	size="25"	存放 pubcp_mininfo_ncache_deltanv.bin。
27	L_deltanv_b	RAW	size="25"	存放 pubcp_mininfo_ncache_deltanv.bin 的 B 分区。
28	L_gdsp_a	RAW	size="10"	存放 pubcp_DM_DSP.bin, 数字处理等。
29	L_gdsp_b	RAW	size="10"	存放 pubcp_DM_DSP.bin 的 B 分区。
30	L_ldsp_a	RAW	size="20"	存放 pubcp_LTEA_DSP.bin。
31	L_ldsp_b	RAW	size="20"	存放 pubcp_LTEA_DSP.bin 的 B 分区。
32	Pm_sys_a	RAW	size="1"	存放 cm4.bin。
33	Pm_sys_b	RAW	size="1"	存放 cm4.bin 的 B 分区。
34	Teecfg_a	RAW	size="1"	安全相关。
35	Teecfg_b	RAW	size="1"	Teecfg 的 B 分区, 起到掉电保护作用。

序号	分区名	分区格式	分区大小 (MBytes)	分区功能描述
36	Boot_a	sparse format	size="64"	存放 boot.img, kernel 驱动相关。
37	Boot_b	sparse format	size="64"	存放 boot.img 的 B 分区。
38	Dtbo_a	sparse format	size="8"	存放 dtbo.img。
39	Dtbo_b	sparse format	size="8"	存放 dtbo.img 的 B 分区。
40	super	sparse format	size="4100"	存放 SUPER.img。
41	Socko_a	sparse format	size="75"	存放 socko.img。
42	Socko_b	sparse format	size="75"	存放 socko.img 的 B 分区。
43	Odmko_a	sparse format	size="25"	存放 odmko.img。
44	Odmko_b	sparse format	size="25"	存放 odmko.img 的 B 分区。
45	Vbmeta_a	sparse format	size="1"	存放 vbmeta-sign.img。
46	Vbmeta_b	sparse format	size="1"	存放 vbmeta-sign.img 的 B 分区。
47	Metadata	RAW	size="16"	Erase misc section operation。
48	Sysdumpdb	RAW	size="10"	Erase misc section operation。
49	Vbmeta_system_a	sparse format	size="1"	存放 vbmeta_system.img。
50	Vbmeta_system_b	sparse format	size="1"	存放 vbmeta_system.img 的 B 分区。
51	Vbmeta_vendor_a	sparse format	size="1"	存放 vbmeta_vendor.img。
52	Vbmeta_vendor_b	sparse format	size="1"	存放 vbmeta_vendor.img 的 B 分区。
53	userdata	sparse format	剩余大小	存放 userdata.img, 包含用户的数据。

1.2 EMMC 分区介绍

在 EMMC 方案中, 可以通过查看对应的 pac 包中的 Productname.xml 文件看到分区的详细信息。展锐 Android 11.0 中, 采用 v-ab 的格式进行分区, 而 Android 10.0、Android 9.0 中, 并未采用 v-ab 得格式进行分区。

Android 10.0、Android 9.0 分区具体如下:

```
<Partitions>
  <!-- size unit is MBytes -->
  <Partition id="prodnv" size="5"/>
  <Partition id="miscdata" size="1"/>
  <Partition id="recovery" size="40"/>
```

```
<Partition id="misc" size="1"/>
<Partition id="trustos" size="6"/>
<Partition id="trustos_bak" size="6"/>
<Partition id="sml" size="1"/>
<Partition id="sml_bak" size="1"/>
<Partition id="uboot" size="1"/>
<Partition id="uboot_bak" size="1"/>
<Partition id="uboot_log" size="4"/>
<Partition id="logo" size="6"/>
<Partition id="fbootlogo" size="6"/>
<Partition id="l_fixnv1" size="2"/>
<Partition id="l_fixnv2" size="2"/>
<Partition id="l_runtimenv1" size="2"/>
<Partition id="l_runtimenv2" size="2"/>
<Partition id="gpsgl" size="1"/>
<Partition id="gpsbd" size="1"/>
<Partition id="wcnmodem" size="10"/>
<Partition id="persist" size="2"/>
<Partition id="l_modem" size="25"/>
<Partition id="l_deltanv" size="1"/>
<Partition id="l_gdsp" size="10"/>
<Partition id="l_ldsp" size="20"/>
<Partition id="pm_sys" size="1"/>
<Partition id="teecfg" size="1"/>
<Partition id="teecfg_bak" size="1"/>
<Partition id="boot" size="35"/>
    <Partition id="dtbo" size="8"/>
<Partition id="super" size="4100"/>
<Partition id="cache" size="150"/>
<Partition id="socko" size="75"/>
<Partition id="odmko" size="25"/>
<Partition id="vbmeta" size="1"/>
<Partition id="vbmeta_bak" size="1"/>
<Partition id="metadata" size="16"/>
<Partition id="sysdumpdb" size="10"/>
<Partition id="vbmeta_system" size="1"/>
<Partition id="vbmeta_vendor" size="1"/>
<Partition id="userdata" size="0xFFFFFFFF"/>
</Partitions>
```

