CPE

CommEngine eMMC Support

Application Note

For a comprehensive list of changes to this document, see the Revision History.

Table of Contents

CommEngine eMMC Support

1 Scope

This document provides details regarding the current and future support for Embedded Multi-Media Card (eMMC) devices in the CommEngine codebase. This document is applicable to CommEngine release versions 5.02L03 and newer. The eMMC devices referred to in this document are based on the eMMC JEDEC standard version 5.1.

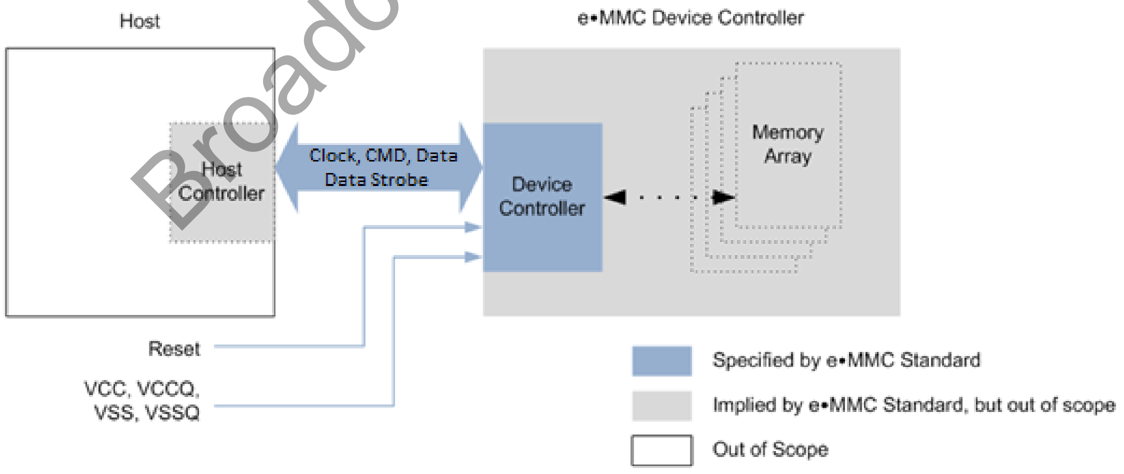
2 Related Documents

Document (or Item) Name		*Number*	*Source*
[1]	*JEDEC Standard: Embedded Multi-Media Card (eMMC) Electrical Standard (5.1)*	–	https://www.jedec.org/standards-documents/technology-focus-areas/flash-memory-ssds-ufs-emmc/e-mmc
[2]	*Simplified SD Host Controller Spec*	–	https://www.sdcard.org/developers/overview/host_controller/simple_spec/
[3]	*BCM63138 AHB Subsystem*	63138-DS1xx	docSAFE

3 Introduction to eMMC

Embedded MultiMediaCard (eMMC) devices are managed flash memories that consist of an embedded flash controller and a high-density flash memory array. Data transfers between an eMMC host controller and eMMC devices take place over a block based eMMC interface as shown in Figure 1.

Figure 1: eMMC System Overview



3.1 eMMC Device Interface

This section provides details regarding the internal structure of eMMC devices and the embedded Broadcom host controller.

- eMMC Bus Protocol: Communication between the host controller and the eMMC device takes place using a message-based protocol. Commands and responses are both handled by a bi-directional CMD line, while data transfers occur over an 8-bit (max) data bus.
- Data transfer sizes: Data transfers occur in multiples of native sector sizes. eMMC devices have native sector sizes of either 512 or 4K bytes. However, all 4K byte sector based devices ship with emulation mode enabled, thereby functioning exactly like 512-byte sector based devices. Emulation mode can be disabled (a device can be forced to operate in native mode) by using that eMMC internal register settings.
- Addressing: For eMMC devices <2 Gb, byte addressing is used. For eMMC devices with higher densities, sector based addressing is used, with a 512-bytes sector size (regardless of native sector size). Devices with 4 Kb sector sizes, operating in native mode, require all their 512-byte sector based addresses to be 4Kb aligned, and all data transfers must occur in 4 Kb size multiples.
- Data Transfer Speeds: eMMC devices support a variety of bus speed modes, with each mode being backward compatible with the previous one. Table 1 shows all the available speed modes.

Table 1: eMMC Speed Modes

Name	Data rate	Bus Frequency	Max. Data Transfer Rate
Legacy	Single	26 MHz	26 MBps
High-Speed SDR	Single	52 MHz	53 MBps
High-Speed DDR	Dual	52 MHz	104 MBps
HS200	Single	200 MHz	200 MBps
HS400	Dual	200 MHz	400 MBps

NOTE: Speed Mode availability changes depending on the Soc, eMMC device and board design. Please contact your FAE to get a list of supported speed modes for your exact implementation.

3.2 eMMC Memory Organization

The memory area inside eMMC devices is divided into several fixed and configurable physical partitions:

- Boot Partitions: These partitions are accessible while booting from eMMC and thus are used for storing first stage boot loaders. Data from boot partitions is streamed out sequentially on every clock cycle when the eMMC device is operating in boot mode.
- RPMB Partitions: Replay Protected Memory Block partitions offer several security related features pertaining to authentication. RPMB partition data is protected against unauthorized access by requiring an authentication key.
- User Data Partition: This is the primary storage area used in most eMMC implementations. This area is used for data, root filesystem and kernel images and is usually logically partitioned using either GPT or MBR partitioning schemes.
- General Purpose Partitions: These are dynamically created partitions (at the expense of the User Data partition space). The main use case for using these partitions is the ability to enable certain enhanced features (such as better reliability) on them.

