UNISOC ROM Partition Config Customization Guide For Kernel4.14

Release Date	2019/9/26
Document No.	
Version	V1.1
Document Type	Customization Guide
Platform	All platforms using K4.14
OS Version	Kernel4.14

声明 Statement

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，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。

All data and information contained in or disclosed by this document is confidential and proprietary information of UNISOC and all rights therein are expressly reserved. This document is provided for reference purpose, no license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, and no express and implied warranties, including butwithout limitation, the implied warranties of fitness for any particular purpose, and non-infringement, as well as any performance. By accepting this material, the recipient agrees that the material and the information contained therein is to be held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of UNISOC. UNISOC may make any changes at any time without prior notice. Although every reasonable effort is made to present current and accurate information, UNISOC makes no guarantees of any kind with respect to the matters addressed in this document. In no event shall UNISOC be responsible or liable, directly or indirectly, for any damage or loss caused or alleged to be caused by or in connection with the use of or reliance on any such content.

关键字 Keywords

MTD	memory technology device
UBI	Unsorted Block Images
UBIFS	Unsorted Block Image File System
eMMC	Embedded Multi Media Card

版本历史 Revision history

版本 Version	日期 Date	作者 Author	描述 Description
V1.0	2019/9/6	UNISOC	初稿
V1.1	**2019/9/26**	**UNISOC**	**增加适用 platform**

前　言 Foreword

一 范围 Scope

章节旨在初步介绍 andriodQ UNISOC ROM 空间的分配情况，详细内容还需要从代码层面去深入的理解。

二 内容定义 Details Definitions

1. 定义 Definitions

2. 符号定义 Symbols

3. 缩略语 Abbreviations

MTD	memory technology device
UBI	Unsorted Block Images
UBIFS	Unsorted Block Image File System
eMMC	Embedded Multi Media Card

三 参考文献 References

N/A

目　录 Contents

1 概览 Overview

首先看看有哪些分区，如图 1 所示：

1	prodnv	sparse format	size="5" /MBytes>	这个分区是 prodnv.bin ，该分区就是开机后系统中的 productinfo 分区，保存 adc 校准参数、eng.db 数据库。
2	Miscdata	RAW	size="1" /MBytes>	这个分区用来保存 ota、recovery 时的一些数据。
3	recovery	sparse format	size="40" /MBytes>	这个分区用来存放 recovery.img ，恢复出厂设置相关。
4	misc	RAW	size="1" /MBytes>	此分区包含杂项系统关闭开关上的窗体中的设置相关。
5	trustos	RAW	size="6" /MBytes>	这个分区用来存放 tos-sign.bin
6	Trustos_bak	RAW	size="6" /MBytes>	这分区是 trustos 的备份，防止 trustos 破坏导致系统无法开机。
7	sml	RAW	size="1" /MBytes>	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" /MBytes>	这分区是 sml 的备份 ,防止 sml 破坏导致系统无法开机。
9	uboot	RAW	size="1" /MBytes>	存放 ubootloader img
10	Uboot_bak	RAW	size="1" /MBytes>	这分区是 ubootloader img 的备份 ,防止 sml 破坏导致系统无法开机.
11	Uboot_log	RAW	size="4" /MBytes>	存放 uboot log
12	logo	RAW	size="6" /MBytes>	这个分区用来存放开机 logo 图片。
13	fbootlogo	RAW	size="6" /MBytes>	这个分区用来存放 fastboot 模式的 logo 图片。
14	L_fixnv1	RAW	size="2" /MBytes>	这个分区用来存放 pubcp_nvitem.bin，射频参数相关。
15	L_fixnv2	RAW	size="2" /MBytes>	这分区是 fixnv1 的备份，防止 fixnv 破坏导致系统无法开机。
16	L_runtimenv1	RAW	size="2" /MBytes>	这个分区是运行时由 modem 生成，是 fixnv 的一份复制。

