# Adding Partitions with MA35D1 eMMC/SD Card

Example Code Introduction for 64-bit NuMicro® Family

## Document Information

| | |
|---|---|
| Application | This sample code introduces the method of adding primary partitions, extended partitions, and logical partitions on the SD card/eMMC on the MA35D1 Buildroot platform. |
| BSP Version | Buildroot_2024 |
| Hardware | 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. Overview

The current software of MA35D1's SD card and eMMC does not support GPT (GUID Partition Table) partition booting and only uses MBR (Master Boot Record) partition mode. This partition mode supports up to 4 primary partitions. If more partitions are needed, an extended partition should be created outside the primary partitions, and then multiple logical partitions can be created on this extended partition, thus breaking through the 4-partition limitation of MBR.

## 1.1 Principle

The MBR partition structure primarily consists of three components: a 446-byte bootloader, a 64-byte partition table, and a 2-byte end marker. These 512 bytes are typically found in the first sector of an eMMC device. Its core function is to verify the integrity of the partition table and locate the active partition, among other tasks, as illustrated below.

| Offset | Description | Size (Byte) |
|---|---|---|
| 0 | Code area | 440 (max 446) |
| 440 | Selected disk flag | 4 |
| 444 | Empty value, 0x0000 | 2 |
| 446 | MBR partition (Four entries of 16-byte primary partition table) | 64 |
| 511 | MBR finish flag: 0x55AA | 2 |

Table 1-1 MBR Structure

The MBR partition table (offset 446 to 550 bytes) occupies 64 bytes, with a total of 4 entries. The definition of each byte in each entry is as follows.

| Offset | Size | Meaning |
|---|---|---|
| 00H | 1 | Partition status: 00->Inactive partition; 80->Active partition;<br>Other values are meaningless |
| 01H | 1 | Partition starting head number (HEAD), using all 8 bits |

| | | |
|---|---|---|
| 02H | 2 | Partition start sector number (SECTOR), occupying bits 0-5 of 02H; the start cylinder number (CYLINDER) of this partition, occupying bits 6-7 of 02H and all 8 bits of 03H |
| 04H | 1 | File system flag, 0x83: Linux ext2/ext3, 0x0C: FAT32 (LBA) |
| 05H | 1 | Partition end head number (HEAD), using all 8 bits |
| 06H | 2 | The sector number (SECTOR) of the partition end, occupying bits 0~5 of 06H; the cylinder number (CYLINDER) of the partition end, occupying bits 6~7 of 06H and all 8 bits of 07H |
| 08H | 4 | The starting relative sector number of the partition, used to locate the partition's position on the disk |
| 0CH | 4 | The total number of sectors in the partition, used to calculate the partition size |

Table 1-2 MBR Partition Structure

The MBR partitioning scheme supports up to four primary partitions. If more partitions are needed, an extended partition must be created outside the primary partitions. Multiple logical partitions can then be created on this extended partition.

The extended partition table has the same format as the primary partition in the Master Boot Record (MBR) and can be understood as a special type of primary partition. Each MBR allows only one extended partition, which must be used after creating logical partitions within it. The logical partition information within the extended partition is managed through the Extended Boot Record (EBR) linked list.

The EBR of the first logical partition is located at the head of the extended partition, and the first 64 KB of the starting sector of each remaining logical partition contains an EBR, occupying 512 bytes. The first 16 bytes of the EBR describe the starting position and size of the current logical partition, while the last 16 bytes point to the EBR of the next logical partition, forming a chained structure.

In MA35D1, the default partition only has one primary partition (Primary1) managed by MBR, which is used to store the rootfs. The distribution of the entire eMMC storage program is shown below.

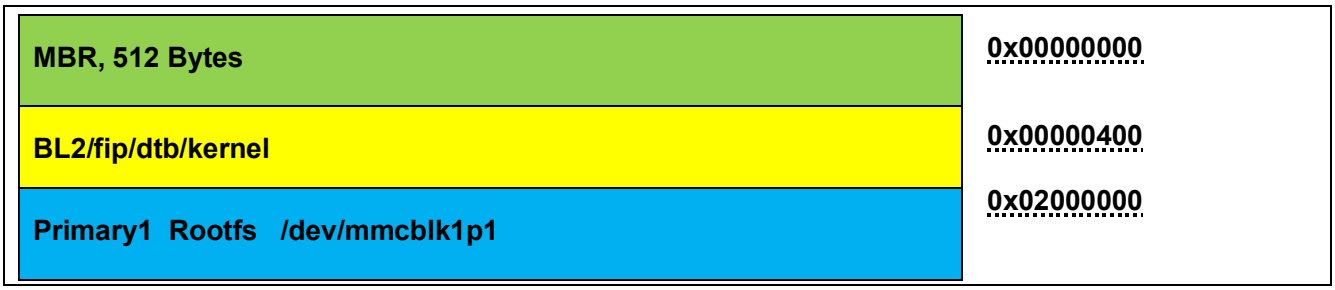



Figure 1-1 eMMC Default Partition and Program Storage

Based on above example, this article will add three primary partitions and one extended partition, and then create three logical partitions on the extended partition. The partition layout is as follows.

