Rev. 1, 8/2021

S32G eMMC应用：GP，RPMB

by John Li(nxa08200)

目前 S32G 的 BSP(BSP29)的 eMMC 启动，是将 U-Boot 的镜像放在 eMMC 的 user partition 中，可以考虑将储存 boot image 的部分配置为 GP，这样的损坏可能性要小一些。另外一种办法是将 U-Boot 储存在外部 QSPI NOR 中。

由于目前 S32G2 的 Rom codes 不支持从 eMMC Boot Partition 启动，所以 Boot Partition 的内容以后如果有更新的芯片支持，再做讨论。

另外也会说明一下如何使用 eMMC RPMB 的应用。

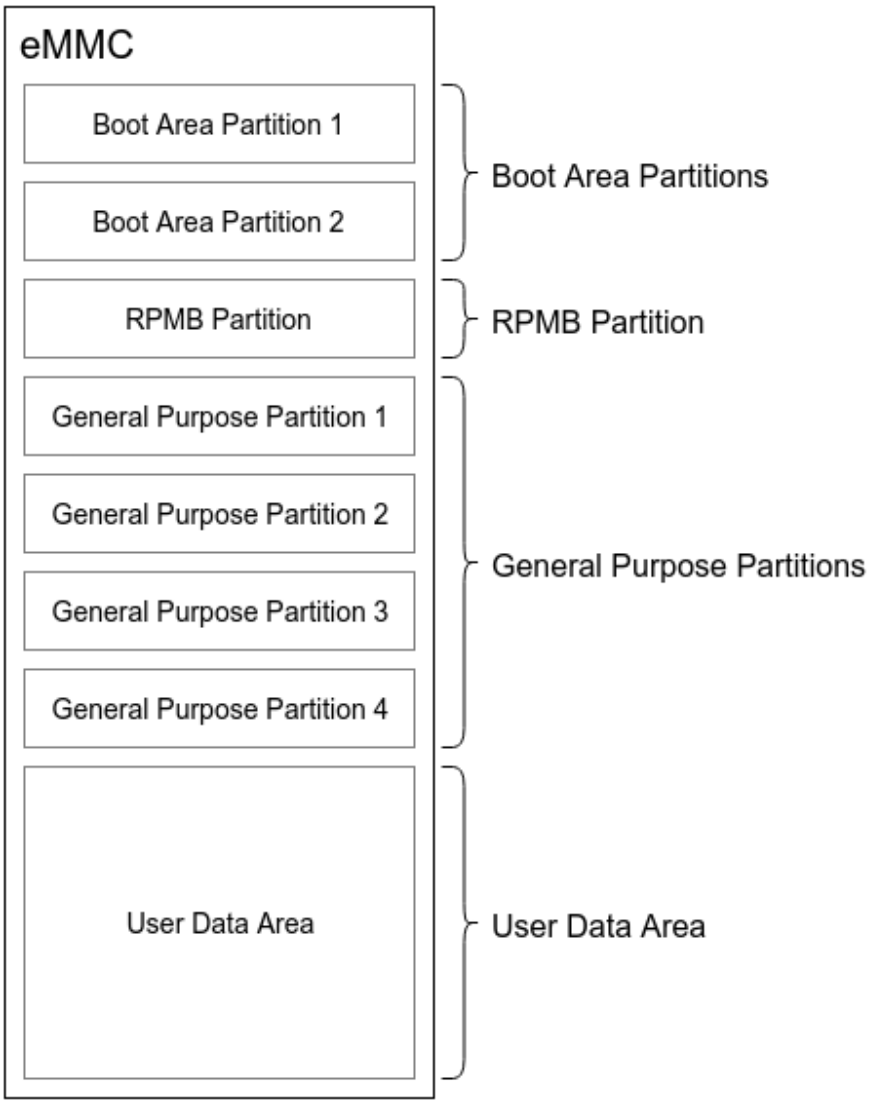
通常情况下，对 eMMC 的烧录都需要通过烧录工具，在 S32G 上，可能是通过网络的 OTA 升级，所以是需要在烧录工具的镜像中实现以下功能，本文中是用正式启动镜像来测试的。

	日期	注释	作者
V1	2021-8	• 创建文档	• John Li
V2	2021-8	• 删除 Boot partition 相关内容，因为 Rom codes 不支持	• John Li

