BROADCOM®

CommEngine eMMC Support

Broadcom Limited
Web: www.broadcom.com
Corporate Headquarters: San Jose, CA

CommEngine eMMC Support

Version 2.0 (May 11, 2017)

Broadcom Limited
Web: www.broadcom.com
Corporate Headquarters: San Jose, CA
© 2016 by Broadcom All rights reserved.

Revision History

Revision	Date	Change Description
1.0	02/01/16	Initial release
1.1	05/11/2017	Blank eMMC instructions

Table of Contents

List of Tables

List of Figures

1 Scope

This document provides details regarding the current and future support for 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 **[1]**.

1.1 References

[1] JEDEC Standard: Embedded Multi-Media Card (e•MMC) Electrical Standard (5.1)

[2] 63138B0 AHB Subsystem

[3] Simplified SD Host Controller Spec

2 Introduction to eMMC

eMMC (Embedded MultiMediaCard) devices are managed flash memories which consist of a 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.

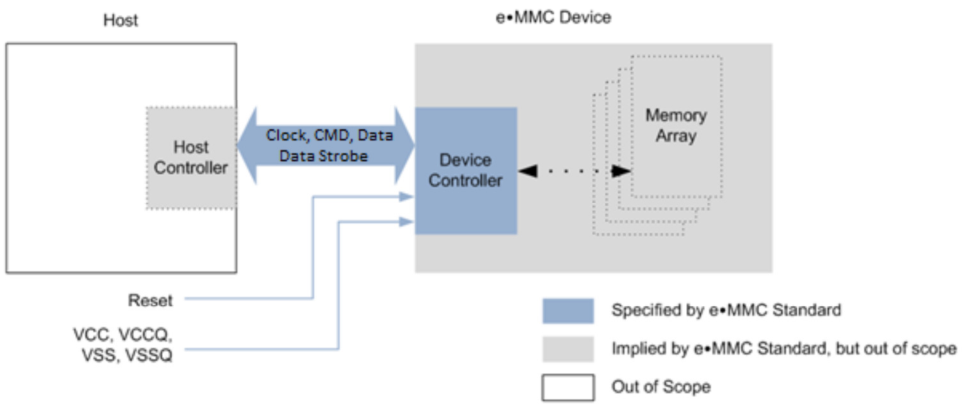

Figure 1: eMMC System Overview

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

2.1 eMMC Device Interface

- **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 via 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 (i.e device can be forced to operate in native mode) via eMMC internal register settings.

- **Addressing**: For eMMC devices <2Gb, 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 4Kb 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 4kb size multiples.

- **Data Transfer Speeds:** eMMC devices support a variety of bus speed modes, with each mode being backwards 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

2.2 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 clk 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 User Data partitions). The main use case for using these partitions is the ability to enable certain enhanced features (such as better reliability) on them.

Before issuing and read or write commands, the host has to explicitly select which partition is to be accessed by updating internal eMMC device registers.

2.3 eMMC Host Interface

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

2.3.1 eMMC Host Controller Interface

The eMMC host controller register interface conforms to the standardized SD Host Controller Specification **[2]**. 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

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

2.3.2 eMMC Boot Controller

Broadcom's custom eMMC boot controller gets automatically activated when straps are configured for eMMC boot. The boot controller intercepts all requests to the eMMC memory XIP window, and performs sequential data fetches from the eMMC device into internal SRAMS. Note that since the data fetches are sequential, any forward or backward jumps in eMMC memory addresses will result in long delays.

The boot controller can also be made to manually fetch eMMC data by writing explicit eMMC addresses into its registers and then polling for the data to appear in internal boot controller SRAMS. For more information regarding the operation of the eMMC boot controller please refer to **[3]**.