Android 11.0 分区具体如下：

```
<Partitions>
```

```
<!-- size unit is MBytes -->
<Partition id="prodnv" size="10"/>
<Partition id="miscdata" size="1"/>
<Partition id="misc" size="1"/>
<Partition id="trustos_a" size="6"/>
<Partition id="trustos_b" size="6"/>
<Partition id="sml_a" size="1"/>
<Partition id="sml_b" size="1"/>
<Partition id="teecfg_a" size="1"/>
<Partition id="teecfg_b" size="1"/>
<Partition id="uboot_a" size="1"/>
<Partition id="uboot_b" size="1"/>
<Partition id="uboot_log" size="4"/>
<Partition id="logo" size="8"/>
<Partition id="fbootlogo" size="8"/>
<Partition id="l_fixnv1" size="2"/>
<Partition id="l_fixnv2" size="2"/>
<Partition id="l_runtimenv1" size="2"/>
<Partition id="l_runtimenv2" size="2"/>
<Partition id="gnssmodem_a" size="1"/>
<Partition id="gnssmodem_b" size="1"/>
<Partition id="wcnmodem_a" size="10"/>
<Partition id="wcnmodem_b" size="10"/>
<Partition id="persist" size="2"/>
<Partition id="l_modem_a" size="25"/>
<Partition id="l_modem_b" size="25"/>
<Partition id="l_deltanv_a" size="1"/>
<Partition id="l_deltanv_b" size="1"/>
<Partition id="l_gdsp_a" size="10"/>
<Partition id="l_gdsp_b" size="10"/>
<Partition id="l_ldsp_a" size="20"/>
<Partition id="l_ldsp_b" size="20"/>
<Partition id="l_agdsp_a" size="6"/>
<Partition id="l_agdsp_b" size="6"/>
<Partition id="l_cdsp_a" size="1"/>
<Partition id="l_cdsp_b" size="1"/>
<Partition id="pm_sys_a" size="1"/>
<Partition id="pm_sys_b" size="1"/>
<Partition id="boot_a" size="64"/>
<Partition id="boot_b" size="64"/>
<Partition id="vendor_boot_a" size="100"/>
```

```
<Partition id="vendor_boot_b" size="100"/>
<Partition id="dtb_a" size="8"/>
<Partition id="dtb_b" size="8"/>
<Partition id="dtbo_a" size="8"/>
<Partition id="dtbo_b" size="8"/>
<Partition id="super" size="4100"/>
<Partition id="socko_a" size="75"/>
<Partition id="socko_b" size="75"/>
<Partition id="odmko_a" size="25"/>
<Partition id="odmko_b" size="25"/>
<Partition id="vbmeta_a" size="1"/>
<Partition id="vbmeta_b" size="1"/>
<Partition id="metadata" size="16"/>
<Partition id="sysdumpdb" size="10"/>
<Partition id="vbmeta_system_a" size="1"/>
<Partition id="vbmeta_system_b" size="1"/>
<Partition id="vbmeta_vendor_a" size="1"/>
<Partition id="vbmeta_vendor_b" size="1"/>
<Partition id="vbmeta_system_ext_a" size="1"/>
<Partition id="vbmeta_system_ext_b" size="1"/>
<Partition id="vbmeta_product_a" size="1"/>
<Partition id="vbmeta_product_b" size="1"/>
<Partition id="userdata" size="0xFFFFFFFF"/>
</Partitions>
```