目录

1 eMMC 的分区情况

大部分 eMMC 都有类似如下的分区，其中 BOOT、RPMB 和 UDA 一般是默认存在的，GPP 分区需要手动创建。



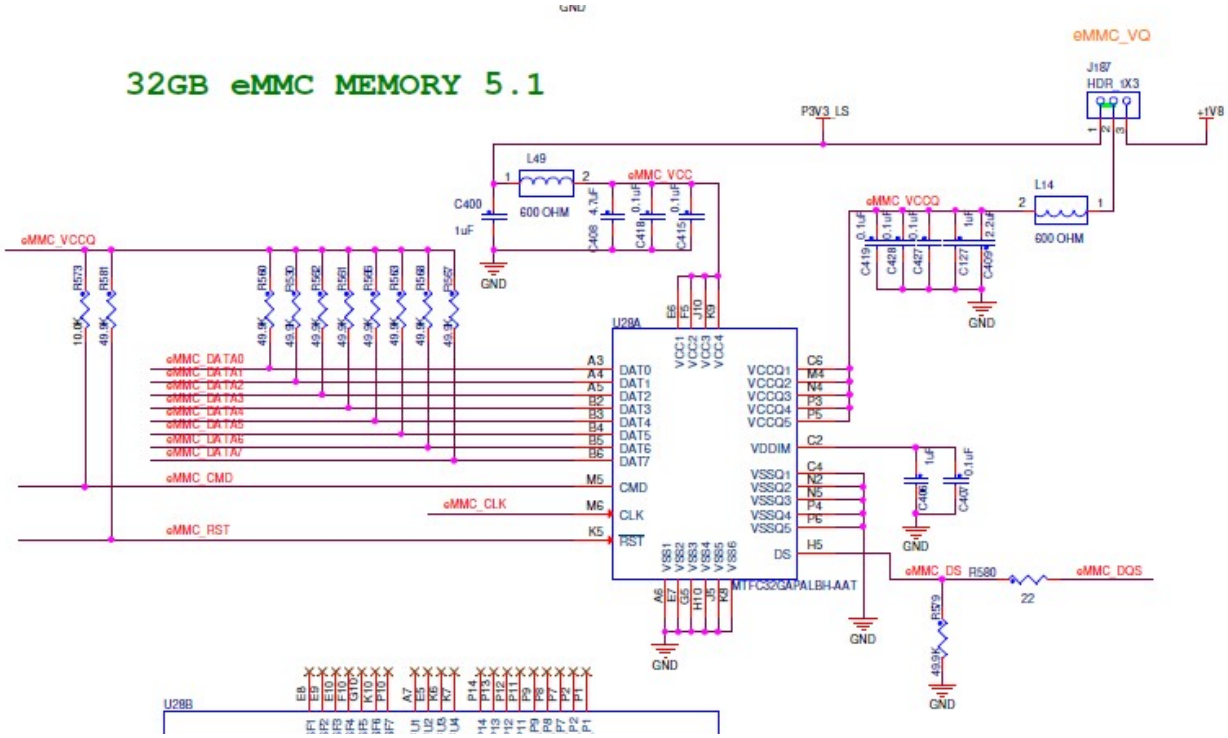
BOOT 主要是为了支持从 eMMC 启动系统而设计的；RPMB 即 Replay Protected Memory Block 简称，通常用来保存安全限管的数据；GPP 主要用于存储系统或者用户数据。UDA 通常会进行再分区，然后根据不同目的存放相关数据，或者格式化成不同文件系统。相对而言，所以 Boot partition, UDA 比较常见，RPMB 在用于保存密码时也会用到，GPP 较不常用，实用中使用 UDA 替代比较多。

