MA35D1 EMMC/SD card 增加分区

NuMicro® 64 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码介绍 MA35D1 的 buildroot 平台，SD card/EMMC 添加主分区、扩展分区及逻辑分区的方法。
BSP 版本	Buildroot_2024
开发平台	NuMaker-HMI-MA35D1-S1 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

MA35D1 的 SD 卡及 EMMC 目前软件不支持 GPT（GUID Partition Table）分区启动，仅使用 MBR（Master Boot Record）分区模式，这种分区模式最多支持 4 个主分区。如果想要更多的分区，就需要在主分区外，再分一个扩展分区，然后在这个扩展分区上再分多个逻辑分区，从而突破 MBR 的 4 分区限制。

1.1 原理

MBR（主引导记录）分区结构主要由三部分组成，446 字节引导程序，64 字节分区表及 2 字节的结束标记，这 512 字节通常在 EMMC 的第一个扇区，其核心作用是检查分区表完整性，定位活动分区等。如表：

偏移	描述	长度（字节）
0	代码区	440（最大 446）
440	选用磁盘标志	4
444	一般为空值，0x0000	2
446	MBR 分区表（四个 16 字节的主分区表入口）	64
511	MBR 结束标志：0x55AA	2

表 1-1 MBR 结构

其中 MBR 分区表（偏移 446 到 550 字节）占 64 字节，共 4 个表项，每个表项各字节定义如下表：

偏移	长度	意义
00H	1	分区状态：00->非活动分区；80->活动分区；其它数值没意义
01H	1	分区起始磁头号（HEAD），用到全部 8 位

02H	2	分区起始扇区号（SECTOR），占据 02H 的位 0~5；该分区的起始磁柱号（CYLINDER）,占据 02H 的位 6~7 和 03H 的全部 8 位
04H	1	文件系统标志，0x83：Linux ext2/ext3，0x0C：FAT32（LBA）
05H	1	分区结束磁头号（HEAD），用到全部 8 位
06H	2	分区结束扇区号（SECTOR），占据 06H 的位 0~5；该分区的结束磁柱号（CYLINDER），占据 06H 的位 6~7 和 07H 的全部 8 位
08H	4	分区起始相对扇区号，用于定位分区在磁盘中的位置
0CH	4	分区总的扇区数，用于计算分区大小

表 1-2 MBR 分区表结构

MBR 分区模式，最多支持 4 个主分区。如果想要更多的分区，就需要在主分区外，再分一个扩展分区，然后在这个扩展分区上可以分多个逻辑分区。

扩展分区表在 MBR 中和主分区一样的格式，可以理解为一种特殊的主分区，每个 MBR 仅允许存在一个扩展分区，必须在其中创建逻辑分区后才能使用，扩展分区内的逻辑分区信息通过 EBR（Extended Boot Record）链表进行管理。

第一个逻辑分区的 EBR 位于扩展分区的头部，其余每个逻辑分区的起始扇区减去 64 KB 的位置包含一个 EBR，占 512 字节，EBR 的前 16 字节，描述当前逻辑分区的起始位置和大小，后 16 个字节指向下一个逻辑分区的 EBR，形成链式结构。