17	L_runtimenv2	RAW	size="2" /MBytes>	这个分区是L_runtimenv1的备份，起到掉电保护的作用。
18	Gpsgl	RAW	size="1" /MBytes>	这个分区用来存放gnssmodem.bin
19	Gpsbd	RAW	size="1" /MBytes>	这个分区用来存放gnssbdmodem.bin
20	Wcnmodem	RAW	size="10" /MBytes>	这个分区用来存放cm4_v2.bin，是Connectivity 芯片的的协议栈相关。
21	persist	sparse format	size="2" /MBytes>	这个分区用来存放persist.img
22	L_modem	RAW	size="25" /MBytes>	这个分区用来存放 pubcp _modem.dat，通信协议栈相关。
23	L_deltanv	RAW	size="25" /MBytes>	这个分区用来存放 pubcp_minfo_ncache_deltanv.bin
24	L_gdsp	RAW	size="10" /MBytes>	这个分区用来存放 pubcp_DM_ DSP.bin，数字处理等。
25	L_ldsp	RAW	size="20" /MBytes>	这个分区用来存放pubcp_LTEA _DSP.bin.

26	Pm_sys	RAW	size="1" /MBytes>	这个分区用来存放 cm4.bin
27	Teecfg	RAW	size="1" /MBytes>	安全相关
28	Teecfg_bak	RAW	size="1" /MBytes>	这个分区是 Teecfg 的备份分区，起到掉电保护作用。
29	boot	sparse format	size="35" /MBytes>	这个分区用来存放 boot.img ，kernel 驱动相关。
30	dtbo	sparse format	size="8" /MBytes>	存放 dtbo.img
31	super	sparse format	size="4100" /MBytes>	这个分区用来存放 SUPER.img，android 系统相关。
32	cache	sparse format	size="150" /Mbytes>	这个分区用来存放 cache.img，在 CTS 测试，恢复出厂设置是需要使用。
33	socko	sparse format	size="75" /MBytes>	这个分区用来存放 socko.img
34	odmko	sparse format	size="25" /MBytes>	这个分区用来存放 odmko.img

35	vbmeta	sparse format	size="1" /MBytes>	这个分区用来存放 vbmeta-sign.img
36	Vbmeta_bak	sparse format	size="1" /MBytes>	这个分区是 vbmeta 的备份分区，起到掉电保护作用。
37	Metadata	RAW	size="16" /MBytes>	Erase misc section operation
38	Sysdumpdb	RAW	size="10" /MBytes>	Erase misc section operation
39	Vbmeta_system	sparse format	size="1" /MBytes>	这个分区用来存放 vbmeta_system.img
40	Vbmeta_vendor	sparse format	size="1" /MBytes>	这个分区用来存放 vbmeta_vendor.img
41	userdata	sparse format	size=0xFFFFFFFF	这个分区用来存放 userdata.img ，包含用户的数据。

图 1

图 1 是展讯平台 ROM 空间划分情况以及分区格式、分区大小，分区功能的的初步描述，先对展讯平台的分区情况有了一个宏观上的认识，接下来我们对 eMMC 的分区做进一步的介绍。

2 框架

2.1 EMMC 分区形式

在 eMMC 方案中我们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。

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>
```

3 分区操作

3.1 如何增加分区

增加一个分区请参考如下方法进行，这个不区分 eMMC 和 nand ，修改方法是一致的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。

例如：

增加一个 fast_logo 分区；

第一步：增加一行

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


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。

第二步：增加 file 定义

```
<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>
```

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。

如果想增加一个 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。

3.2 如何修改分区大小

这个操作可以说成是增加一个分区的一部分，因为新增加一个分区也要配置分区的大小。随着项目的进行起初配置的分区大小已经无法满足当前的需求的时候就需要重新配置分区的大小，那么如何配置分区大小那？操作很简单举例如下：

第一、 修改 system 分区修改 system 分区大小涉及以下几点：

（1） /device/sprd/项目/平台名/平台名.xml 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。

第二、修改 cache 分区

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

```
<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。

第三、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。
```

注意：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。

第四、修改 dtbo 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="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。
```