2 S32G+BSP29 上默认的 eMMC 启动

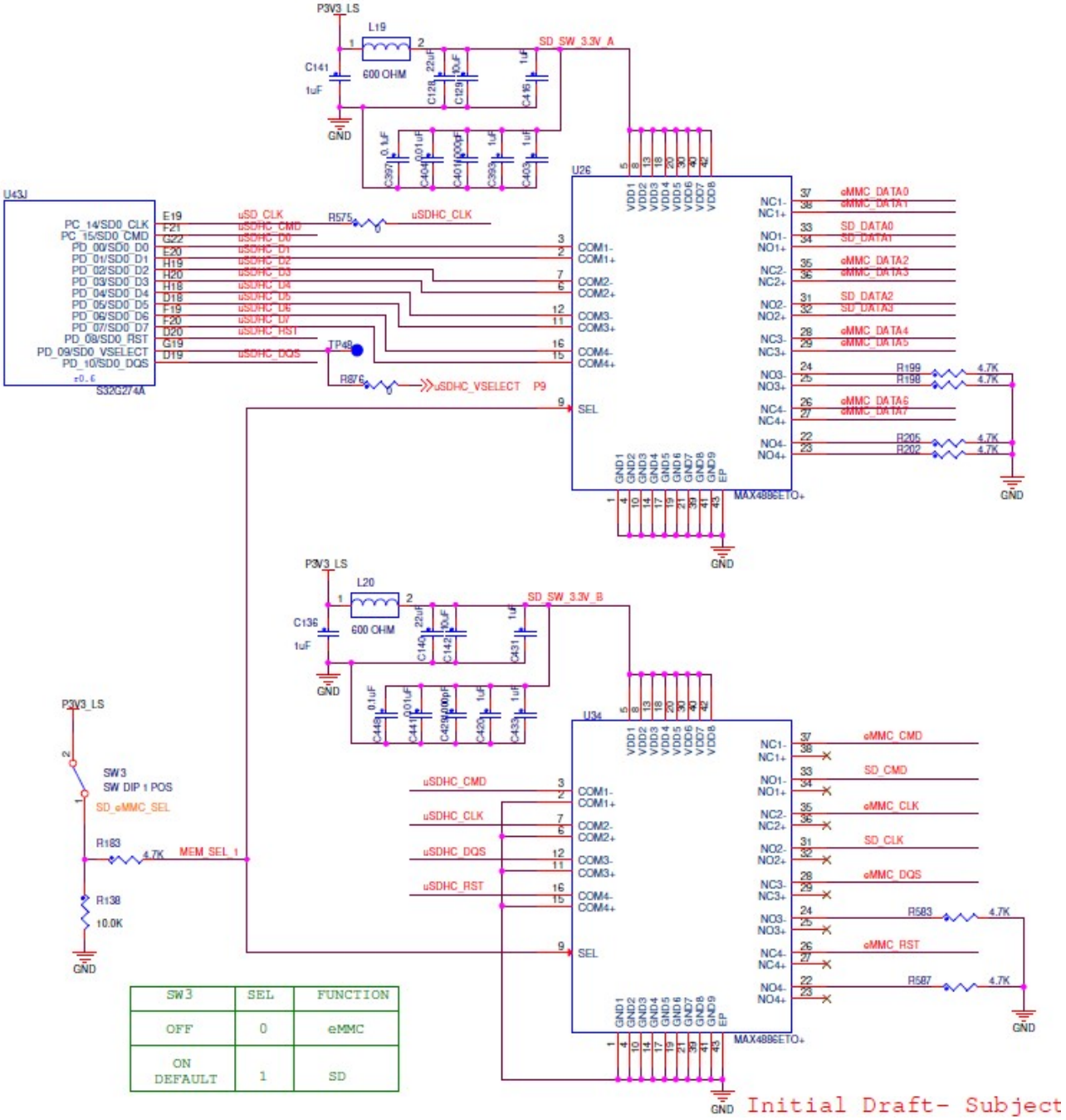
2.1 eMMC 硬件设计

从 NXP 公网 www.nxp.com/s32g 下载测试使用 S32G RDB2 板的硬件资料，包括原理图。

S32G RDB2 板目前的原理图版本是 S32G-VNP-RDB2-SCH-REV-D .pdf:SCH-47800, 所使用的 eMMC 是 MTFC32GAPALBH-AAT。32GB eMMC5.1，默认是工作在 DDR50 模式，IO 电平是 3.3V，如果需要工作在 HS400 模式需要修改软件和 IO 电平为 1.8V，相关修改办法请参考 BSP 用户手册，或文档《S32G_Uboot_BSP29_V3-20210607.pdf》,《S32G_Kernel_V...pdf》，是否使用 HS400 与本文无关，所以本文是基于默认的 3.3V, DDR50 的模式测试。



S32G 仅支持一个 uSDHC 口，所以在 RDB2 板上设计了 SDcard 与 eMMC 的切换器，所以本文中对 eMMC 的整体镜像烧写是使用 SDcard 启动后切换的方式，而实际应用中，一般客户是通过网络在线烧写，无论什么方式，烧写 Boot partition 都是使用 Linux 的命令的方式，所以本质不变，切换电路如下：



SW3	SEL	FUNCTION
OFF	0	eMMC
ON DEFAULT	1	SD

所以通过切换 SW3，可以在 SDcard 与 eMMC 之间切换。

其次，根据 S32G 芯片手册说明：

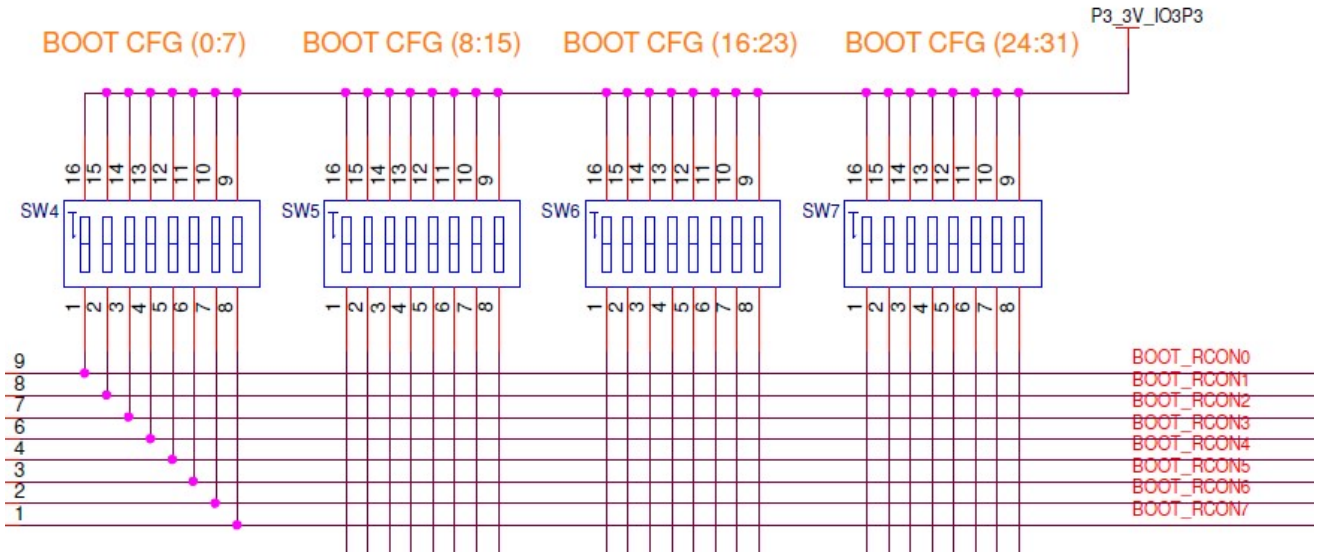
Table 169. Boot device selection

BOOT_CFG[7:5]	Boot device
000	QuadSPI flash memory
001	Reserved
010	SD
011	MMC/eMMC

所以从 SDcard 修改为从 eMMC 启动，还需要把 Boot_Cfg5 从 0 改成 1，参考原理图如下：

SW4~SW7 Default Setting

	1	2	3	4	5	6	7	8
SW4	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
SW5	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW6	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
SW7	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF



也就是说如果是 Sdcard 启动，SW4_7=1。如果是 eMMC 启动，SW4_67=0b11。

最后要注意 Boot RCON 配置，fuse Map 定义如下：

BOOT_CFG[23]	BOOT_CFG[22]	BOOT_CFG[21	BOOT_CFG[20]	BOOT_CFG[19]