在 MA35D1 上，默认的分区只有 MBR 管理一个主分区（Primary1），用以存储 rootfs，整片 EMMC 存储程序分布如图：



图 1-1 EMMC 默认分区及程序存储

本文将在此基础上，增加三个主分区、一个扩展分区，再在扩展分区上分 3 个逻辑分区，以此为例分区如下图：



图 1-2 EMMC 示例分区结构

增加分区后的 EMMC 存储程序分布如图 1-3：

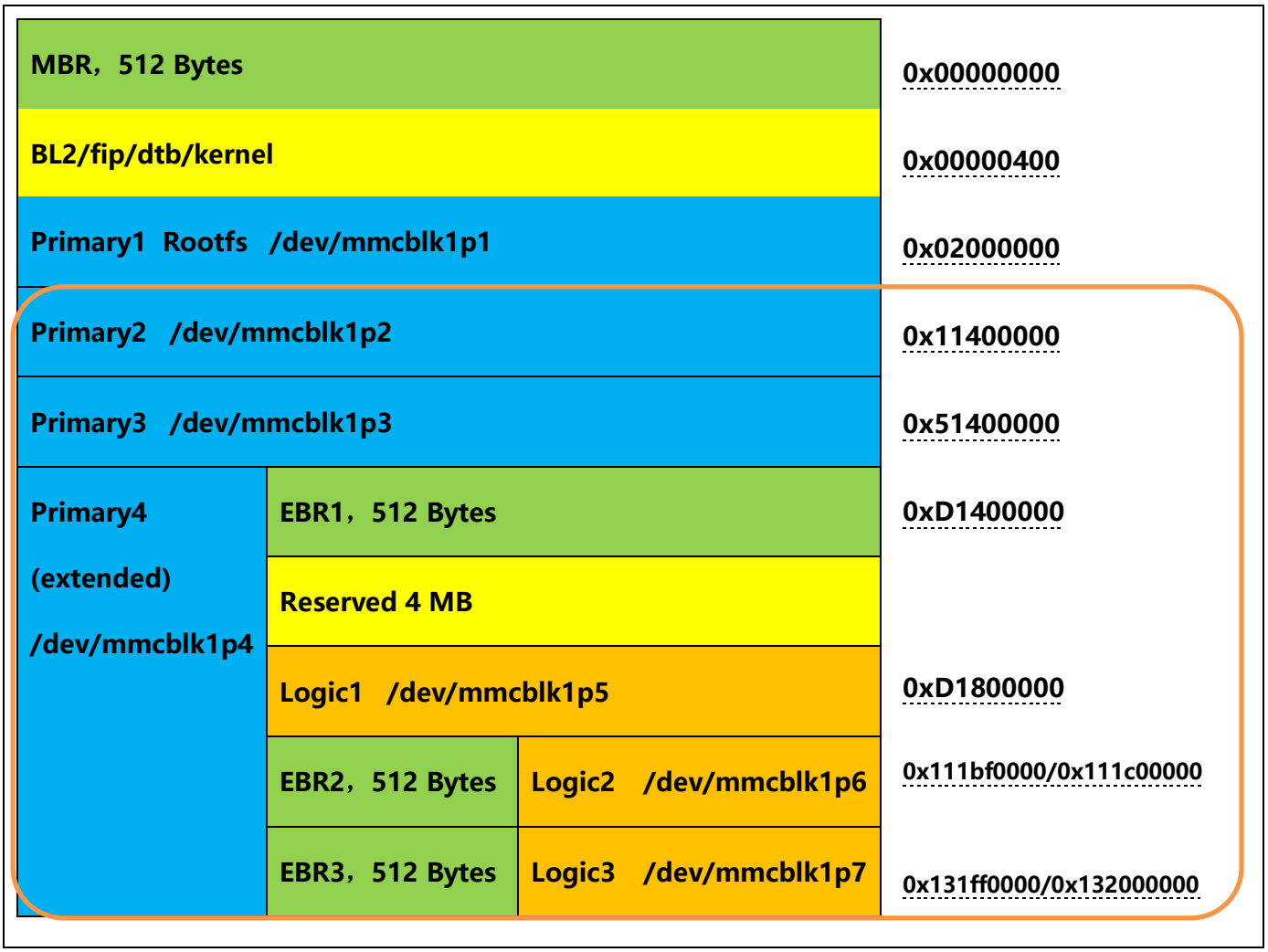


图 1-3 增加分区后 EMMC 程序分布图

图 1-3 中，MBR、BL2/fip/dtb/kernel、rootfs、EBR1、EBR2、EBR3 均需要打包至 pack 烧录，MBR 管理三个主分区及一个扩展分区，EBR 管理三个逻辑分区，需要注意的是 EBR 的位置，EBR1 在扩展分区 offset 为 0 的位置，EBR2 和 EBR3 在管理逻辑分区起始地址减 64 KB 的位置，均占 512 字节。