2 分区操作

2.1 增加分区

增加一个分区请参考如下方法进行，只需要修改对应工程的 xml 文件，就可以完成分区的增加工作。

下边以增加一个 fast_logo 分区举例，步骤如下：

步骤 1 增加一行，代码如下：

```
<Partition id="fast_logo" size="1"/>
```

说明

对于我们增加的分区 size 一般是依照实际要写入的 bin 文件的大小来定义，大小以 M 为单位。Size 设置的过大比较浪费空间，设置太小下载时就会报错。

步骤 2 增加 file 定义，代码如下：

```
<File>
  <ID>Fast_Logo</ID>
<IDAlias> Fast_Logo </IDAlias>
<Type>CODE2</Type> //还有一种Type EraseFlash2 Type 决定是否会有文件写入到给分区，如果是 CODE2 就可以写入数据到该分区，如果是 EraseFlash2 就对该分区只进行格式化操作。
  <Block id="fast_logo">
    <Base>0x0</Base>
    <Size>0x0</Size>
  </Block>
  <Flag>1</Flag> //如果不置1是无法选择文件写入的。
  <CheckFlag>0</CheckFlag>
  <Description>fast logo image file</Description> //这个描述可以自定义
</File>
```

步骤 3 修改好对应的 xml 文件之后，重新制作 pac 包使用工具，加载新生成的 pac 包或者重新在工具中选择对应的 Product，就可以看到增加的这个分区了。

----结束

说明

增加一个 sd 分区和上边的修改方案是一样的，不过需要将 Type 修改成 EraseFlash2。这样 EMMC 方案的 fdl2 如果检测到分区 id 是 sd 会将其格式化成 FAT 文件系统，写入 FAT 文件系统信息。

2.2 修改分区大小

修改分区大小是增加一个分区的一部分，新增加一个分区也要配置分区的大小。方法如下：

修改 system 分区

修改工具工程配置 project.xml 文件中对应的 system 分区的大小，如下：

`<Partition id="super" size="4100"/>`：这里配置的是物理的分区大小，即system分区大小，这个值必须大于等于 Boardconfig.mk中配置的文件系统镜像大小。

`<Partition id="userdata" size="0xFFFFFFFF"/>`：data分区是根据flash 总大小减去其他空间总大小的差值，因此这里不需要修改。

修改好 xml 文件之后请重新制作 pac 包，以确保修改成功。

修改 cache 分区

修改工具工程配置 project.xml 文件中对应的 cache 分区的大小，如下：

`<Partition id="cache" size="150"/>`：这里配置的是物理的分区大小，请修改成需要配置的大小,物理分区必须大于等于 Boardconfig.mk中配置的文件系统镜像大小。

修改好 xml 文件之后请重新制作 pac 包，以确保修改成功。

修改 prodnv 分区

修改工具工程配置 project.xml 文件中对应的 prodnv 分区的大小，如下：

`<Partition id="prodnv" size="5"/>`：这里配置的是物理分区大小，请修改成需要配置的大小，物理分区必须大于等于 Boardconfig.mk中配置的文件系统镜像大小。

修改好这个之后还需要修改备份 prodnv 的大小，修改如下：

```
<File backup="1">
  <ID>ProdNV</ID>
  <IDAlias>ProdNV</IDAlias>
  <Type>CODE2</Type>
  <Block id="prodnv">
    <Base>0x0</Base>
    <Size>0x500000</Size>：这里是备份prodnv的大小5M，请修改成需要配置的大小。
  </Block>
  <Flag>1</Flag>
  <CheckFlag>0</CheckFlag>
  <Description>Download prodnv section operation</Description>
</File>
```

修改好 xml 文件之后请重新制作 pac 包，以确保修改成功。

修改 dtbo 分区

修改工具工程配置 project.xml 文件中对应的 dtbo 分区的大小，如下：

`<Partition id="dtbo" size="8"/>`：这里配置的是物理分区大小，请修改成需要配置的大小，物理分区必须大于等于 Boardconfig.mk 中配置的文件系统镜像大小。

修改其他分区大小

只需要修改工具中 project.xml 文件即可，修改好 xml 文件之后请重新制作 pac 包，以确保修改成功。

2.3 删除分区

删除一个分区的方法和增加一个分区就是一个对立的过程，进行相反的操作就可以完成这个工作，即删除 xml 文件中的 id 项和对应的 file 项就可以了，修改好了之后同样需要重新制作 pac 包，使用工具加载新生成的 pac 或者在工具中重新选择修改好的 Product，即可检查修改是否生效。