TDH : Time Hold Delay 00: Data aligned at PosEdge of Internal reference clock 01: Data aligned with 2X serial flash				CKN 0 : Diffrential Clock not required 1: Diffrential Clock required
				SD Speed 0 - Default Speed 1 - High Speed
	MMC Boot Modes 0000- 1-bit Normal Speed 0001 - 4-bit Normal Speed 0010 - 8-bit Normal Speed 0011- 1-bit HIGH Speed 0100 - 4-bit HIGH Speed 0101 - 8-bit HIGH Speed 0110 - 4-bit DDR HIGH Speed 0111 - 8-bit DDR HIGHl Speed			

所以默认的 Boot_cfg[22~19]对应的 SW6[7~4]默认=0b0000，这样的 1bit Normal Speed 模式，最高可以修改成 SW6_7=off, SW6_654=on。设置为 8bit DDR High Speed 模式。

2.2 eMMC 的镜像烧写办法与启动

本文使用 SDcard 启动，然后通过 Uboot 把整个镜像写到 eMMC 的 User partition 中去，之前把一个有特殊标记的 U-boot 放在 Sdcard 的 rootfs 中。

1. 烧写整个 sdcard 镜像到一张空白的 sdcard 中：

   本文使用 BSP29 来测试，下载其默认 demo 镜像 binaries_auto_linux_bsp29.0_s32g274_pfe/s32g274ardb2/fsl-image-auto-s32g274ardb2.sdcard 使用如下 Linux 命令烧写到 sdcard 中：

```
cat /proc/partitions
major minor  #blocks  name
…
  8       32   15558144 sdc
```

…

sudo dd if=./fsl-image-auto-s32g274ardb2.sdcard of=/dev/sdc bs=1M && sync

2. 把 sdcard 中的镜像导出成一个*.sdcard 文件：

sudo fdisk /dev/sdc

…

Command (m for help): p

…

Device Boot Start End Sectors Size Id Type

/dev/sdc1 8192 139263 131072 64M c W95 FAT32 (LBA)

/dev/sdc2 139264 1081343 942080 460M 83 Linux

所以 rootfs 结束位置是 sector 1081343

sudo dd if=/dev/sdc of=s32g_emmc.sdcard bs=512 count=1081349 //1081349 >1081343 导出*.scard 镜像

3. 在 sdcard 上创建一个新的 ext3 分区：

sudo fdisk /dev/sdc

Command (m for help): p

Disk /dev/sdc: 14.9 GiB, 15931539456 bytes, 31116288 sectors

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: dos

Disk identifier: 0xb89db51a

Device Boot Start End Sectors Size Id Type

/dev/sdc1 8192 139263 131072 64M c W95 FAT32 (LBA)

/dev/sdc2 139264 1081343 942080 460M 83 Linux

Command (m for help): p

Disk /dev/sdc: 14.9 GiB, 15931539456 bytes, 31116288 sectors

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: dos

```
Disk identifier: 0xb89db51a

Device     Boot  Start     End Sectors  Size Id Type
/dev/sdc1         8192  139263  131072   64M  c W95 FAT32 (LBA)
/dev/sdc2       139264 1081343  942080  460M 83 Linux

Command (m for help): n
Partition type
   p   primary (2 primary, 0 extended, 2 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (3,4, default 3):
First sector (2048-31116287, default 2048): 1081350
Last sector, +sectors or +size{K,M,G,T,P} (1081350-31116287, default 31116287): +1G

Created a new partition 3 of type 'Linux' and of size 1024 MiB.

Command (m for help): p
Disk /dev/sdc: 14.9 GiB, 15931539456 bytes, 31116288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb89db51a

Device     Boot   Start     End Sectors  Size Id Type
/dev/sdc1          8192  139263  131072   64M  c W95 FAT32 (LBA)
/dev/sdc2        139264 1081343  942080  460M 83 Linux
/dev/sdc3       1081350 3178495 2097146 1024M 83 Linux
w
```

重新 mount:

```
sudo mkfs.ext3 -L temp /dev/sdc3
```

4. 将导出的镜像入在这个分区中：

重新 mount:

```
sudo cp s32g_emmc.sdcard /media/vmuser/temp/
sync
```

5. 使用 sdcard 启动，停止在 uboot 中,将 partition3 上的*.sdcard 加载到内存中

```
=> mmc part
Partition Map for MMC device 0  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
  1     8192            131072          b89db51a-01     0c
  2     139264          942080          b89db51a-02     83
  3     1081350         2097146         b89db51a-03     83
ext4ls mmc 0:3
<DIR>       4096 .
<DIR>       4096 ..
<DIR>      16384 lost+found
       553651200 s32g_emmc.sdcard
=> ext4load mmc 0:3 80080000 s32g_emmc.sdcard
31467520 bytes read in 1386 ms (21.7 MiB/s)
```

6. 动态将 SW3 切成 OFF，则 S32G 切换到连接 eMMC。

7. 将内存中的*.sdcard 写入到 eMMC 中：

```
mmc rescan
mmc write 80080000 0 e8000
MMC write: dev # 0, block # 0, count 950272 ... 950272 blocks written: OK
```