1.2 执行结果

将添加了新分区表的 pack 镜像烧录至 EMMC 中，查看分区表，执行结果如下图，图中分区 mmcblk1p2~7 即为新加的分区。

```
# fdisk -l
Disk /dev/mmcblk1: 15 GB, 15634268160 bytes, 30535680 sectors
238560 cylinders, 4 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot StartCHS    EndCHS        StartLBA     EndLBA    Sectors  Size Id Type
/dev/mmcblk1p1    512,0,1     1023,3,32        65536     565247     499712  244M 83 Linux
/dev/mmcblk1p2    1023,3,32   1023,3,32       565248    2662399    2097152 1024M 83 Linux
/dev/mmcblk1p3    1023,3,32   1023,3,32      2662400    6856703    4194304 2048M 83 Linux
/dev/mmcblk1p4    1023,3,32   1023,3,32      6856704   23633919   16777216 8192M  f Win95 Ext'd (LBA)
/dev/mmcblk1p5    1023,3,32   1023,3,32      6864896    8962047    2097152 1024M 83 Linux
/dev/mmcblk1p6    1023,3,32   1023,3,32      8970240   10018815    1048576  512M 83 Linux
/dev/mmcblk1p7    1023,3,32   1023,3,32     10027008   12124159    2097152 1024M 83 Linux
```

图 1-4 增加分区后 fdisk 指令执行结果

```
# parted /dev/mmcblk1 print
Model: MMC MMC16G (sd/mmc)
Disk /dev/mmcblk1: 15.6GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      33.6MB  289MB   256MB   primary   ext4
 2      289MB   1363MB  1074MB  primary   ext4
 3      1363MB  3511MB  2147MB  primary   ext4
 4      3511MB  12.1GB  8590MB  extended               lba
 5      3515MB  4589MB  1074MB  logical   ext4
 6      4593MB  5130MB  537MB   logical   ext4
 7      5134MB  6208MB  1074MB  logical   ext4
```

图 1-5 增加分区后 parted 指令执行结果

2. 代码介绍

本节内容将完整介绍如何生成带扩展分区及逻辑分区表的 pack 镜像，分区方案以图 1-3 为准。涉及到的修改文件包括：

- *board/nuvoton/ma35d1/post-image.sh* 、
- *board/nuvoton/ma35d1/nuwriter/pack-sdcard.json*

1. 修改 post-image.sh，确定扩展分区及各逻辑分区的起始地址，分区大小等信息，以 MB 为基本单位，确认大小，不能越界到其它分区。

```
IMAGE_CMD_sdcard()
{
    EXT2_SIZE=$(sed -n -e 's/^BR2_TARGET_ROOTFS_EXT2_SIZE=/ /p' ${BR2_CONFIG} | sed 's/M/
/g' | sed 's/\"/ /g')

    BOOT_SPACE_ALIGNED=$BOOT_SPACE #32MB
    SDCARD_SIZE=$((16*1024*1024*1024)) #EMMC 16G
################################################################################
    MMC_BLK1_START=$BOOT_SPACE  #0x2000000
    MMC_BLK1_END=$(($BOOT_SPACE_ALIGNED+$IMAGE_ROOTFS_ALIGNMENT+$EXT2_SIZE))#32+4+240M

    MMC_BLK2_START=$MMC_BLK1_END #0x11400000
    MMC_BLK2_SIZE=1024 #1G
    MMC_BLK2_END=$(($MMC_BLK2_START+$MMC_BLK2_SIZE))

    MMC_BLK3_START=$MMC_BLK2_END #0x51400000
    MMC_BLK3_SIZE=2048 #2G
    MMC_BLK3_END=$(($MMC_BLK3_START+$MMC_BLK3_SIZE))

    MMC_EXTENED_START=$MMC_BLK3_END #0xd1400000
    MMC_EXTENED_SIZE=8192 #8G
    MMC_EXTENED_END=$(($MMC_EXTENED_START+$MMC_EXTENED_SIZE))

    MMC_LOGIC_RESERVE_SIZE=4 #4M
    MMC_LOGIC1_START=$(($MMC_EXTENED_START+$MMC_LOGIC_RESERVE_SIZE)) #0xd1800000
    MMC_LOGIC1_SIZE=1024 #1G
    MMC_LOGIC1_END=$(($MMC_LOGIC1_START+$MMC_LOGIC1_SIZE))

    MMC_LOGIC2_START=$(($MMC_LOGIC1_END+$MMC_LOGIC_RESERVE_SIZE)) #0x111c00000
    MMC_LOGIC2_SIZE=512 #512M
    MMC_LOGIC2_END=$(($MMC_LOGIC2_START+$MMC_LOGIC2_SIZE))

    MMC_LOGIC3_START=$(($MMC_LOGIC2_END+$MMC_LOGIC_RESERVE_SIZE)) #0x132000000
    MMC_LOGIC3_SIZE=1024 #1G
    MMC_LOGIC3_END=$(($MMC_LOGIC3_START+$MMC_LOGIC3_SIZE))
################################################################################
     if grep -Eq "^BR2_TARGET_MA35D1_SECURE_BOOT=y$" ${BR2_CONFIG}; then
     SDCARD=${BINARIES_DIR}/${IMGDEPLOYDIR}/${IMAGE_BASENAME}-${MACHINE}-
enc.rootfs.sdcard
    fi
```