The Boot, RPMB, and User Data partitions are factory created and cannot be deleted. Before issuing and read or write commands, the host must explicitly select which partition is to be accessed by updating internal eMMC device registers.

3.3 eMMC Host Interface

Broadcom SoC's have an embedded eMMC host interface block that consists of an eMMC host controller and a custom eMMC boot controller.

Check with the SoC's data sheet to determine the highest speed supported by the eMMC Host controller.

3.3.1 eMMC Host Controller Interface

The eMMC host controller register interface conforms to the standardized SD Host Controller Specification, (see Related Documents). The host controller interface is responsible for:

- Translating eMMC commands into bus eMMC bus transactions.
- Retrieving eMMC command responses.
- Setting up DMAs for incoming/outgoing data.

3.3.2 eMMC Boot Controller

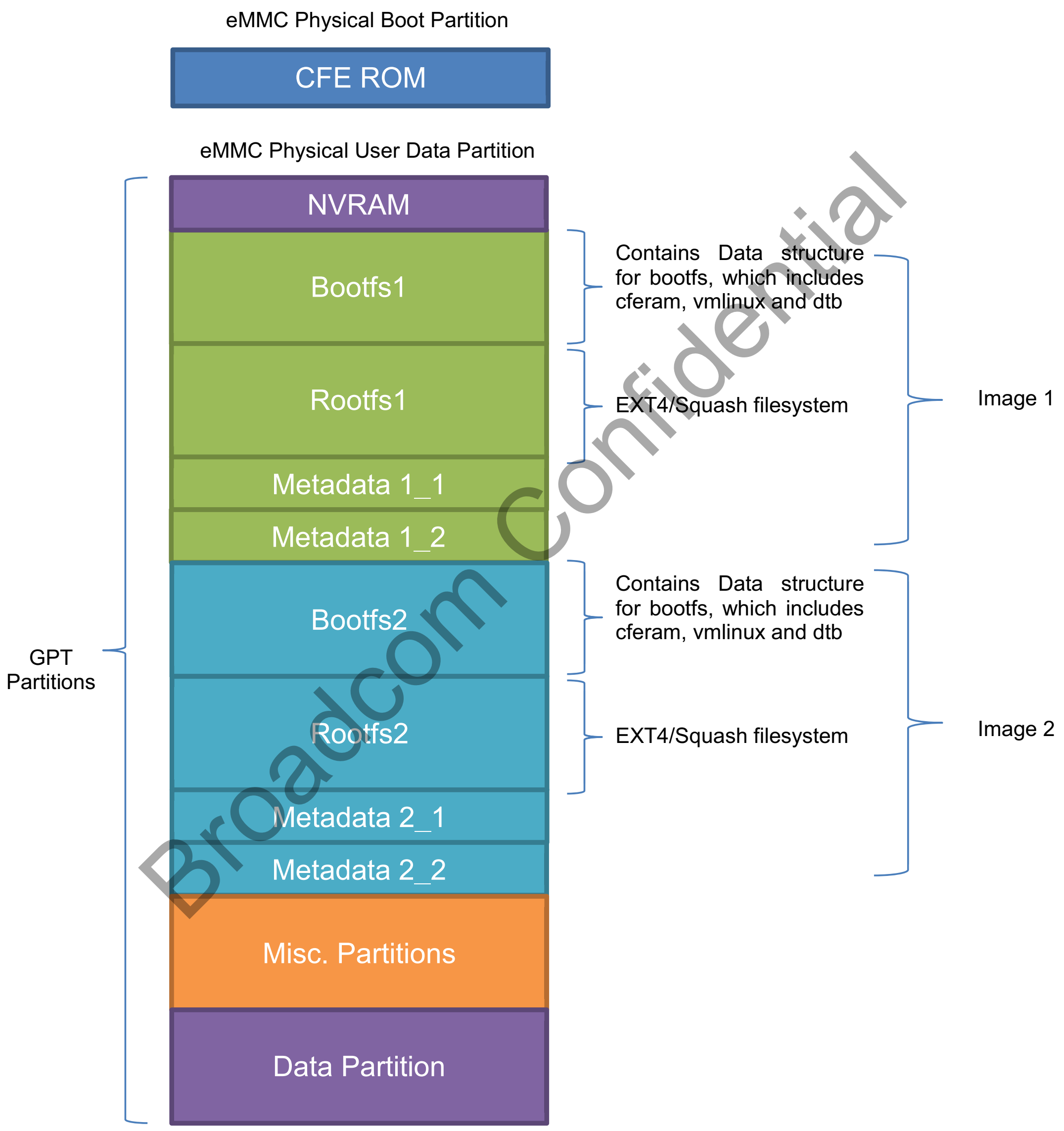
Broadcom's custom eMMC boot controller coordinates all access to the eMMC Host Controller interface during the early boot process. For more information regarding the eMMC boot controller please contact Broadcom Customer Support.

4 eMMC Images and Flash Layout

4.1 Flash Layout

eMMC flash layout mimics the layout for the pure UBI implementation. Instead of UBI devices and volumes, GPT logical partitions are used to store bootfs, rootfs, and metadata. The proposed flash layout is shown in Figure 2.