8. SW4 设置为 6,7 on=emmc boot, 重启：则系统从 eMMC 启动

```
U-Boot 2020.04+g61b2dc53d2 (May 26 2021 - 13:29:19 +0000)
CPU: NXP S32G274A rev. 2.1.0
Reset cause: Power-On Reset
…
[    1.939737] mmc0: SDHCI controller on 402f0000.usdhc [402f0000.usdhc] using ADMA
[    2.042840] mmc0: new DDR MMC card at address 0001
[    2.043589] mmcblk0: mmc0:0001 S0J57X 29.6 GiB //user partition
[    2.043911] mmcblk0boot0: mmc0:0001 S0J57X partition 1 31.5 MiB //boot partition
[    2.044227] mmcblk0boot1: mmc0:0001 S0J57X partition 2 31.5 MiB
[    2.044385] mmcblk0rpmb: mmc0:0001 S0J57X partition 3 4.00 MiB, chardev (243:0) //rpmb partition
```

```
[    2.045976]  mmcblk0: p1 p2
…
root@s32g274ardb2:/home# ls
root  u-boot.s32
```

2.3 增加 MMC 内核测试工具

mmc utils: 配置 mmc 的启动设备需要用到 mmc utils 工具，但是 S32G BSP29 没有安装，yocto 安装步骤如下：

1.查找 yocto 是否已有对应的工具，及版本信息 ~/bsp29/fsl-auto-yocto-bsp/build_s32g274ardb2$ bitbake -s | grep mmc mmc-utils :0.1+gitAUTOINC+73d6c59af8-r0

2.修改所编译的 image 对象 bb 文件

如 ~/bsp29/fsl-auto-yocto-bsp/sources/meta-alb/recipes-fsl/images$ vim fsl-image-auto.bb

3. 添加工具

```
49 # Other useful tools
50 IMAGE_INSTALL_append = " rsync irqbalance i2c-tools"
51 IMAGE_INSTALL_append = " mmc-utils"
```

也可以直接 git 下来在交叉编译环境中 standalone 编译：

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/cjb/mmc-utils.git
cd mmc-utils/
source ~bsp29/sdk/environment-setup-aarch64-fsl-linux
make
```

然后把生成的 mmc 命令放入到 emmc 中，参考 2.2 节。

3 eMMC GP 功能的测试

3.1 eMMC GP 功能的说明

eMMC 提供了 General Purpose Partitions (GPP)，主要用于存储系统和应用数据。在很多使用 eMMC 的产品中，GPP 都没有被启用，因为它在功能上与 UDA 类似，产品上直接使用 UDA 就可以满足需求。eMMC 最多可以支持 4 个 GPP，每一个 GPP 的大小可以单独配置。GPAP 配置定义完成之后每一个 GPAP 的起始地址都为 0x0。eMMC 标准中，为 GPP 定义了两类属性，Enhanced attribute 和 Extended attribute。每个 GPP 可以设定两类属性中的一种属性，不可以同时设定多个属性。

- Enhanced attribut

  Default, 未设定 Enhanced attribute。

  Enhanced storage media， 设定 GPP 为 Enhanced storage media。

在 eMMC 标准中，实际上并未定义设定 Enhanced attribute 后对 eMMC 的影响。Enhanced attribute 的具体作用，由芯片制造商定义。在实际的产品中，设定 Enhanced storage media 后，一般是把该分区的存储介质从 MLC 改变为 SLC，提高该分区的读写性能、寿命以及稳定性。由于 1 个存储单元下，MLC 的容量是 SLC 的数倍，所以在总的存储单元数量一定的情况下，如果把原本为 MLC 的分区改变为 SLC，会减少 eMMC 的容量，就是说，此时 eMMC 的实际总容量比标称的总容量会小一点。

- Extended attribute

Default, 未设定 Extended attribute。

System code， 设定 GPP 为 System code 属性，该属性主要用在存放操作系统类的、很少进行擦写更新的分区。

Non-Persistent，设定 GPP 为 Non-Persistent 属性，该属性主要用于存储临时数据的分区，例如 tmp 目录所在分区、 swap 分区等。

在 eMMC 标准中，同样也没有定义设定 Extended attribute 后对 eMMC 的影响。Extended attribute 的具体作用，由芯片制造商定义。Extended attribute 主要是跟分区的应用场景有关，厂商可以为不用应用场景的分区做不同的优化处理。

3.2 eMMC GP 功能的测试

由于正式启动的镜像就是工作在 eMMC 上的，不方便直接在上面创建 GP 分区，所以以下代码是理论分析，实际测试需要在在线的烧录工具镜像中实现，比如说用 ramfs 启动或网络文件系统启动：

- 测试命令：

```
./mmc gp create --help
Usage:
    mmc gp create <-y|-n|-c> <length KiB> <partition> <enh_attr> <ext_attr> <device>
        Create general purpose partition for the <device>.
        Dry-run only unless -y or -c is passed.
        Use -c if more partitioning settings are still to come.
        NOTE!  This is a one-time programmable (unreversible) change.
        To set enhanced attribute to general partition being created set
         <enh  attr> to 1 else set it to 0.
        To set extended attribute to general partition
         set <ext_attr> to 1,2 else set it to 0
```

- 检查可以enhance的最大大小

```
root@s32g274ardb2:/home# ./mmc extcsd read /dev/mmcblk0 |grep MAX_ENH_SIZE_MULT -A 1
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x000760
 i.e. 15466496 KiB
```

- 创建两个GP 分区

Create gp2