2. 使用 dd 指令创建一个完整的空镜像，大小为待分区 SD 卡/EMMC 大小，本例为 16 G。再使用 parted 指令对以上镜像根据图 1-3 的方案完成分区，包括主分区、扩展分区，及多个逻辑分区。

```
  # Initialize a sparse file
  dd if=/dev/zero of=${SDCARD} bs=1 count=0 seek=$SDCARD_SIZE &>${NULLDEV}
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s mklabel msdos

################################################################################
  #PRIMARY1
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart primary $MMC_BLK1_START $MMC_BLK1_END

  #PRIMARY2
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart primary $MMC_BLK2_START $MMC_BLK2_END

  #PRIMARY3
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart primary $MMC_BLK3_START $MMC_BLK3_END

  #EXTENDED
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart extended $MMC_EXTENED_START $MMC_EXTENED_END

  #LOGIC1
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart logical $MMC_LOGIC1_START $MMC_LOGIC1_END

  #LOGIC2
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart logical $MMC_LOGIC2_START $MMC_LOGIC2_END

  #LOGIC3
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} -s unit MiB \
mkpart logical $MMC_LOGIC3_START $MMC_LOGIC3_END
################################################################################

# print the result
  ${HOST_DIR}/bin/fakeroot -u ${HOST_DIR}/sbin/parted ${SDCARD} print
```

3. 使用 dd 指令从已分区完成的镜像中读取主分区，扩展分区及逻辑分区分区表，读取位置参考图 1-3，注意 MBR 在 EMMC 最前面 offset 为 0 的位置，EBR1 在扩展分区 offset 为 0 的位置，EBR2 和 EBR3 在逻辑分区起始地址减 64 KB 的位置，均占 512 字节，最终生成 4 个分区表文件。

```
# MBR
dd if=${SDCARD} of=${BINARIES_DIR}/MBR.sdcard.bin \ conv=notrunc,fsync seek=0
count=512 bs=1 &>${NULLDEV}
```

```
#EBR1
dd if=${SDCARD} of=${BINARIES_DIR}/EBR1.sdcard.bin \ conv=notrunc,fsync
skip=$((1024*1024*$MMC_EXTENED_START)) seek=0 \ count=512 bs=1 &>${NULLDEV}

#EBR2
dd if=${SDCARD} of=${BINARIES_DIR}/EBR2.sdcard.bin \ conv=notrunc,fsync
skip=$((1024*1024*$MMC_LOGIC2_START-65536)) \ seek=0 count=512 bs=1 &>${NULLDEV}

#EBR3
dd if=${SDCARD} of=${BINARIES_DIR}/EBR3.sdcard.bin \ conv=notrunc,fsync
skip=$((1024*1024*$MMC_LOGIC3_START-65536)) \ seek=0 count=512 bs=1 &>${NULLDEV}

printf "EBR1 addr: %x\n" $((1024*1024*$MMC_EXTENED_START))
printf "EBR2 addr: %x\n" $((1024*1024*$MMC_LOGIC2_START-65536))
printf "EBR3 addr: %x\n" $((1024*1024*$MMC_LOGIC3_START-65536))
```

4. 在 pack-sdcard.json 脚本添加上面新生成的逻辑分区的分区表文件，offset 字段按图 1-3 规划，即上面读取的位置填写。

```
{
  "offset": "0x2400000",
  "file": "rootfs.ext4",
  "type": 0
},
{
  "offset": "0xD1400000",
  "file": "EBR1.sdcard.bin",
  "type": 0
},
{
  "offset": "0x111bf0000",
  "file": "EBR2.sdcard.bin",
  "type": 0
},
{
  "offset": "0x131ff0000",
  "file": "EBR3.sdcard.bin",
  "type": 0
}
```