Figure 2: eMMC Flash Layout


eMMC Physical Boot Partition
CFE ROM
eMMC Physical User Data Partition
NVRAM
Bootfs1
Rootfs1
Metadata 1_1
Metadata 1_2
Bootfs2
Rootfs2
Metadata 2_1
Metadata 2_2
Misc. Partitions
Data Partition
GPT Partitions
Contains Data structure for bootfs, which includes cferam, vmlinux and dtb
EXT4/Squash filesystem
Image 1
Contains Data structure for bootfs, which includes cferam, vmlinux and dtb
EXT4/Squash filesystem
Image 2

4.2 eMMC Images

4.2.1 Image Types

Due to the inherent physical partitioning present in eMMC devices, a single image format is not possible for both flash programmers and CFE/Linux firmware update. Instead eMMC images are available in two formats:

- Tagged/Headered Images: The following images are to be used for firmware upgrades via CFE or Linux:
  - `cfe_fs_kernel_emmc_<rootfs>`
    This image contains the cferom, bootfs, and rootfs and is to be used for runtime image upgrades.
  - `fs_kernel_emmc_<rootfs>`
    This image contains the bootfs and rootfs only, and is to be used for runtime image upgrades.
- Whole flash Images: The following images are to be used with flash programmers only and contain full GPT partition tables along with the image data. There are separate images for the eMMC boot and user data physical partitions.
  - `<profile>_emmc_bootpartition_flash_image_<board>.w`
    This image contains the cferom binary and is to be written to the physical boot partition of the eMMC device via a flash programmer.
  - `<profile>_emmc_datapartition_flash_image_<board>_<rootfs>.w`
    This image consists of a valid GPT partition table along with GPT partitions containing the image data. This image is to be written to the physical user data partition of the eMMC device via a flash programmer.

NOTE: The tagged image format images DO NOT contain any partitioning information and therefore CANNOT be used with flash programmers. For programming via a flash programmer, the programmer must initialize/erase the eMMC device first, and then program the appropriate whole flash "`.w`" images to the corresponding physical partitions.

Tagged images have an image header containing offsets to all the relevant image contents, appended to the start of the image. Figure 3 shows the eMMC image format.

Figure 3: eMMC Tagged Image Format

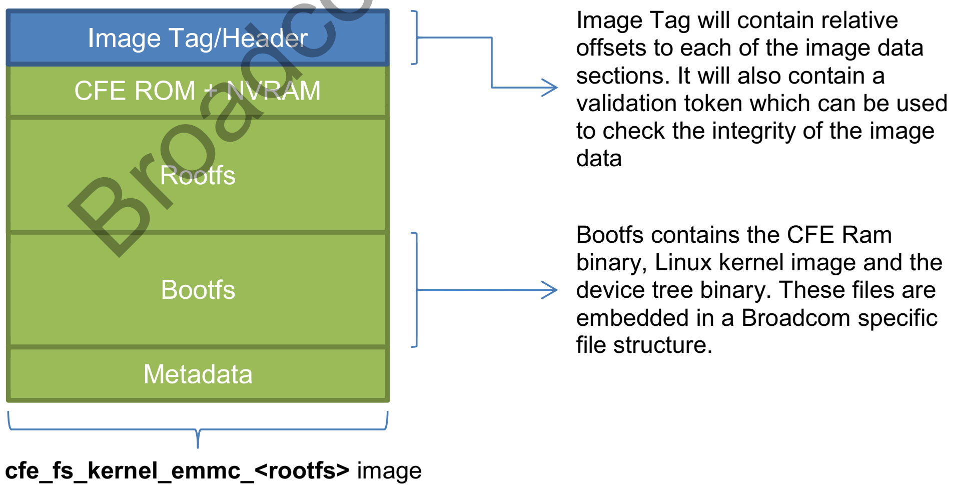
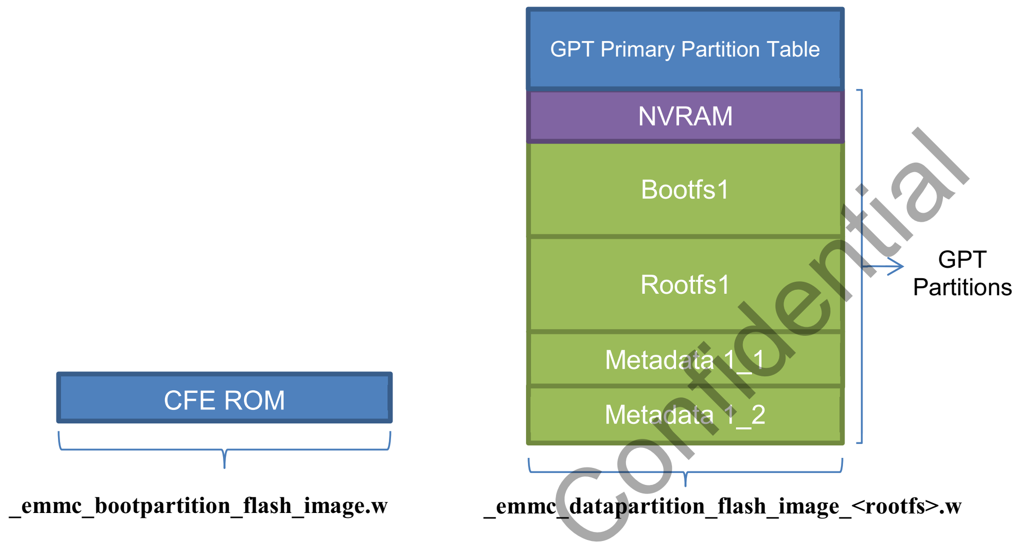


Image Tag/Header
CFE ROM + NVRAM
Rootfs
Bootfs
Metadata
Image Tag will contain relative offsets to each of the image data sections. It will also contain a validation token which can be used to check the integrity of the image data
Bootfs contains the CFE Ram binary, Linux kernel image and the device tree binary. These files are embedded in a Broadcom specific file structure.
cfe_fs_kernel_emmc_<rootfs> image

4.2.1.1 Whole Image Format

Whole flash images are to be used exclusively by flash programmers. These images are binary data which need to be burned as is, from offset zero, into the eMMC physical boot and user data partitions. The whole image for the eMMC physical user data partition contains a primary GPT partition table along with all of the image data at the proper partition offsets. The whole image for the eMMC physical boot partition contains the CFEROM binary. Figure 4 shows the eMMC whole image formats.