```
./mmc gp create -n 93888 2  1 0   /dev/mmcblk0
```

Enhanced GP1 Partition Size [GP_SIZE_MULT_1]: 0x00000b
i.e. 90112 KiB
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x0000ea
i.e. 3833856 KiB

Note: Please use -n, just check and set the eMMCregister, if it is not the last gpto create

Create gp1
mmc gp create **-y** 524288 1 **1** 0 /dev/mmcblk3
Enhanced GP1 Partition Size [GP_SIZE_MULT_1]: 0x000040
i.e. 524288 KiB
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x0000ea
i.e. 3833856 KiB

NOTE! This is a one-time programmable (unreversible) change, Needs power cycle

- 使用 GP

ls /dev/mmcblk0*
mmcblk0 mmcblk0boot0 mmcblk0boot1 **mmcblk0gp0 mmcblk0gp1** mmcblk3rpmb
fdisk-l
Disk /dev/mmcblk3: **6**GiB, 6476005376 bytes, 12648448 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk **/dev/mmcblk3gp1**: 88 MiB, 92274688 bytes, 180224 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk **/dev/mmcblk3gp0**: 512 MiB, 536870912 bytes, 1048576 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

Note: eMMCspends about 2G to get the enhanced88MiB + 512MiB = 600MiBThe total volume to about 6.6G from about 8G.

fdisk/dev/mmcblk3gp0
Command (m for help): p
Disk /dev/mmcblk3gp0: 512 MiB, 536870912 bytes, 1048576 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabeltype: dos
Disk identifier: 0xd0d48a7b
Device Boot Start End Sectors Size Id Type
/dev/mmcblk3gp0p1 2048 264191 262144 128M c W95 FAT32 (LBA)
/dev/mmcblk3gp0p2 264192 1048575 784384 383M 83 Linux
mkfs.vfat-F 32 /dev/mmcblk3gp0p1

mkfs.ext3 /dev/mmcblk3gp0p2

4 eMMC RPMB 功能的测试

4.1 eMMC RPMB 功能的说明

- RPMB 介绍：