第五、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。

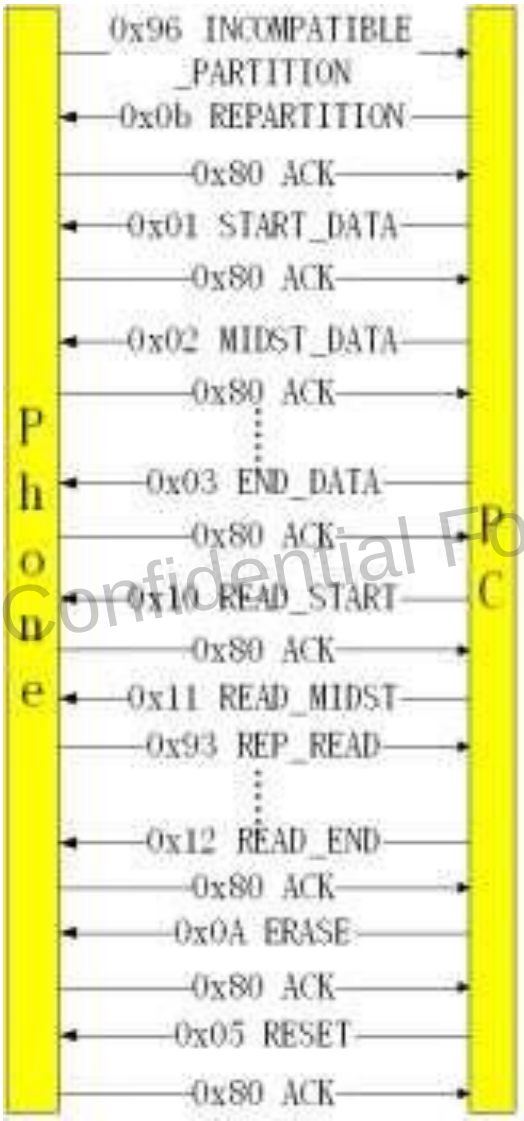
3.3 如何删除分区

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。

4 Repartition 分区生效流程

这一节从软件流程上说明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（nand/eMMC）。

整个下载过程采用的是简单的问答式协议（如下图），由 PC（下载工具）主控，相当于 server，手机则相当于 client。从下图协议握手过程可以看到，下载过程第一步就是重分区。



重分区 repartition：我们在下载 pac 的时候，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：

/ u-boot15/common/cmd_download.c

```
int do_download(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{

      ……
      /* 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);
      dl_cmd_register(BSL_CMD_CHECK_ROOTFLAG, dl_cmd_check_rootflag);
      ……
      usb_init(0);

      /* uart download doesn't supoort disable hdlc, so need check it */
      if (FDL_get_DisableHDLC() == NULL)
                dl_send_ack (BSL_INCOMPATIBLE_PARTITION);
      else {
                Da_Info.dwVersion = 2;
                Da_Info.bDisableHDLC = 1;

#ifndef CONFIG_NAND_BOOT
      if (gd->boot_device == BOOT_DEVICE_EMMC)
                Da_Info.bIsOldMemory = emmc_2ndhand_detect();
      else
#endif
                Da_Info.bIsOldMemory = 0;

                ack_packet.body.type = BSL_INCOMPATIBLE_PARTITION;
                memcpy((unchar *)ack_packet.body.content, (unchar *)&Da_Info,sizeof(Da_Info));
                ack_packet.body.size = sizeof(Da_Info);
```

```
                dl_send_packet(&ack_packet);
        }

        /* 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 SL_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。

/u-boot15/common/dloader/dl_cmd_proc.c

```
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，以下以 emmc 中的处理函数为例：

```
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。/u-boot15/common/sprd_common_rw.c


```
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。

5 芯片工程存储（Emmc）相关代码概述

本章节主要从 device splloader uboot kernel 四方面介绍关于 emmc 的配置。

5.1 Device

工程的 device 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 recovery.fstab；作用：同 fstab 用于恢复出厂设置的文件系统挂载。