Figure 4: eMMC Whole Image Formats for BOOT and DATA Partitions


GPT Primary Partition Table
NVRAM
Bootfs1
Rootfs1
Metadata 1_1
Metadata 1_2
GPT Partitions
CFE ROM
_emmc_bootpartition_flash_image.w
_emmc_datapartition_flash_image_<rootfs>.w

4.2.2 RootFS Options

Two main file system options are available and are built with every eMMC enabled build:

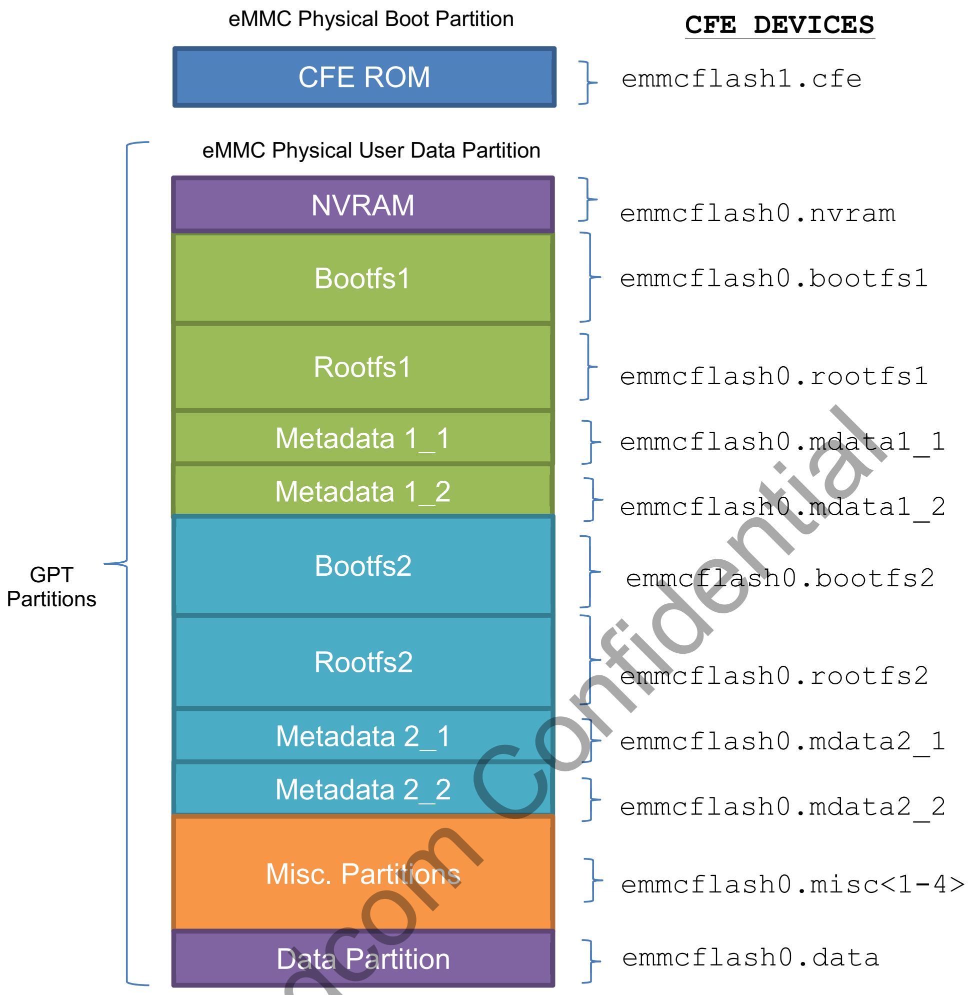
- SquashFS: For a read-only rootfs, with a smaller image size
- Ext4: For a RO/RW rootfs, with journaling support, and a much larger image size

5 eMMC Support in CFE RAM

The CFE source code has a CFE device framework, where cfe_device structures can be associated with specific memory ranges of flash devices. These CFE devices also have associated common file operations which in turn hook into the CFE's eMMC driver code. Once these CFE devices are created, file I/O can be performed on these devices in order to read/write/ delete image data.

The CFE has the capability to read GPT partition tables present at the beginning or end of the eMMC physical user data partition. Once the partition table is parsed, individual partitions are mapped onto cfe_device structures. This results in the creation of CFE devices which correspond directly to eMMC logical partitions. These CFE devices can then be manipulated using a host of eMMC related commands. Figure 5 shows how the eMMC logical partitions map to CFE devices.

Figure 5: Mapping eMMC Partitions To CFE Devices



5.1 eMMC Partition Management in CFE

For the CFE to create/manage GPT partitions in the eMMC user data physical partition, we need to be able to write primary and backup GPT partition tables to the eMMC device. These partition tables are written to special CFE devices emmcflash0.gpt0 and emmcflash0.gpt1. These CFE devices are automatically created and are mapped to the first and last few blocks on the eMMC's user data physical partition, ensuring that the partition tables are always written to the proper locations on the eMMC.