3 eMMC Images and Flash Layout

3.1 Flash Layout

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

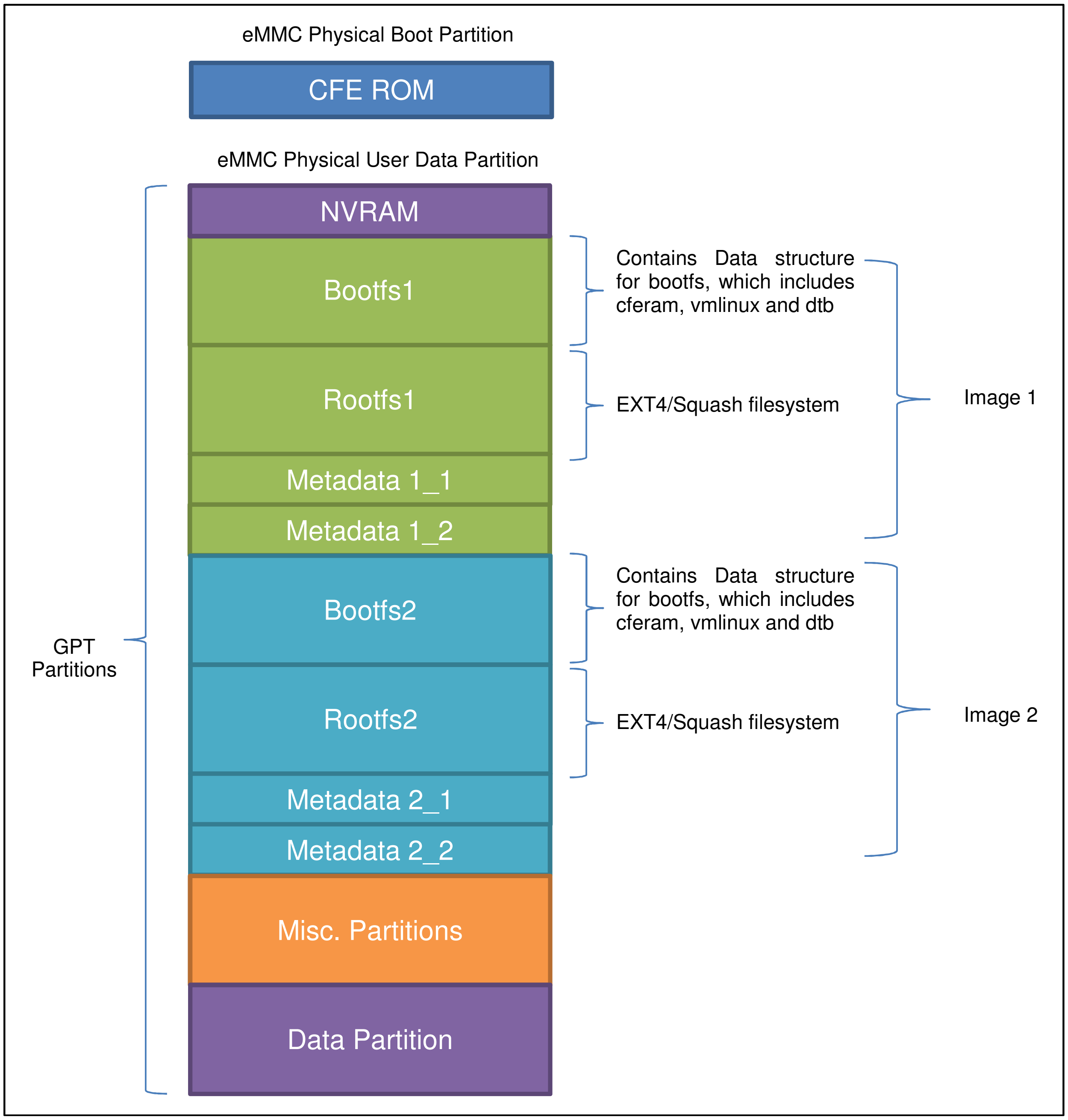
eMMC Physical Boot Partition
CFE ROM
eMMC Physical User Data Partition
NVRAM
Bootfs1
Rootfs1
Metadata 1_1
Metadata 1_2
Bootfs2
Rootfs2
Metadata 2_1
Metadata 2_2
Misc. Partitions
Data Partition
GPT Partitions
Contains Data structure for bootfs, which includes cferam, vmlinux and dtb
EXT4/Squash filesystem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

Figure 2: eMMC flash layout

3.2 eMMC Images

3.2.1 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**: These images are to be used for firmware upgrades via CFE or Linux . These images have names ending in either **cfe_fs_kernel_emmc_<rootfs>** or **fs_kernel_emmc_<rootfs>**
- **Whole flash Images**: These 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 DATA physical partitions. These images have names ending in a **.w** extension

3.2.1.1 Tagged Image Format

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.