Emmc:

```
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
vendor /vendor ext4 ro,barrier=1 wait,logical,first_stage_mount
product /product ext4 ro,barrier=1 wait,logical,first_stage_mount
#/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/socko /mnt/product/socko ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,check,first_stage_mount
#/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/odmko /mnt/product/odmko ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,check,first_stage_mount
/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/userdata /data        ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,check,fileencryption=aes-256-xts
#/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/userdata /data        ext4
noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,forceencrypt=footer,check
/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/metadata /metadata    ext4
nodev,noatime,nosuid,errors=panic wait,formattable,first_stage_mount
/devices/platform/soc/soc:ap-ahb/20200000.usb/musb-hdrc.*.auto/usb*      auto          vfat
defaults voldmanaged=usbdisk:auto
/devices/platform/soc/soc:ap-ahb/20300000.sdio/mmc_host/mmc1/mmc1:*/block/mmcblk1      auto
vfat defaults voldmanaged=sdcard0:auto,noemulatedsd,encryptable=footer
/dev/block/platform/soc/soc:ap-ahb/20600000.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/20600000.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-ahb/20600000.sdio/by-name/socko /mnt/vendor/socko ext4
ro,noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,avb=socko,check
```

```
/dev/block/platform/soc/soc:ap-ahb/20600000.sdio/by-name/odmko /mnt/vendor/odmko ext4
ro,noatime,nosuid,nodev,nomblk_io_submit,noauto_da_alloc wait,avb=odmko,check
/dev/block/platform/soc/soc:ap-ahb/20600000.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
```

BoardPartitionconfig.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

Emmc 版本 ：主要配置相关镜像的 size 和文件系统格式：

```
# ext4 partition layout
#BOARD_VENDORIMAGE_PARTITION_SIZE := 419430400
BOARD_VENDORIMAGE_FILE_SYSTEM_TYPE := ext4
TARGET_COPY_OUT_VENDOR=vendor
TARGET_USERIMAGES_USE_EXT4 := true
BOARD_BOOTIMAGE_PARTITION_SIZE := 36700160
BOARD_RECOVERYIMAGE_PARTITION_SIZE := 41943040
#BOARD_SYSTEMIMAGE_PARTITION_SIZE := 3145728000
BOARD_CACHEIMAGE_PARTITION_SIZE := 150000000
BOARD_PRODNVIMAGE_PARTITION_SIZE := 5242880
BOARD_USERDATAIMAGE_PARTITION_SIZE := 134217728
BOARD_DTBOIMG_PARTITION_SIZE := 8388608
BOARD_DTBIMG_PARTITION_SIZE := 8388608
BOARD_FLASH_BLOCK_SIZE := 4096
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_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_SOCKOIMAGE_FILE_SYSTEM_TYPE := ext4
BOARD_ODMKOIMAGE_FILE_SYSTEM_TYPE := ext4
BOARD_SOCKOIMAGE_PARTITION_SIZE := 78643200 # 75M
```

```
BOARD_ODMKOIMAGE_PARTITION_SIZE := 26214400 # 25M
```

5.2 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);
```

5.3 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 和 nand 片初始化，文件，分区系统的初始化和镜像的写入任务，本章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 分区的重分区代码如下：

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

    ret = ufs_lu_reconstruct(factory_lu_tbl, &lu_common_tbl);
    if (UFS_SUCCESS != ret)
            return -1;

    /*future extention, now we only have one none boot lu*/
    dev_id = get_lun_by_bootable(NONE_BOOT);
    if (-1 == dev_id)
            return -1;

    dev_desc = get_dev("ufs", dev_id);
    if (NULL == dev_desc)
            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;
        }
}
```

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

函数 write_gpt_table 为重分区的重点函数，涉及到包括各个分区名称以及起始地址信息的 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;
}
```


分区寻找以及镜像写入在建立好 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对应的分区中。具体代码实现在路径 u-boot15/common/sprd_common_rw.c 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;
}
```

5.4 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 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 = <&vddemmccore>;
      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

```
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

#
# MMC/SD/SDIO Host Controller Drivers
#
# CONFIG_MMC_DEBUG is not set
# CONFIG_MMC_ARMMMCI 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
CONFIG_NEW_LEDS=y
CONFIG_LEDS_CLASS=y
# CONFIG_LEDS_CLASS_FLASH is not set
# CONFIG_LEDS_BRIGHTNESS_HW_CHANGED is not set
```