5.2 eMMC CFE Commands

In addition to the general commands for flashing new images and setting CFE parameters, new eMMC commands have been added to the CFE menu. The new commands are shown in Table 2.

Table 2: eMMC Related CFE Commands

Name	Description	Arguments
showdevs	List all CFE devices	None
emmcfmtgpt	Create eMMC GPT partitions	Partition sizes in Kb
emmcdmpgpt	Dump eMMC GPT partition config	–
emmci	Display essential eMMC device information	None
emmcea	Erase all eMMC GPT partitions	None
emmcei	Erase all partitions belonging to an emmc image	Image number
emmcep	Erase specific partition	Partition name
emmcspw	Set a 32-bit word in an eMMC partition	Partition name, address, value
emmcdpw	List 32-bit words from an eMMC partition	Partition name, address, count
emmcdbfs	List the bootfs entries from an eMMC partition	(debug) Partition name, optional file name
emmcgi	Put the eMMC device in Idle mode	(debug) None
emmcr	Boot a specific eMMC Linux image	(debug) Image number (1 or 2)

6 eMMC Support in Linux

6.1 Build

eMMC enabled Linux images can be built by using build profiles generated by the maketargets utility and the `EMMC.arch` file.

6.2 Source Code and Utilities

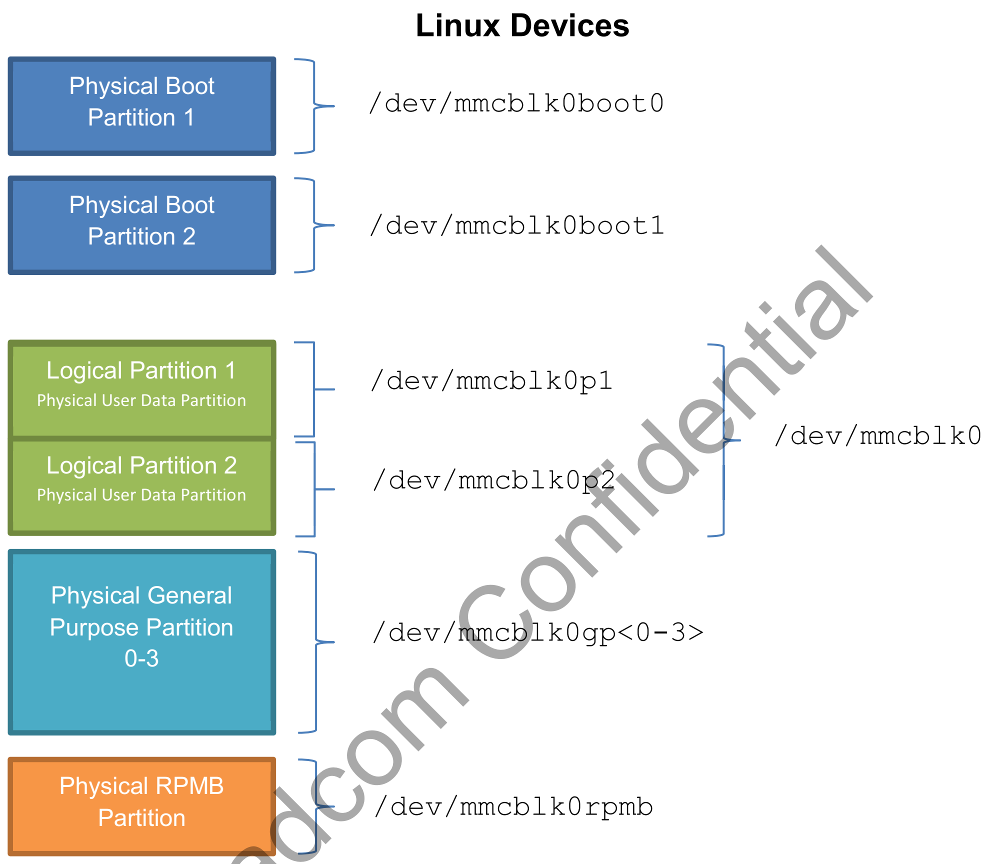
eMMC devices are natively supported in Linux by the SD Host Controller Interface (SDHCI) framework. CommEngine currently has the following Linux related eMMC code:

- eMMC driver: A thin driver for the Broadcom eMMC host controller has been added in kernel/linux-4.1/drivers/mmc/host/sdhci-bcm63xx.c. The driver simply invokes SDHCI framework functions.
- eMMC device tree nodes: All eMMC controller configuration is done by device tree entries in `kernel/dts/bcm_b53_template.dtsi`. The current device tree entry sets up the controller to operate in High-Speed DDR mode (52 MHz Bus, 104 Mbps).
- eMMC userspace utilities: A set of userspace utilities for configuring and querying eMMC devices at runtime have been added in `userspace/gpl/apps/mmc-utils`.

6.3 eMMC Linux Partitions

Linux automatically detects and creates block devices for physical and logical eMMC partitions. Figure 6 shows how eMMC partitions are mapped to block devices under Linux.