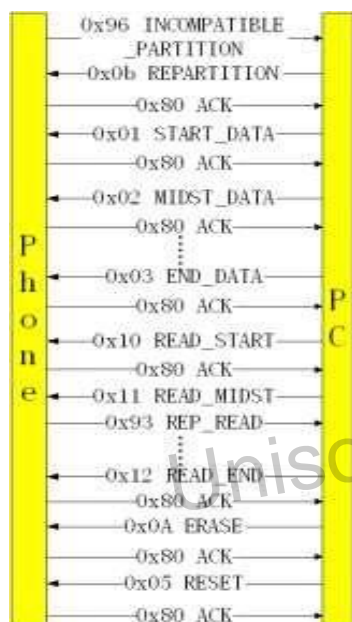
Unisoc Confidential For hiar

3 分区生效流程

本章节介绍从软件流程上增删改物理分区之后，是如何生效的，即软件上如何将新的分区信息写入到手机内部的存储器件上。

整个下载过程采用的是简单的问答式协议，如图 3-1 所示，由 PC 主控，相当于 server，手机则相当于 client。

图3-1 下载协议握手过程



从图 3-1 协议握手过程可以看到，下载过程第一步就是重分区。在下载 pac 时，fdl2 会向 PC 下载工具发送分区不一致的信号（BSL_INCOMPATIBLE_PARTITION），然后下载工具会下发重分区命令（BSL_REPARTITION），fdl2 收到重分区命令后，会执行相应的注册函数，解析 pac 中的 xml 分区文件，进行重新分区工作。

以下以 uboot15 代码为例说明：

do_download 函数

/u-boot15/common/cmd_download.c 文件中 do_download 函数定义如下：

```

int do_download(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    /*省略部分代码*/
}
    
```

```
/* register all cmd process functions */
dl_cmd_register(BSL_CMD_START_DATA, dl_cmd_write_start);
dl_cmd_register(BSL_CMD_MIDST_DATA, dl_cmd_write_midst);
dl_cmd_register(BSL_CMD_END_DATA, dl_cmd_write_end);
dl_cmd_register(BSL_CMD_READ_FLASH_START, dl_cmd_read_start);
dl_cmd_register(BSL_CMD_READ_FLASH_MIDST, dl_cmd_read_midst);
dl_cmd_register(BSL_CMD_READ_FLASH_END, dl_cmd_read_end);
dl_cmd_register(BSL_ERASE_FLASH, dl_cmd_erase);
dl_cmd_register(BSL_REPARTITION, dl_cmd_repartition);
dl_cmd_register(BSL_CMD_NORMAL_RESET, dl_cmd_reboot);
dl_cmd_register(BSL_CMD_POWER_DOWN_TYPE, dl_powerdown_device);
//dl_cmd_register(BSL_CMD_READ_CHIP_TYPE, dl_cmd_mcu_read_chiptype);
dl_cmd_register(BSL_CMD_READ_MCP_TYPE, dl_cmd_read_mcptype);

/*省略部分代码*/

usb_init(0);

/* uart download doesn't support disable hdlc, so need check it */
if (FDL_get_DisableHDLCL() == NULL)
    dl_send_ack (BSL_INCOMPATIBLE_PARTITION);
else {
    if (FDL_get_SupportRawDataProc())
        Da_Info.dwVersion = 4;
    else
        Da_Info.dwVersion = 2;
    Da_Info.bDisableHDLCL = 1;
}

.....

offset += dl_da_tlv(ack_packet.body.content + offset,
    (uint16_t)E_RSA, 260, type_encrypt_data);
#endif

ack_packet.body.size = offset;
#else

memcpy((uchar *)ack_packet.body.content, (uchar *)&Da_Info, sizeof(Da_Info));
ack_packet.body.size = sizeof(Da_Info);
#endif

dl_send_packet(&ack_packet);
}
```

```
#ifdef CONFIG_HANDSHAKE_DATA_CHECK
    dl_handshake_check_handler();
#endif

/* enter command handler */
dl_cmd_handler();
return 0;
}
```

在 `do_download` 函数中进行完一系列命令的注册后，`uboot` 会向下载工具发送 `BSL_INCOMPATIBLE_PARTITION` 的信号，之后进入 `dl_cmd_handler` 函数，等待工具发的 `command`。工具在收到手机发送的 `BSL_INCOMPATIBLE_PARTITION` 信号后，会向手机端发送 `BSL_REPARTITION` 的 `command`。手机侧执行 `dl_cmd_repartition` 函数，进行重分区工作。

dl_cmd_repartition 函数

/u-boot15/common/dloader/dl_cmd_proc.c 文件中 `dl_cmd_repartition` 函数定义如下：

```
int dl_cmd_repartition(dl_packet_t *pakcet, void *arg)
{
    .....
    /*接收工具发送的新的XML物理分区信息*/
    _parse_repartition_header(raw_data, &rp_info, &p_part_list);

    if (0 == rp_info.version) {
        part_cell_length = REPARTITION_UNIT_LENGTH;
    } else {
        part_cell_length = REPARTITION_UNIT_LENGTH_V1;
        size = rp_info.table_size;
    }

    if (0 != (size % part_cell_length)) {
        printf("%s:recvd packet size(%d) error \n", __FUNCTION__, size);
        dl_send_ack(BSL_INCOMPATIBLE_PARTITION);
        return 0;
    }

    total_partition_num = size / part_cell_length;
    debugf("Partition total num:%d \n", total_partition_num);
    op_res = dl_repartition(p_part_list, total_partition_num, rp_info.version, rp_info.unit);
    _send_reply(op_res);

    /* in download mode, write log must be after repartiton action */
}
```

```
reinit_write_log();  
  
return 0;  
}
```