RPMB（Replay Protected Memory Block）Partition 是 eMMC 中的一个具有安全特性的分区。eMMC 在写入数据到 RPMB 时，会校验数据的合法性，只有指定的 Host 才能够写入，同时在读数据时，也提供了签名机制，保证 Host 读取到的数据是 RPMB 内部数据，而不是攻击者伪造的数据。RPMB 在实际应用中，通常用于存储一些有防止非法篡改需求的数据，例如手机上指纹支付相关的公钥、序列号等。RPMB 可以对写入操作进行鉴权，但是读取并不需要鉴权，任何人都可以进行读取的操作，因此存储到 RPMB 的数据通常会进行加密后再存储。

- 容量大小

两个 RPMB Partition 的大小是由 Extended CSD register 的 BOOT_SIZE_MULT Field 决定，大小的计算公式如下：Size = 128Kbytes x BOOT_SIZE_MULT 一般情况下，Boot Area Partition 的大小是 128KB 的倍数，EMMC 中默认为 4 MB，即 RPMB_SIZE_MULT 为 32，部分芯片厂家会提供改写 RPMB_SIZE_MULT 的功能来改变 RPMB Partition 的容量大小。RPMB_SIZE_MULT 最大可以为 128，即 Boot Area Partition 的最大容量大小可以为 128 x 128 KB = 16384 KB = 16 MB。

- Replay Protect 原理

使用 eMMC 的产品，在产线生产时，会为每一个产品生产一个唯一的 256 bits 的 Secure Key，烧写到 eMMC 的 OTP 区域（只能烧写一次的区域），同时 Host 在安全区域中（例如：TEE）也会保留该 Secure Key。在 eMMC 内部，还有一个 RPMB Write Counter。RPMB 每进行一次合法的写入操作时，Write Counter 就会自动加一 。通过 Secure Key 和 Write Counter 的应用，RMPB 可以实现数据读取和写入的 Replay Protect。

- RPMB 数据读取

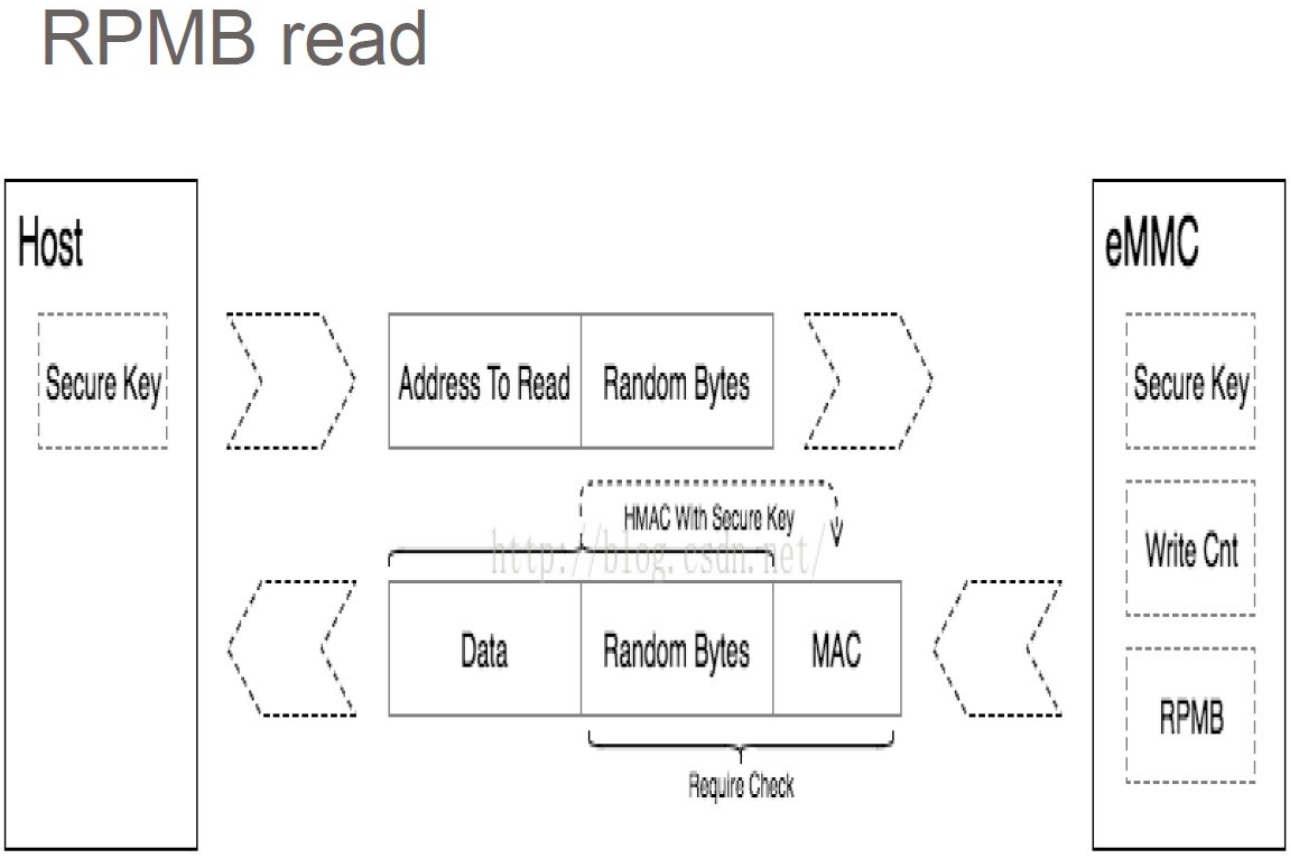
RPMB 数据读取的流程如下：

1、 Host 向 eMMC 发起读 RPMB 的请求，同时生成一个 16 bytes 的随机数，发送给 eMMC。

2、 eMMC 将请求的数据从 RPMB 中读出，并使用 Secure Key 通过 HMAC SHA-256 算法，计算读取到的数据和接收到的随机数拼接到一起后的签名。然后，eMMC 将读取到的数据、接收到的随机数、计算得到的签名一并发送给 Host。

3、 Host 接收到 RPMB 的数据、随机数以及签名后，首先比较随机数是否与自己发送的一致，如果一致，再用同样的 Secure Key 通过 HMAC SHA-256 算法对数据和随机数组合到一起进行签名，如果签名与 eMMC 发送的签名是一致的，那么就可以确定该数据是从 RPMB 中读取到的正确数据，而不是攻击者伪造的数据。

通过上述的读取流程，可以保证 Host 正确的读取到 RPMB 的数据。

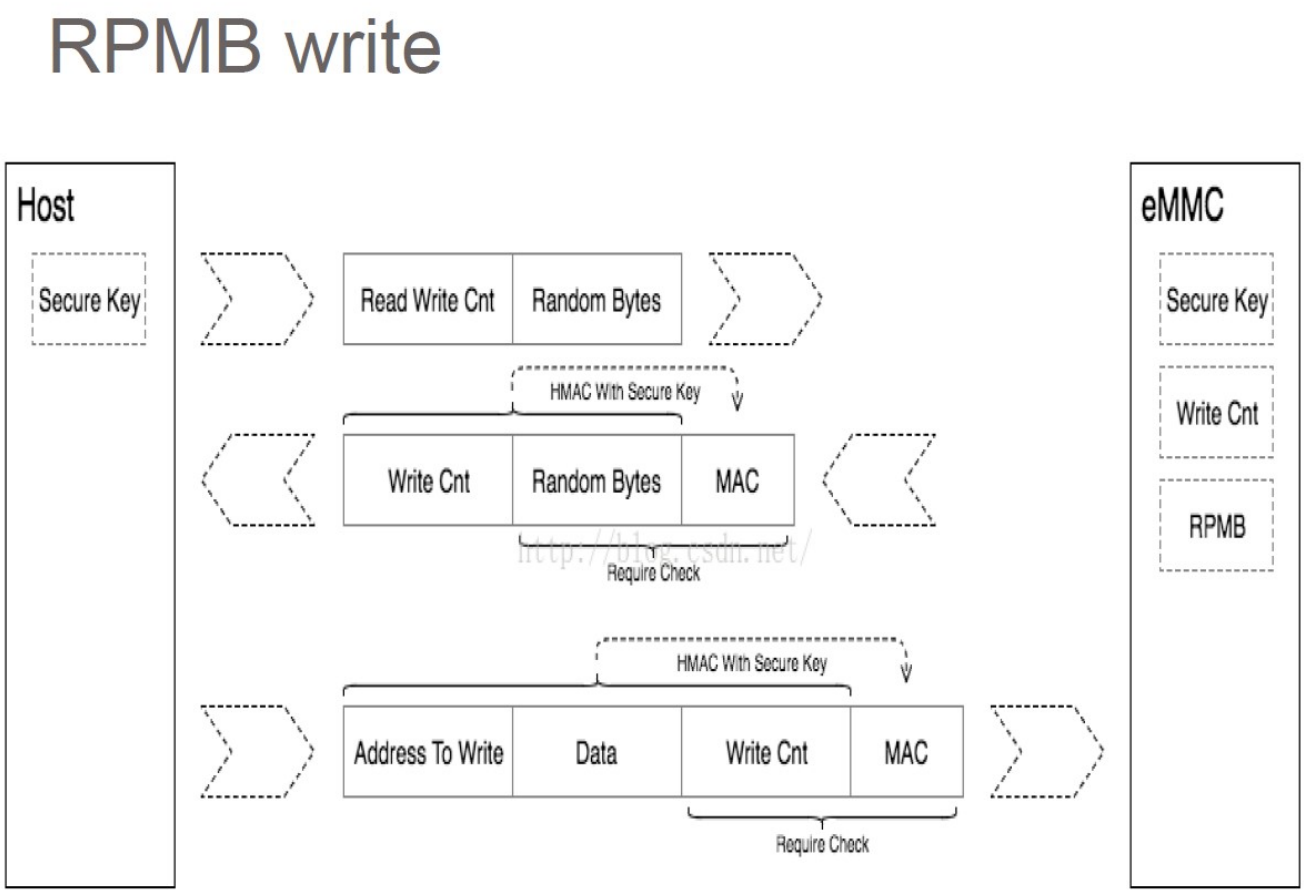


RPMB 数据写入，RPMB 数据写入的流程如下：

1、Host 按照上面的读数据流程，读取 RPMB 的 Write Counter（通过 Write Counter 来识别数据的有效性）。

2、Host 将需要写入的数据和 Write Counter 拼接到一起并计算签名，然后将数据、Write Counter 以及签名一并发给 eMMC。

3、eMMC 接收到数据后，先对比 Write Counter 是否与当前的值相同，如果相同那么再对数据和 Write Counter 的组合进行签名，然后和 Host 发送过来的签名进行比较，如果签名相同则鉴权通过，将数据写入到 RPMB 中。



通过上述的写入流程，可以保证 RPMB 不会被非法篡改。

4.2 eMMC RPMB 功能的测试

- RPMB 的启动信息：