Disk
├── MBR : The locations of the primary partition 1/2/3, and extended partition are recorded.
  ├── primary partition 1 (correspond to /dev/sda1)
  ├── primary partition 2 (correspond to /dev/sda2)
  ├── primary partition 3 (correspond to /dev/sda3)
  └── extended partition (correspond to /dev/sda4, No actual data is stored; the 0th sector of the partition stores EBR1)
    ├── logic partition 1 (EBR 1 manage, correspond to /dev/sda5)
    ├── logic partition 2 (EBR 2 manage, correspond to /dev/sda6, EBR2 is stored at the starting address of partition6 minus 64 KB)
    └── logic partition 3 (EBR 3 manage, correspond to /dev/sda7, EBR3 is stored at the starting address of partition7 minus 64 KB)

Figure 1-2 eMMC Example Partition Structure

The distribution of eMMC storage programs after adding partitions is shown in Figure 1-3:

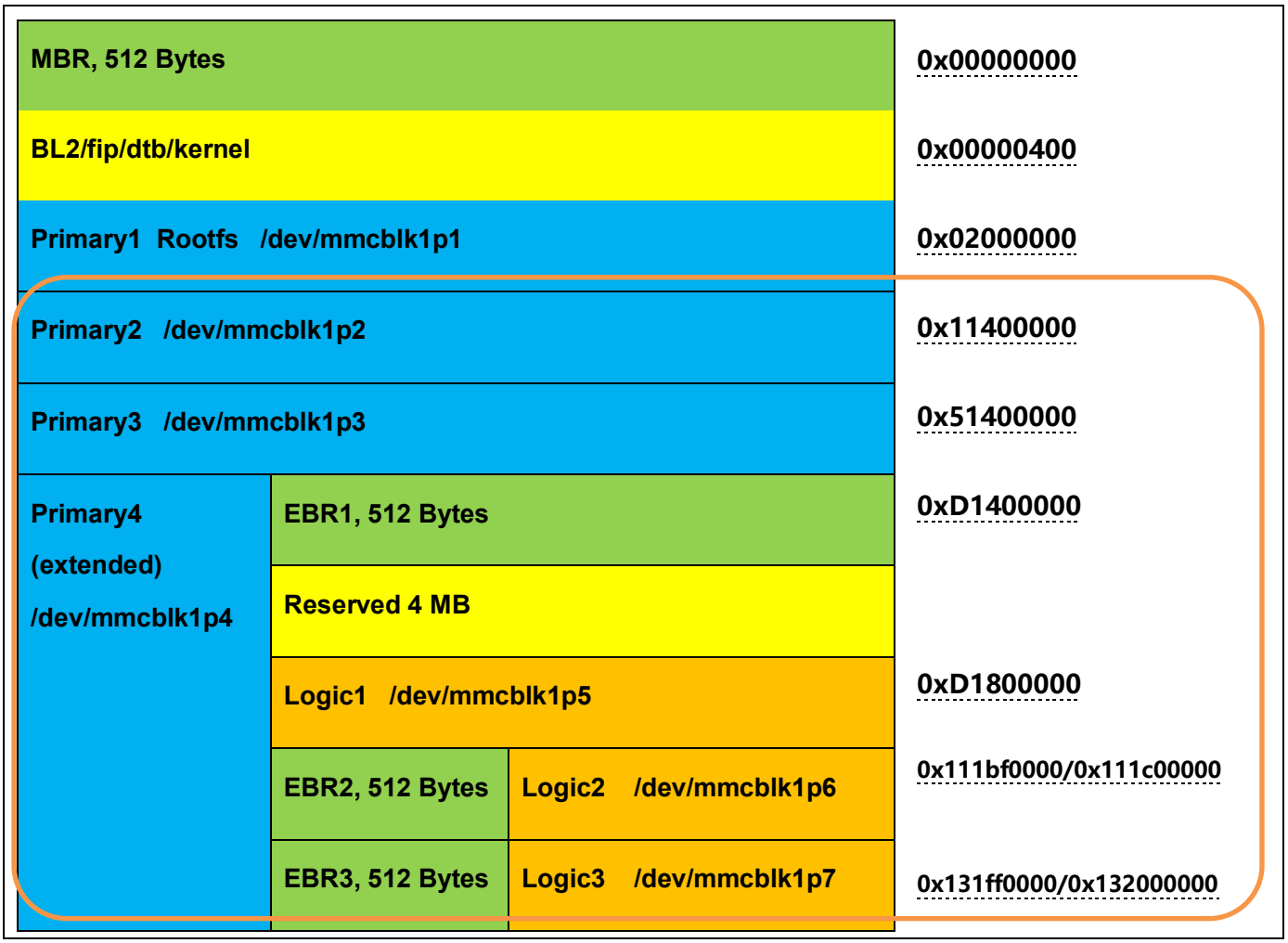



Figure 1-3 eMMC Program Distribution Diagram

In Figure 1-3, MBR, BL2/fip/dtb/kernel, rootfs, EBR1, EBR2, and EBR3 all need to be packaged into the pack image. MBR manages three primary partitions and one extended partition, while EBR manages three logical partitions. It is important to note the location of EBR. EBR1 is located at the offset of 0 in the extended partition, and EBR2 and EBR3 are stored at the location where the starting address of the logical partition is reduced by 64 KB, each occupying 512 bytes.

## 1.2 Demo Result

Program the pack image with the newly added partition table to the eMMC, view the partition table, and the execution result is shown below. The partitions mmcblk1p2~7 in Figure 1-4 are the newly added partitions.

```
# 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
```

Figure 1-4 Execution Result of the Fdisk Command after Adding Partitions

```
# 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
```