5. NuWriter 会使用最新的 json 脚本，压缩生成带有分区表文件的 pack 镜像。

```
#pack all file in json
${HOST_DIR}/bin/nuwriter.py -p ${NUWRITER_TARGET}/pack-sdcard.json;
cp pack/pack.bin pack-${IMAGE_BASENAME}-${MACHINE}-sdcard.bin;
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本

  - Buildroot_2024

- NuWriter 版本

  - NuWriter GUI v1.09

3.2 硬件需求

- 电路组件

  - NuMaker-HMI-MA35D1-S1 V1.0

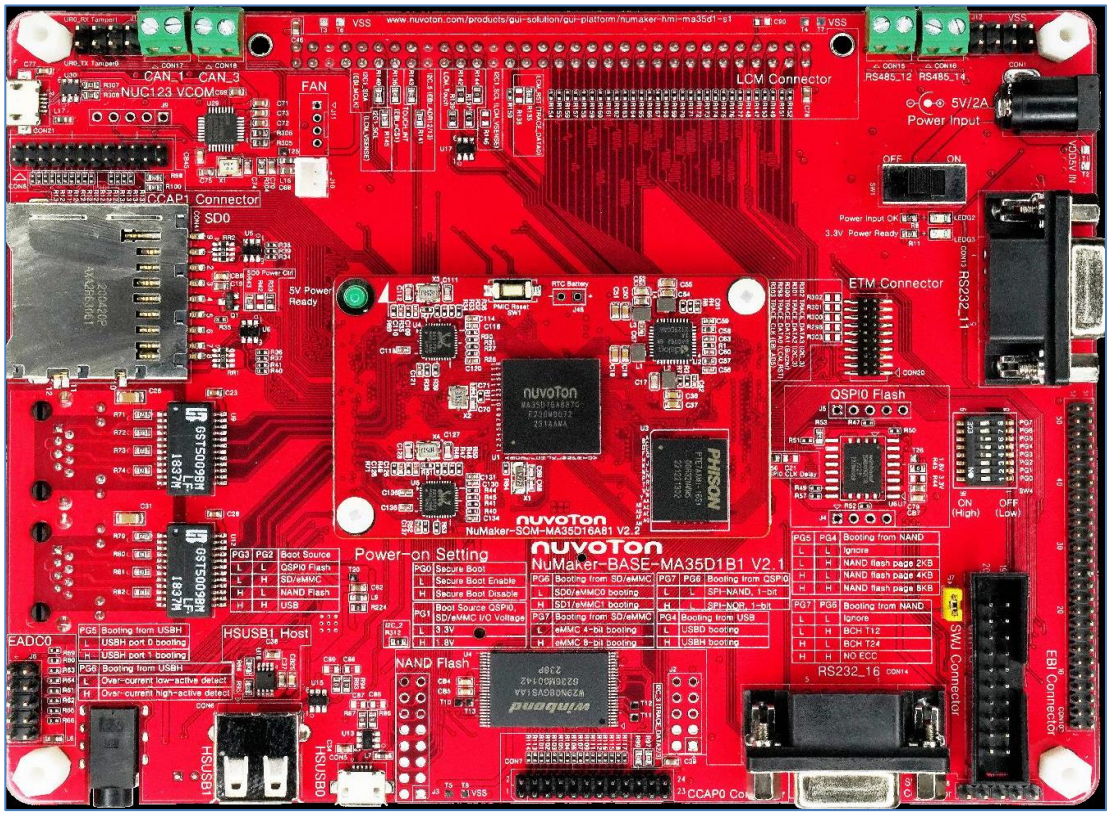


图 3-1 NuMaker-HMI-MA35D1-S1 V1.0

4. 目录信息

目录结构如下图所示：

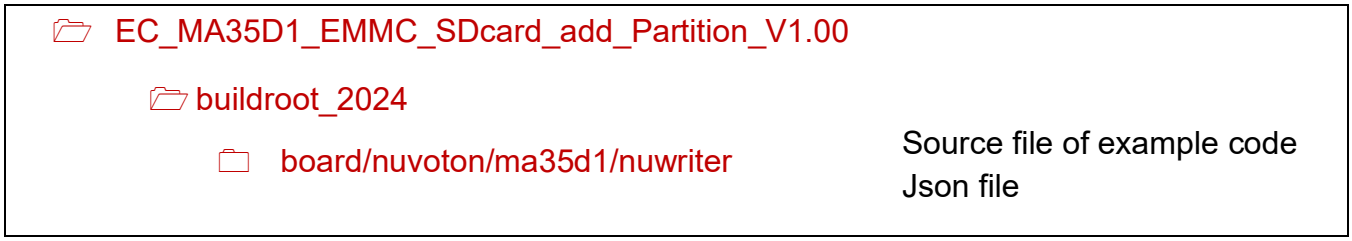


图 4-1 目录信息

5. 范例程序执行

1. 根据第 2 章修改完成 *board/nuvoton/ma35d1/post-image.sh*、*board/nuvoton/ma35d1/nuwriter/pack-sdcard.json* 两个文件后，进入 buildroot 根目录下，执行 make，最后会显示分区方案在虚拟机下的执行结果，以及 EBR 值，如下图。