从上面的函数可知，在接收到 pac 包中 xml 分区信息，并做了一些解析后，执行 dl_repartition 函数，进行分区。

dl_repartition 函数

/u-boot15/common/dloader/dl_nand_operate.c 文件中 dl_repartition 函数定义如下：

```
OPERATE_STATUS dl_repartition(uchar * partition_cfg, uint16_t total_partition_num, uchar version, uchar size_unit)  
{  
    ...  
    res = _parser_repartition_cfg(partition_info, partition_cfg, total_partition_num, version, size_unit); if (res < 0) {  
        free(partition_info);  
        return OPERATE_SYSTEM_ERROR;  
    } res = common_repartition(partition_info, (int)total_partition_num);    free(partition_info);    if (0 == res)    return  
    OPERATE_SUCCESS;  
    else  
        return OPERATE_SYSTEM_ERROR;  
}
```

其中：

- _parser_repartition_cfg: 解析工具传输的分区信息。
- common_repartition: 执行分区动作函数。

common_repartition 函数

/u-boot15/common/sprd_common_rw.c 文件中 common_repartition 函数定义如下：

```
common_repartition(disk_partition_t *partitions, int parts_count)  
{    ...  
    memset(&local_part_info, 0, sizeof(disk_partition_t));  
    dev_desc = get_dev_hwpart("mmc", 0, PARTITION_USER);  
    if (NULL == dev_desc) {  
        errorf("get mmc device hardware part(%d) fail\n", PARTITION_USER);  
        return -1;  
    }  
    while (counter < 3) {  
        ret = gpt_restore(dev_desc, SPRD_DISK_GUID_STR, partitions, parts_count);  
        if (0 == ret)  
            break;  
        counter++;  
    }  
}
```

```
if (3 == counter) {  
    return -1;  
} else {  
    init_part(dev_desc);  
    return 0;  
}
```

由上可知，执行 `gpt_restore`，将新的分区信息写入。这里在写入时会判断写入是否成功，如果不成功会重复尝试三次。

Unisoc Confidential For hiar

4 EMMC 相关代码概述

4.1 Device

工程的 `device\sprd\mpool\module\partition` 中，与存储相关的配置主要包括以下几个：

- xml 文件作用：负责 ROM 分区
- fstab 文件作用：文件系统挂载

EMMC 存储下 fstab 文件内容如下：

```
system /system ext4 ro,barrier=1 wait,logical,first_stage_mount,avb_keys=/avb/q-gsi.avbpubkey:/avb/r-gsi.avbpubkey:/avb/s-gsi.avbpubkey,slotselect
system_ext /system_ext ext4 ro,barrier=1 wait,logical,first_stage_mount,slotselect
vendor /vendor ext4 ro,barrier=1 wait,logical,first_stage_mount,slotselect
product /product ext4 ro,barrier=1 wait,logical,first_stage_mount,slotselect
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065,inline_xattr,inline_data latemount,wait,fileencryption=aes-256-
xts:aes-256-cts:v2,keydirectory=/metadata/vold/metadata_encryption,check,reservedsize=128M,checkpoint=fs,formattable
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/metadata /metadata ext4 nodev,noatime,nosuid,errors=panic
wait,formattable,first_stage_mount,check
/devices/platform/soc/soc:aon/5ff00000.usb/musb-hdrc.*.auto/usb* auto vfat defaults voldmanaged=usbdisk:auto
/devices/platform/soc/soc:ap-apb/71100000.sdio/mmc_host/mmc1/mmc1:*/block/mmcblk1 auto vfat defaults
voldmanaged=sdcard0:auto,noemulatedsd,encryptable=footer
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/prodnv /mnt/vendor ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,check
/dev/block/platform/soc/soc:ap-ahb/71400000.sdio/by-name/cache /cache ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,check
# Should after mount prodnv for prodnv wholly occupying /mnt/vendor
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/socko /mnt/vendor/socko ext4
ro,noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,avb=socko,check,slotselect
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/odmko /mnt/vendor/odmko ext4
ro,noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,avb=odmko,check,slotselect
/dev/block/platform/soc/soc:ap-apb/71400000.sdio/by-name/misc /misc emmc defaults defaults
#/dev/block/memdisk.0 /system ext4 rw,barrier=1 wait
#/dev/block/memdisk.1 /data ext4 noatime,nosuid,nodev,noauto_da_alloc,journal_async_commit,errors=panic wait
```