Figure 6: Mapping eMMC Partitions to Linux Block Devices



For ease of use, softlinks are made using the GPT partition labels to the actual eMMC Linux devices, as shown in the following example.

```
#ls –al /dev | grep ‘>’
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 bootfs1 -> /dev/mmcblk0p2
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 bootfs2 -> /dev/mmcblk0p6
lrwxrwxrwx    1 admin    root            15 Jan  1 00:00 data -> /dev/mmcblk0p10
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 mdata1_1 -> /dev/mmcblk0p4
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 mdata1_2 -> /dev/mmcblk0p5
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 mdata2_1 -> /dev/mmcblk0p8
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 mdata2_2 -> /dev/mmcblk0p9
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 nvram -> /dev/mmcblk0p1
lrwxrwxrwx    1 admin    root             9 Jan  1 00:00 root -> mmcblk0p3
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 rootfs1 -> /dev/mmcblk0p3
lrwxrwxrwx    1 admin    root            14 Jan  1 00:00 rootfs2 -> /dev/mmcblk0p7
```

7 Getting Started With eMMC

This section provides details regarding the initial configuration of blank eMMC devices and describes various methods that are available to flash initial eMMC images.

7.1 Physical Partitioning Operations

eMMC devices have an option of changing the physical layout of the underlying storage space. These physical partitioning operations are irreversible and require that the device go through a full power-cycle. NONE of these Physical Partitioning Operations are required for CommEngine eMMC images to work.

If your implementation does not require these Physical Partitioning Operations, then skip to Loading Images Via JTAG and Ethernet.

NOTE: All physical partitioning operations should ideally be done by the flash programmer itself, BEFORE any data is written to the device. It is possible to do these operations via Linux, however, this will result in corruption of data on the eMMC device.

7.1.1 Configuring Pseudo SLC (pSLC) mode

Before any data is written to the eMMC device, users have an option of configuring the User Data Partition in pSLC or ‘enhanced user data’ mode. pSLC increases the lifespan of the User Data Partition by storing one-bit per underlying NAND cell and as a result increasing the number of Program/Erase (P/E) cycles of the User Data Area by a factor of 10. However, this increase in lifespan comes at the expense of storage density with the overall storage capacity of the User Data Area getting reduced.

pSLC mode can be configured vis a Flash programmer (recommended), or via Linux, as described below.

7.1.1.1 Configuring pSLC Mode Via Flash Programmer

The recommended way to configure pSLC mode is via the Flash programmer, before any data is written to the device. pSLC mode is configured by writing to the eMMC device’s EXT_CSD registers and most programmers have built-in functions that allow users to configure pSLC mode by just specifying the pSLC area size.

If the Flash programmer requires manual configuration of EXT_CSD registers in order to enable pSLC mode, then follow this configuration sequence:

1. Enable size definition for pSLC area:
   EXT_CSD[ERASE_GROUP_DEF] = 0x1
2. Set 32-bit pSLC area start address by writing to four EXT_CSD registers:
   EXT_CSD[ENH_START_ADDR_3] = See [1] for valid values
   EXT_CSD[ENH_START_ADDR_2] = See [1] for valid values
   EXT_CSD[ENH_START_ADDR_1] = See [1] for valid values
   EXT_CSD[ENH_START_ADDR_0] = See [1] for valid values
3. Set pSLC size by writing to three EXT_CSD registers:
   EXT_CSD[ENH_SIZE_MULT_2] = See [1] for valid values
   EXT_CSD[ENH_SIZE_MULT_1] = See [1] for valid values
   EXT_CSD[ENH_SIZE_MULT_0] = See [1] for valid values
4. Set enhanced attribute on USERDATA partition:
   EXT_CSD[PARTITIONS_ATTRIBUTE] = 0x1

5. Set partitioning complete flag (Note that after this flag is set, you cannot perform any other physical partitioning operation. Therefore make sure all other physical partition configuration (e.g. Setting GP partitions) has been done before setting this flag)
   EXT_CSD[PARTITION_SETTING_COMPLETED] = 0x1
6. Write the EXT_CSD configuration to the eMMC device
7. Power cycle the board

For details about EXT_CSD register addresses and valid register values for configuring pSLC mode, please refer to [1].

7.1.1.2 Configuring pSLC Mode Via Linux

If pSLC mode is configured from a Linux image running on the SoC, ALL DATA ON THE DEVICE may be lost and the device may be unbootable. After configuring pSLC mode via Linux, the eMMC device must be erased and reprogrammed to guarantee data integrity. To configure pSLC mode after a Linux image has been flashed on to the eMMC, execute the following commands on the busy box shell:

```
# mmc enh_area set <-y|-n> <offset in KiB> <size in KiB> <device>
```

To set first 100MB of user data partition as pSLC:

```
# mmc enh_area set -y 0 10240 /dev/mmcblk0

Done setting ENH_USR area on /dev/mmcblk0
setting OTP PARTITION_SETTING_COMPLETED!
Setting OTP PARTITION_SETTING_COMPLETED on /dev/mmcblk0 SUCCESS
Device power cycle is needed for settings to take effect.
Confirm that PARTITION_SETTING_COMPLETED bit is set using 'extcsd read' after power cycle.
```

7.1.2 Configuring General Purpose Partitions (GP)

The User Data Partition can be further divided into four extra General Purpose Partitions (The User Data Partition still exists but is smaller). GP partitions should ideally be configured BEFORE any data is written to the device via the Flash programmer itself. If GP partitions are configured after data has been written to the device, ALL DATA ON THE DEVICE may be lost. Note that configuring GP partitions is an irreversible operation and requires a full power-cycle to take effect.