Figure 1-5 Execution Result of the Parted Command after Adding Partitions

# 2. Code Description

This section will provide a comprehensive introduction to how to generate a pack image with extended partitions and logical partition tables. The partition scheme is based on Figure 1-3. The modification files involved include:

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

1. Modify post-image.sh to determine the starting address, partition size, and other information of the extended partition and each logical partition, using MB as the basic unit. Confirm the size and ensure it does not cross over into other partitions.

```
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. Use dd command to create a complete empty image with a size equal to the size of the SD card/eMMC to be partitioned, which in this case is 16 GB. Then, use the parted command to partition the above image according to the scheme shown in Figure 1-3, including a primary partition, an extended partition, and multiple logical partitions.

```
  # 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. Use dd command to read the primary partition, extended partition, and logical partition tables from the partitioned image. Refer to Figure 1-3 for the reading locations. Note that the MBR is located at the very beginning of the eMMC with an offset of 0, EBR1 is located at the beginning of the extended partition with an offset of 0, and EBR2 and EBR3 are located at the beginning of the logical partition, 64 KB before its starting position, each occupying 512 bytes. Finally, four partition table files are generated.

```
# 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. Add the partition table file of the newly generated logical partition to the pack-sdcard.json script, with the offset field planned according to Figure 1-3.