`first.mk` && `main.mk`：该文件控制了与文件系统相关的镜像文件的格式和大小配置。例如：`system.img`、`cache.img`、`userdata.img`、`prodnv.img` 等。

EMMC 主要配置相关镜像的 size 和文件系统格式：

```
# default value is 512M, using resize to adapter real size
BOARD_USERDATAIMAGE_PARTITION_SIZE ?= 536870912
```



```
BOARD_SUPER_PARTITION_SIZE ?= 4299161600
BOARD_GROUP_UNISOC_SIZE ?= 4299161600
# ext4 partition layout
#BOARD_VENDORIMAGE_PARTITION_SIZE ?= 419430400
BOARD_BOOTIMAGE_PARTITION_SIZE ?= 67108864
BOARD_RECOVERYIMAGE_PARTITION_SIZE ?= 41943040
#BOARD_SYSTEMIMAGE_PARTITION_SIZE ?= 3145728000
BOARD_CACHEIMAGE_PARTITION_SIZE ?= 104857600
BOARD_PRODNVIMAGE_PARTITION_SIZE ?= 10485760
BOARD_DTBOIMG_PARTITION_SIZE ?= 8388608
BOARD_DTBIMG_PARTITION_SIZE ?= 8388608
BOARD_FLASH_BLOCK_SIZE ?= 4096
BOARD_PERSISTIMAGE_PARTITION_SIZE ?= 2097152
#BOARD_PRODUCTIMAGE_PARTITION_SIZE ?= 419430400
BOARD_SOCKOIMAGE_PARTITION_SIZE ?= 78643200 # 75M
BOARD_ODMKOIMAGE_PARTITION_SIZE ?= 26214400 # 25M
BOARD_VENDOR_BOOTIMAGE_PARTITION_SIZE ?= 104857600 # 100M
TARGET_SYSTEMIMAGES_SPARSE_EXT_DISABLED := true
TARGET_USERIMAGES_SPARSE_EXT_DISABLED := false
BOARD_PERSISTIMAGE_PARTITION_SIZE := 2097152
TARGET_PRODNVIMAGES_SPARSE_EXT_DISABLED := true
TARGET_CACHEIMAGES_SPARSE_EXT_DISABLED := false
USE_SPRD_SENSOR_HUB := true
#BOARD_PRODUCTIMAGE_PARTITION_SIZE := 419430400
BOARD_PRODUCTIMAGE_FILE_SYSTEM_TYPE := ext4
TARGET_COPY_OUT_PRODUCT=product

BOARD_SYSTEM_EXTIMAGE_FILE_SYSTEM_TYPE ?= ext4
BOARD_PRODUCTIMAGE_FILE_SYSTEM_TYPE ?= ext4
#creates the metadata directory
BOARD_USES_METADATA_PARTITION ?= true
BOARD_BUILD_SYSTEM_ROOT_IMAGE ?= false
TARGET_USES_MKE2FS ?= true
#enable F2FS for userdata.img
BOARD_USERDATAIMAGE_FILE_SYSTEM_TYPE ?= f2fs
TARGET_PRODNVIMAGES_SPARSE_EXT_DISABLED ?= true
TARGET_CACHEIMAGES_SPARSE_EXT_DISABLED ?= false
BOARD_CACHEIMAGE_FILE_SYSTEM_TYPE ?= ext4
BOARD_PRODNVIMAGE_FILE_SYSTEM_TYPE ?= ext4
TARGET_SYSTEMIMAGES_SPARSE_EXT_DISABLED ?= true
TARGET_USERIMAGES_SPARSE_EXT_DISABLED ?= false
```

```
BOARD_VENDORIMAGE_FILE_SYSTEM_TYPE ?= ext4
TARGET_USERIMAGES_USE_EXT4 ?= true
```

4.2 SPL

EMMC 代码配置：EMMC 工程使用 `emmc_boot.c`，读取 EMMC 中的 `uboot` 分区中的镜像并 `load` 到内存中相应的地址中。代码如下：

```
Emmc_Read(PARTITION_BOOT2, 0, CONFIG_SYS_EMMC_U_BOOT_SECTOR_NUM, (uint8 *)
CONFIG_SYS_NAND_U_BOOT_DST);
```