7.1.2.1 Configuring GP Partitions Via Flash Programmer

The recommended way to configure GP partitions is ideally via the Flash programmer, before any data is written to the device. GP partitions are configured by writing to the eMMC device's EXT_CSD registers. Most programmers have built-in functions that allow users to configure GP partitions by just specifying the number of GP partitions required along with their respective sizes.

If the Flash programmer requires manual configuration of EXT_CSD registers in order to enable GP partitions, then follow this configuration sequence:

1. Enable size definition for GP partitions:
   EXT_CSD[ERASE_GROUP_DEF] = 0x1
2. Set GP size by writing to four EXT_CSD registers for each GP:
   EXT_CSD[GP_SIZE_MULT_GP#_2] = See [1] for valid values
   EXT_CSD[GP_SIZE_MULT_GP#_1] = See [1] for valid values
   EXT_CSD[GP_SIZE_MULT_GP#_0] = See [1] for valid values

3. Set enhanced attribute on GP partition if required:
   EXT_CSD[PARTITIONS_ATTRIBUTE] = See [1] for valid values
4. Set extended attributes on GP partition if required:
   EXT_CSD[EXT_PARTITIONS_ATTRIBUTE] = See [1] for valid values
5. Set partitioning complete flag (Note that after this flag is set, you cannot perform any other physical partitioning operation. Therefore make sure all other physical partition configuration (e.g. Setting pSLC mode) has been done before setting this flag):
   EXT_CSD[PARTITION_SETTING_COMPLETED] = 0x1
6. Write the EXT_CSD configuration to the eMMC device
7. Power cycle the board

For details about EXT_CSD register addresses and valid register values for configuring pSLC mode, please refer to [1].

7.1.2.2 Configuring GP Partitions Via Linux

If GP partitions are configured from a Linux image running on the SoC, ALL DATA ON THE DEVICE may be lost and the device may be unbootable. After configuring GP partitions via Linux, the eMMC device must be erased and reprogrammed to guarantee data integrity. To configure GP partitions after a Linux image has been flashed on to the eMMC, execute the following commands on the busy box shell:

```
# mmc gp create <-y|-n|-c> <length KiB> <partition> <enh_attr> <ext_attr> <device>
```

e.g To create four GP partitions of 100MB each:

```
# mmc gp create -c 10240 1 0 0 /dev/mmcblk0;
# mmc gp create -c 10240 2 0 0 /dev/mmcblk0;
# mmc gp create -c 10240 3 0 0 /dev/mmcblk0;
# mmc gp create -y 10240 4 0 0 /dev/mmcblk0
Done setting ENH_USR area on /dev/mmcblk0
setting OTP PARTITION_SETTING_COMPLETED!
Setting OTP PARTITION_SETTING_COMPLETED on /dev/mmcblk0 SUCCESS
Device power cycle needed for settings to take effect.
Confirm that PARTITION_SETTING_COMPLETED bit is set using 'extcsd read' after power cycle
```

GP partitions appear in linux as /dev/mmcblk0gp0-3.

7.2 Loading Images Via JTAG and Ethernet

JTAG is used to load the CFERAM bootloader. Once the bootloader is running, the full image can be downloaded over the Ethernet. This section shows the steps to load initial images using JTAG and Ethernet.

1. Build an eMMC enabled CFERAM ELF:
   ```
   >cd cfe/build/broadcom/bcm63xx_rom/
   >make BRCM_CHIP=<chipname> BLD_EMMC=1
   ```
2. Strap target to boot from eMMC.
3. Attach to target from JTAG and initialize DDR.
4. Load CFERAM elf file from `cfe/build/broadcom/bcm63xx_ram/cfe<chipname>`.
5. Run the loaded image.
   If the image boots up successfully you should see the eMMC related logs (an example is given below).
   ```
   Initializing eMMC (v0.94). 2013.11.12.
   GPT validation success. !!!
   ```

```
[Booted from eMMC BOOT1 Partition]
EMMC Addr Mode: Sector, ReadBlkLen 512 bytes, WriteBlklen 512 bytes
EMMC device: Samsung 4YMD3R v0.1, Serial 0x93f141f3, Size 3736MB
[eMMC Partition Information] :
  Partition  :  Physical,   Partitioned
  - Data     :  003728MB,   003722MB
  - Boot1    :  000004MB,   004096KB
  - Boot2    :  000004MB,   004096KB
```

6. Check that the default eMMC CFE partitions have been automatically created:

```
CFE> showdevs
Device Name          Description
-------------------  ---------------------------------------------------------
uart0                BCM63xx DUART channel 0
emmcflash0.gpt0      EMMC phys partition DATA, offset:0000000000000000 size:00000032KB
emmcflash0.nvram     EMMC phys partition DATA, offset:0000000000100000 size:00000002KB
emmcflash0.bootfs1   EMMC phys partition DATA, offset:0000000000200000 size:00102400KB
emmcflash0.rootfs1   EMMC phys partition DATA, offset:0000000006600000 size:00102400KB
emmcflash0.mdata1_1  EMMC phys partition DATA, offset:000000000CA00000 size:00000032KB
emmcflash0.mdata1_2  EMMC phys partition DATA, offset:000000000CB00000 size:00000032KB
emmcflash0.bootfs2   EMMC phys partition DATA, offset:000000000CC00000 size:00102400KB
emmcflash0.rootfs2   EMMC phys partition DATA, offset:0000000013000000 size:00102400KB
emmcflash0.mdata2_1  EMMC phys partition DATA, offset:0000000019400000 size:00000032KB
emmcflash0.mdata2_2  EMMC phys partition DATA, offset:0000000019500000 size:00000032KB
emmcflash0.data      EMMC phys partition DATA, offset:0000000019600000 size:00102400KB
emmcflash0.unalloc   EMMC phys partition DATA, offset:000000001FA00000 size:03299296KB
emmcflash0.gpt1      EMMC phys partition DATA, offset:00000000E8FF8000 size:00000032KB
emmcflash1.cfe       EMMC phys partition BOOT1, offset:0000000000000000 size:00004096KB
emmcflash2.cfe       EMMC phys partition BOOT2, offset:0000000000000000 size:00004096KB
```