```
{
  "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 will use the latest JSON script to compress and generate a pack image with a partition table file.

```
#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. Software and Hardware Requirements

## 3.1 Software Requirements

- BSP version
  - Buildroot_2024
- NuWriter Version
  - NuWriter GUI v1.09

## 3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1-S1 V1.0

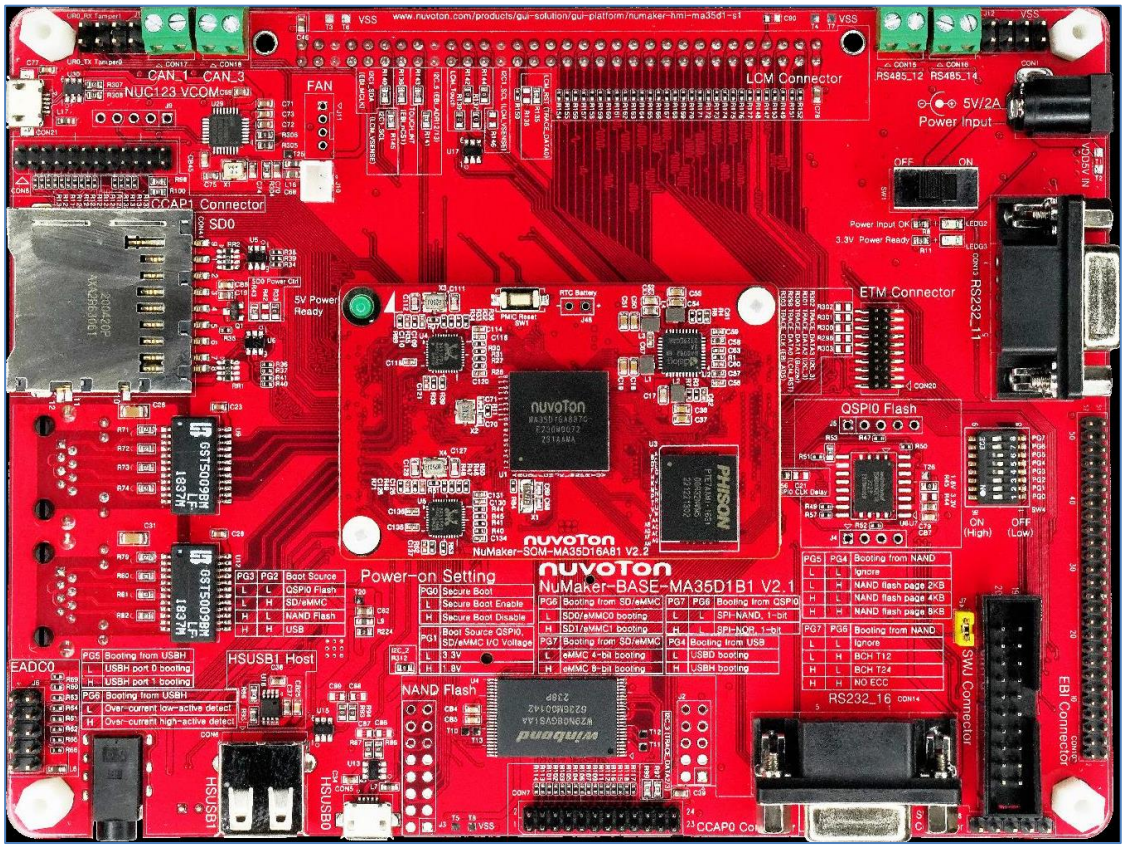



Figure 3-1 NuMaker-HMI-MA35D1-S1 Board

# 4. Directory Information

The directory structure is shown below.

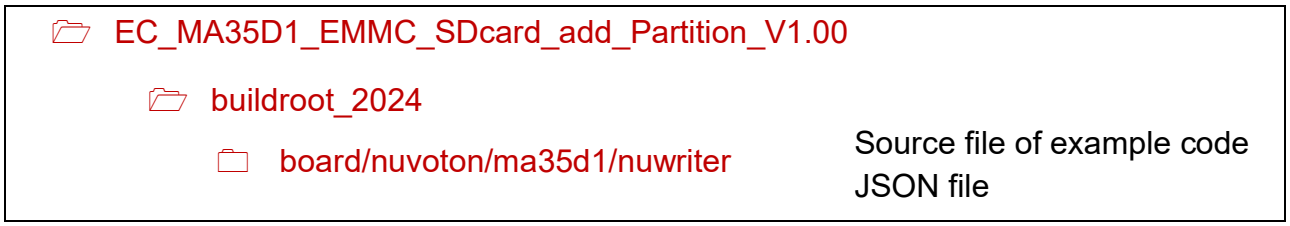



Figure 4-1 Directory Structure

# 5. Example Code Execution

1. Modify board/nuvoton/ma35d1/post-image.sh and board/nuvoton/ma35d1/nuwriter/pack-sdcard.json based on the description in Chapter 2, navigate to the Buildroot root directory and execute "make". Finally, the execution result of the partition scheme under the virtual machine as well as the EBR value will be displayed, as shown below.

```
>>>   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
```

Figure 5-1 Compilation Information under Buildroot

2. After compilation, a pack image containing MBR and EBR will be generated in the *output/images* directory, which can be programmed to the evaluation board using NuWriter.

3. The execution results after startup are shown in Figure 5-2 and Figure 5-3. The partitions mmcblk1p2~7 in the figures are newly added partitions, of which mmcblk1p4 is an extended partition and cannot actually store data.

```
# 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
```

Figure 5-2 Execution Result of the Fdisk Command after Adding Partitions

```
# 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
```

Figure 5-3 Execution Result of the Parted Command after Adding Partitions

4. After formatting the above primary partitions and logical partitions with mkfs.ext4, they can be mounted to appropriate directories for use.

5. When using the scheme presented in this article, you can choose flexibly. If the number of partitions required is less than or equal to 4, you can directly create several primary partitions without further creating extended partitions or logical partitions.

6. The solution presented in this article is exemplified using the MA35D1 platform, but it is also applicable to other platforms in the MA35 series. It can be referenced and used according to actual needs.

# 6. Revision History

| Date | Revision | Description |
|---|---|---|
| 2026.04.15 | 1.00 | Initial version. |

## 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.*