4.3 Uboot

`fdl2` 承担了芯片的几乎所有的下载任务，主要包括存储芯片包括 EMMC 初始化，文件分区系统的初始化和镜像的写入任务，本节重点介绍分区系统和镜像的下载。

EMMC 方案：使用 GPT 分区系统。GPT 分区的重分区在 `/u-boot15/common/sprd_common_rw.c` 中，关于 GPT 分区重分区代码如下：

```
int common_repartition(disk_partition_t *partitions, int parts_count)
{
    block_dev_desc_t *dev_desc;
    int dev_id = 0;
    char *ifname;
    int counter = 0;
    int ret = 0;

    ifname = block_dev_get_name();
    dev_id = get_devnum_hwpart(ifname, USER_PART);
    dev_desc = get_dev_hwpart(ifname, dev_id, USER_PART);

    memset(&local_part_info, 0, sizeof(disk_partition_t));
    while (counter < 3) {
        ret = gpt_restore(dev_desc, SPRD_DISK_GUID_STR, partitions, parts_count);
        if (0 == ret)
            break;
        counter++;
    }

    if (3 == counter) {
        return -1;
    } else {
        init_part(dev_desc);
    }
}
```

```
        return 0;
    }
}
```

其中关键函数 `gpt_restore`，子函数 `gpt_fill_header`、`gpt_fill_pte` 和 `write_gpt_table` 都在目录 `u-boot15/disk/part_efi.c` 中。

函数 `write_gpt_table` 为重分区的关键函数，各个分区名称和起始地址信息的 GPT 分区头写入 mmc 的 user 分区的前面几个 block 中。

```
int write_gpt_table(block_dev_desc_t *dev_desc, gpt_header *gpt_h, gpt_entry *gpt_e)
{
    const int pte_blk_cnt = BLOCK_CNT(((gpt_h->num_partition_entries
                                         * sizeof(gpt_entry)), dev_desc);

    u32 calc_crc32;

    debug("max lba: %x\n", (u32) dev_desc->lba);
    /* Setup the Protective MBR */
    if (set_protective_mbr(dev_desc) < 0)
        goto err;

    /* Generate CRC for the Primary GPT Header */
    calc_crc32 = efi_crc32((const unsigned char *)gpt_e,
                           le32_to_cpu(gpt_h->num_partition_entries) *
                           le32_to_cpu(gpt_h->sizeof_partition_entry));
    gpt_h->partition_entry_array_crc32 = cpu_to_le32(calc_crc32);

    calc_crc32 = efi_crc32((const unsigned char *)gpt_h,
                           le32_to_cpu(gpt_h->header_size));
    gpt_h->header_crc32 = cpu_to_le32(calc_crc32);

    /* Write the First GPT to the block right after the Legacy MBR */
    if (dev_desc->block_write(dev_desc->dev, 1, 1, gpt_h) != 1)
        goto err;

    if (dev_desc->block_write(dev_desc->dev, 2, pte_blk_cnt, gpt_e)
        != pte_blk_cnt)
        goto err;

    prepare_backup_gpt_header(gpt_h);

    if (dev_desc->block_write(dev_desc->dev,
                             (lbaint_t)le64_to_cpu(gpt_h->last_usable_lba)
                             + 1,
```

```

        pte_blk_cnt, gpt_e) != pte_blk_cnt)

    goto err;

    if (dev_desc->block_write(dev_desc->dev,
        (lbaint_t)le64_to_cpu(gpt_h->my_lba), 1,
        gpt_h) != 1)

        goto err;

    debug("GPT successfully written to block device!\n");
    return 0;

err:
    printf("*** Can't write to device %d **\n", dev_desc->dev);
    return -1;
}

```

分区寻找、镜像写入在建立好 GPT 分区后，接下来的工作就是：将接收上位机发送的各个分区的镜像写入到对应的分区中。

具体代码实现在路径 u-boot15/common/sprd_common_rw.c 下的函数 common_raw_write 中：

```

dev_desc = get_dev_hwpart("mmc", 0, PARTITION_USER);
get_partition_info_by_name(dev_desc, part_name, &part_info);