```
dmesg |grep rpmb
[    2.165754] mmcblk0rpmb: mmc0:0001 S0J57X partition 3 4.00 MiB, chardev (243:0)
```

- 设备节点：

```
ls /dev/mmc*
/dev/mmcblk0rpmb
```

- mmc utils 的帮助：

```
./mmc rpmb --help
Usage:
        mmc rpmb write-key <rpmb device> <key file>
                Program authentication key which is 32 bytes length and stored
```

```
            in the specified file. Also you can specify '-' instead of
            key file path to read the key from stdin.
            NOTE!  This is a one-time programmable (unreversible) change.
            Example:
              $ echo -n AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH | \
                mmc rpmb write-key /dev/mmcblk0rpmb -
      mmc rpmb read-counter <rpmb device>
            Counter value for the <rpmb device> will be read to stdout.
      mmc rpmb read-block <rpmb device> <address> <blocks count> <output file> [key file]
            Blocks of 256 bytes will be read from <rpmb device> to output
            file or stdout if '-' is specified. If key is specified - read
            data will be verified. Instead of regular path you can specify
            '-' to read key from stdin.
            Example:
              $ echo -n AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH | \
                mmc rpmb read-block /dev/mmcblk0rpmb 0x02 2 /tmp/block -
            or read two blocks without verification
              $ mmc rpmb read-block /dev/mmcblk0rpmb 0x02 2 /tmp/block
      mmc rpmb write-block <rpmb device> <address> <256 byte data file> <key file>
            Block of 256 bytes will be written from data file to
            <rpmb device>. Also you can specify '-' instead of key
            file path or data file to read the data from stdin.
            Example:
              $ (awk 'BEGIN {while (c++<256) printf "a"}' | \
                echo -n AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH) | \
                mmc rpmb write-block /dev/mmcblk0rpmb 0x02 - -
```

- 创建加密密码：

```
echo 'Authkeymustbe32byteslength_0000' > keyfile.txt
root@s32g274ardb2:/home# ls -l
-rw-r--r-- 1 root root     32 Mar  9 13:40 keyfile.txt
```

- 烧写密码(注意这个对每一片 eMMC 是一次性的，烧完后不可更改，烧完后需要后启)

```
./mmc rpmb write-key /dev/mmcblk0rpmb keyfile.txt
reboot
```

- 使用密码写入数据到 RPMB

```
echo
'256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.2
56bytedattafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.2
56bytedatafile' > data.txt //生成一个 256 字节的数据文件

root@s32g274ardb2:/home# ls -l

-rw-r--r-- 1 root root    256 Mar  9 13:32 data.txt
```

然后将文件加密写入 rpmb:

```
./mmc rpmb write-block /dev/mmcblk0rpmb 0 data.txt keyfile.txt
```

- 如果密码错误，则写入失败

```
echo 'Authkeymustbe32byteslength_1111' > Wrongkeyfile.txt

root@s32g274ardb2:/home# ./mmc rpmb write-block /dev/mmcblk0rpmb 0 data.txt Wrongkeyfile.txt

RPMB operation failed, retcode 0x0002
```

- 使用密码从 RPMB 中读出数据

```
./mmc rpmb read-block /dev/mmcblk0rpmb 0 1 out.txt keyfile.txt

root@s32g274ardb2:/home# ls -l

-rw------- 1 root root    256 Mar  9 13:38 out.txt

root@s32g274ardb2:/home# cat out.txt

256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.2
56bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.256bytedatafile.25
6bytedatafile.256bytedatafile
```

- 如果使用的密码错，则读出失败

```
./mmc rpmb read-block /dev/mmcblk0rpmb 0 1 out.txt Wrongkeyfile.txt

RPMB MAC missmatch
```