```
>>>   Executing post-image script board/nuvoton/ma35d1/post-image.sh
Model:  (file)
Disk /home/user/buildroot_2024/output/images/core-image-buildroot-ma35d1-som-512m.rootfs.sdca
rd: 17.2GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      33.6MB  289MB   256MB   primary
 2      289MB   1363MB  1074MB  primary
 3      1363MB  3511MB  2147MB  primary
 4      3511MB  12.1GB  8590MB  extended               lba
 5      3515MB  4589MB  1074MB  logical
 6      4593MB  5130MB  537MB   logical
 7      5134MB  6208MB  1074MB  logical

EBR1 addr: d1400000
EBR2 addr: 111bf0000
EBR3 addr: 131ff0000
Generate output image(s) in directory 0415-162804164311 complete
Generate pack file in directory 0415-162805162393 complete
```

图 5-1 buildroot 下编译信息

2. 编译完成后，output/images 路径下将生成包含 MBR 及 EBR 的 pack 镜像，通过 NuWriter 烧录至开发板。

3. 启动后执行结果如图 5-2，图 5-3，图中分区 mmcblk1p2~7 即为新加的分区，其中 mmcblk1p4 为扩展分区，不能实际存储数据。

```
# fdisk -l
Disk /dev/mmcblk1: 15 GB, 15634268160 bytes, 30535680 sectors
238560 cylinders, 4 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot StartCHS    EndCHS        StartLBA     EndLBA    Sectors  Size Id Type
/dev/mmcblk1p1    512,0,1     1023,3,32        65536     565247     499712  244M 83 Linux
/dev/mmcblk1p2    1023,3,32   1023,3,32       565248    2662399    2097152 1024M 83 Linux
/dev/mmcblk1p3    1023,3,32   1023,3,32      2662400    6856703    4194304 2048M 83 Linux
/dev/mmcblk1p4    1023,3,32   1023,3,32      6856704   23633919   16777216 8192M  f Win95 Ext'd (LBA)
/dev/mmcblk1p5    1023,3,32   1023,3,32      6864896    8962047    2097152 1024M 83 Linux
/dev/mmcblk1p6    1023,3,32   1023,3,32      8970240   10018815    1048576  512M 83 Linux
/dev/mmcblk1p7    1023,3,32   1023,3,32     10027008   12124159    2097152 1024M 83 Linux
```

图 5-2 增加分区后 fdisk 执行结果

```
# parted /dev/mmcblk1 print
Model: MMC MMC16G (sd/mmc)
Disk /dev/mmcblk1: 15.6GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      33.6MB  289MB   256MB   primary   ext4
 2      289MB   1363MB  1074MB  primary   ext4
 3      1363MB  3511MB  2147MB  primary   ext4
 4      3511MB  12.1GB  8590MB  extended               lba
 5      3515MB  4589MB  1074MB  logical   ext4
 6      4593MB  5130MB  537MB   logical   ext4
 7      5134MB  6208MB  1074MB  logical   ext4
```

图 5-3 增加分区后 parted 指令执行结果

4. 以上主分区及逻辑分区可以使用 mkfs.ext4 格式化之后，挂载至合适目录使用。

5. 使用本文方案时，可灵活选择，如果需要的分区数小于或等于 4 个，则直接分几个主分区，不需要再分扩展分区和逻辑分区。

6. 本文方案以 MA35D1 平台为例完成，也适用于 MA35 series 其他平台，可根据实际需求参考使用。

6. 修订记录

Date	Revision	Description
2026.04.15	1.00	初始发布

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.