```

详细过程函数：遍历 GPT 分区表，寻找对应的分区名和起始地址，并返回。

```

int get_partition_info_by_name_efi(block_dev_desc_t *dev_desc, uchar *partition_name, disk_partition_t *info)
{
    ALLOC_CACHE_ALIGN_BUFFER_PAD(gpt_header, gpt_head, 1, dev_desc->blksz);
    gpt_entry *pgpt_pte = NULL;
    int ret=-1;
    unsigned int i,j,partition_nums=0;
    uchar disk_partition[PARTNAME_SZ];

    if (!dev_desc || !info || !partition_name) {
        printf("%s: Invalid Argument(s)\n", __func__);
        return -1;
    }

    /* This function validates AND fills in the GPT header and PTE */
    if (is_gpt_valid(dev_desc, GPT_PRIMARY_PARTITION_TABLE_LBA,
        gpt_head, &pgpt_pte) != 1) {
        printf("%s: *** ERROR: Invalid Main GPT ***\n", __func__);
        if (is_gpt_valid(dev_desc, (dev_desc->lba - 1),
            gpt_head, &pgpt_pte) != 1) {

```

```

    printf("%s: *** ERROR: Invalid alternate GPT ***\n",
           __func__);
    return -1;
} else {
    /* Rewrite Primary GPT partition table with the value of Backup GPT */
    write_primary_gpt_table(dev_desc, gpt_head, pgpt_pte);
}
}

/*Get the partition info*/
partition_nums=le32_to_cpu(gpt_head->num_partition_entries);
for(i=0;i<partition_nums;i++)
{
    for(j=0;j<PARTNAME_SZ;j++)
    {
        disk_partition[j]=pgpt_pte[i].partition_name[j]&0xFF;
    }
    if(0==strcmp(disk_partition,partition_name))
    {
        /* The ulong casting limits the maximum disk size to 2 TB */
        info->start = (ulong)le64_to_cpu(pgpt_pte[i].starting_lba);
        /* The ending LBA is inclusive, to calculate size, add 1 to it */
        info->size = (ulong)le64_to_cpu((pgpt_pte[i].ending_lba) + 1)
            - info->start;
        info->blksz = dev_desc->blksz;

        sprintf((char *)info->name, "%s", print_efiname(&((pgpt_pte)[i])));
        sprintf((char *)info->type, "U-Boot");
        /*
        debug("%s: start 0x" LBAF " , size 0x" LBAF " , name %s\n", __func__,
              info->start, info->size, info->name);
        */
        ret =0;
        break;
    }
}

/* Remember to free pte */
if(pgpt_pte!=NULL){
    free(pgpt_pte);
}

return ret;
}

```

4.4 Kernel

对于 kernel 配置，EMMC 版本的工程应包含两部分，一个是 dts 的配置，一个是 defconfig 的配置。

EMMC dts 配置

```
&sdio3 {  
    sprd,hs400es-dly = <0x55 0x7f 0x38 0x38>;  
    sprd,hs400-dly = <0x55 0xd3 0x35 0x35>;  
    sprd,hs200-dly = <0x7f 0xcd 0xcd 0xcd>;  
    sprd,ddr52-dly = <0x32 0x23 0x18 0x18>;  
    vmmc-supply = <&vddemmc>;  
    voltage-ranges = <3000 3000>;  
    bus-width = <8>;  
    non-removable;  
    cap-mmc-hw-reset;  
    mmc-hs400-enhanced-strobe;  
    mmc-hs400-1_8v;  
    mmc-hs200-1_8v;  
    mmc-ddr-1_8v;  
    sprd,name = "sdio_emmc";  
    sprd,sdio-adma;  
    no-sdio;  
    no-sd;  
    status = "okay";  
};
```

EMMC defconfig 配置

```
CONFIG_MMC=y  
# CONFIG_PWRSEQ_EMMC is not set  
# CONFIG_PWRSEQ_SIMPLE is not set  
CONFIG_MMC_BLOCK=y  
CONFIG_MMC_BLOCK_MINORS=8  
# CONFIG_SDIO_UART is not set  
# CONFIG_MMC_TEST is not set  
# CONFIG_MMC_EMBEDDED_SDIO is not set  
# CONFIG_MMC_PARANOID_SD_INIT is not set  
# CONFIG_MMC_CRYPT is not set  
  
#  
# MMC/SD/SDIO Host Controller Drivers  
#
```

```
# CONFIG_MMC_DEBUG is not set
# CONFIG_MMC_ARMMMC is not set
# CONFIG_MMC_SPRD_SDHCR10 is not set
# CONFIG_MMC_SDHCI is not set
# CONFIG_MMC_SPI is not set
CONFIG_MMC_SPRD_SDHCR11=y
# CONFIG_MMC_DW is not set
# CONFIG_MMC_VUB300 is not set
# CONFIG_MMC_USHC is not set
# CONFIG_MMC_USDHI6ROL0 is not set
# CONFIG_MMC_MTK is not set
# CONFIG_MEMSTICK is not set
```

Unisoc Confidential For hiar