7. Dump current GPT partition settings. This shows the complete eMMC GPT formatting command which results in the current partitioning scheme.

```
CFE> emmcdmpgpt
emmc format command for current partition configuration:
emmcfmtgpt bootfsKB rootfsKB dataKB misc1KB misc2KB misc3KB misc4KB
emmcfmtgpt 102400 102400 102400 0 0 0 0
```

8. If the current eMMC partitions are not compatible with the image being loaded (the partitions are too small), repartition eMMC. The following example sets bootfs, rootfs, and data partitions to 100 Mb and all misc. partitions to 0 Mb:

```
CFE> emmcfmtgpt 102400 102400 102400 0 0 0 0
```

9. Log on to WebUI and download the `bcm<chipname>_cfe_fs_kernel_emmc_squash/ext4` image.
10. Reboot the board.

7.3 Loading Images Via eMMC Programmer

If an eMMC flash programmer is available, then flash images can be written directly to the device.

1. Ensure that the flash programmer can identify the BOOT and USERDATA physical partitions of the eMMC device.
2. Flash the `bcm<chip>_emmc_`**`bootpartition`**`_flash_image_<board_id>.w` image to the physical **boot partition**.
3. Flash the `bcm<chip>_emmc_`**`datapartition`**`_flash_image_<board_id>.w` image to the to the physical **user data partition**.
4. Modify the device's EXTCSD register as follows:

```
EXT_CSD[BOOT_BUS_CONDITIONS].BOOT_BUS_WIDTH = 0x2 (8-bit wide bus)
EXT_CSD[PARTITION_CONFIG].BOOT_PARTITION_ENABLE = 0x1
EXT_CSD[PARTITION_CONFIG].BOOT_ACK = 0x1
```

7.4 Loading Images Via NAND

If the board has both NAND and eMMC installed and the SoC allows concurrent usage of both NAND and eMMC, it is possible to boot into NAND and program eMMC from the Linux filesystem.

1. Set the straps for NAND and flash a NAND image. Note that the build profile used to generate the NAND image must have eMMC support enabled.
2. Boot into Linux. Check that the eMMC was detected.
```
#ls –al /dev | grep mmc
brw-rw----    1 admin    root      179,   0 Jan  1 00:00 mmcblk0
brw-rw----    1 admin    root      179,  32 Jan  1 00:00 mmcblk0boot0
brw-rw----    1 admin    root      179,  64 Jan  1 00:00 mmcblk0boot1
```
3. Point your TFTP server to the `targets/<BUILD_PROFILE>` directory.
4. Download and flash the eMMC physical boot partition image.
```
#cd /var
#tftp -g -r bcm<chip>_emmc_bootpartition_flash_image_<board_id>.w <tftp server IP>
#echo 0 > /sys/block/mmcblk0boot0/force_ro
#dd if= bcm<chip>_emmc_bootpartition_flash_image_<board_id>.w of=/dev/mmcblk0boot0
```
5. Download and flash the eMMC physical user data partition image.
```
#tftp -g -r bcm<chip>_emmc_datapartition_flash_image_<board_id>.w <tftp server IP>
#dd if= bcm<chip>_emmc_datapartition_flash_image_<board_id>.w of=/dev/mmcblk0
```
6. Update the eMMC device’s EXTCSD register.
```
#/sbin/mmc bootpart enable 1 1 /dev/mmcblk0
#/sbin/mmc bootwidth set 8 /dev/mmcblk0
```
7. Power off the board, change straps to boot from eMMC, power on the board.

7.5 Enabling eMMC Hardware Reset Pin

The hardware reset pin on eMMC devices (RST_N) is disabled by default. Hardware reset functionality on eMMC devices is controlled by the EXT_CSD register settings. If the hardware reset functionality is not enabled, then asserting the RST_N pin will have no affect.

7.5.1 Enabling eMMC Hardware Reset Via Flash Programmer

The recommended way to configure eMMC hardware reset functionality is via the Flash programmer, before any data is written to the device.

Configure EXT_CSD as follows:

- Set the hardware reset function to permanently enabled:
```
EXT_CSD[RST_n_FUNCTION] = 0x1
```

7.5.2 Enabling eMMC Hardware Reset Via Linux

The hardware reset function on eMMC devices can also be enabled via the busybox shell using the following commands:
```
# mmc hwreset enable /dev/mmcblk0
<POWER CYCLE THE BOARD>
```

8 Limitations

CommEngine eMMC support has the following limitations:

- Only eMMC devices with a 512-byte sector size are fully supported in CFE and Linux.
- Devices with 4 Kb native sector sizes will work as long as they are running in 512-byte emulation mode.

Revision History

CPE-AN3001; May 22, 2019

- Updated Image Types
- Updated Configuring pSLC Mode Via Flash Programmer
- Updated Configuring GP Partitions Via Flash Programmer
- Added Physical Partitioning Operations
- Added Enabling eMMC Hardware Reset Pin

CPE-AN3000; August 2017

Initial Release