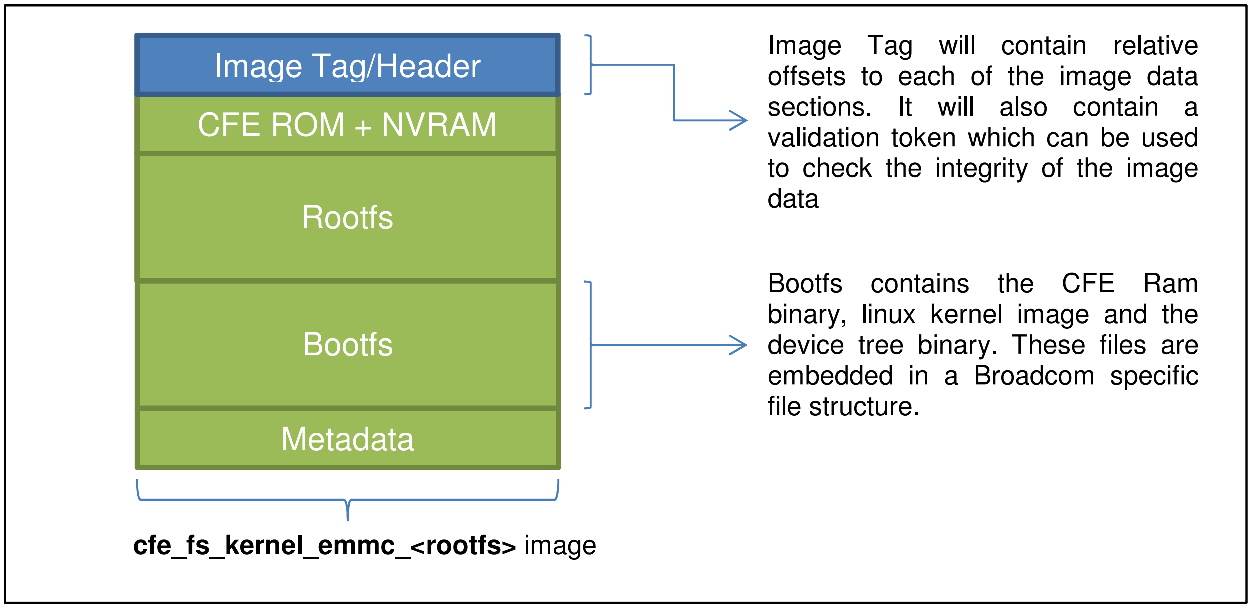
Image Tag/Header
CFE ROM + NVRAM
Rootfs
Bootfs
Metadata
cfe_fs_kernel_emmc_<rootfs> image
Image Tag will contain relative offsets to each of the image data sections. It will also contain a validation token which can be used to check the integrity of the image data
Bootfs contains the CFE Ram binary,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

Figure 3: eMMC Tagged Image Format

3.2.1.2 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 BOOT and DATA partitions. The whole image for the eMMC 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 BOOT partition contains the CFEROM binary. Figure 4 shows the eMMC whole image formats.

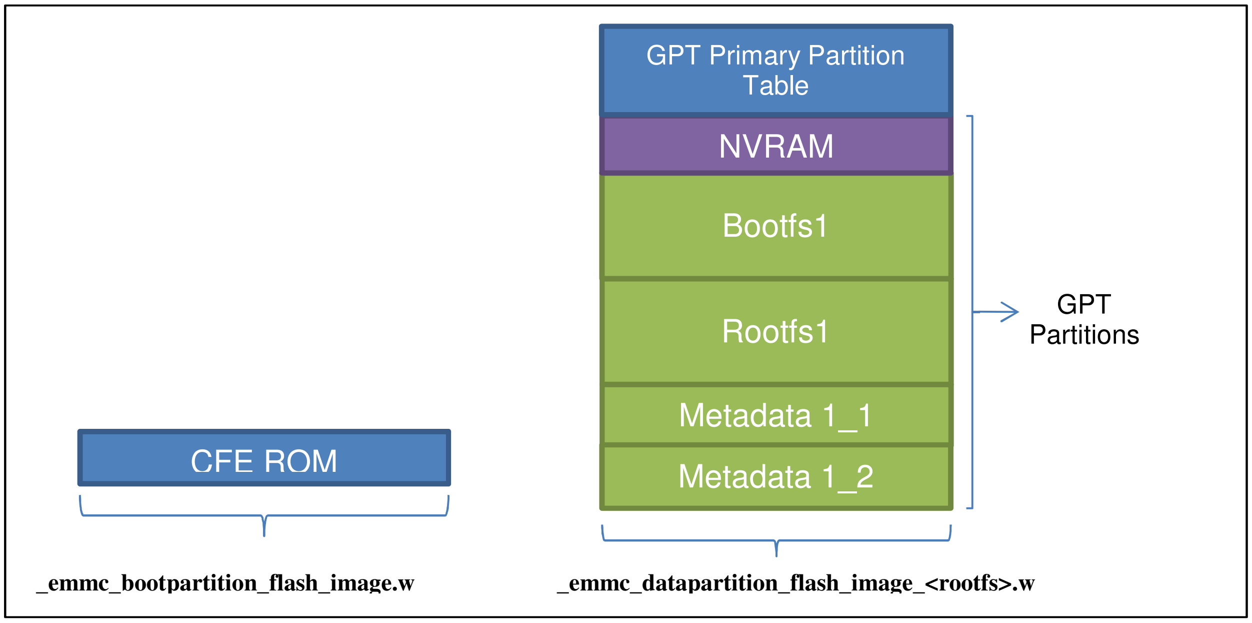
GPT Primary Partition Table
NVRAM
Bootfs1
Rootfs1
Metadata 1_1
Metadata 1_2
GPT Partitions
CFE ROM
_emmc_bootpartition_flash_image.w
_emmc_datapartition_flash_image_<rootfs>.w

Figure 4: eMMC whole image formats for BOOT and DATA partitions

3.2.2 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, much larger image size

4 eMMC Support in CFE RAM

The CFE source code has a CFE device framework, where `cfe_device` structures can be associated to 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/end of the eMMC 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 eMMC logical partitions will map to CFE devices.

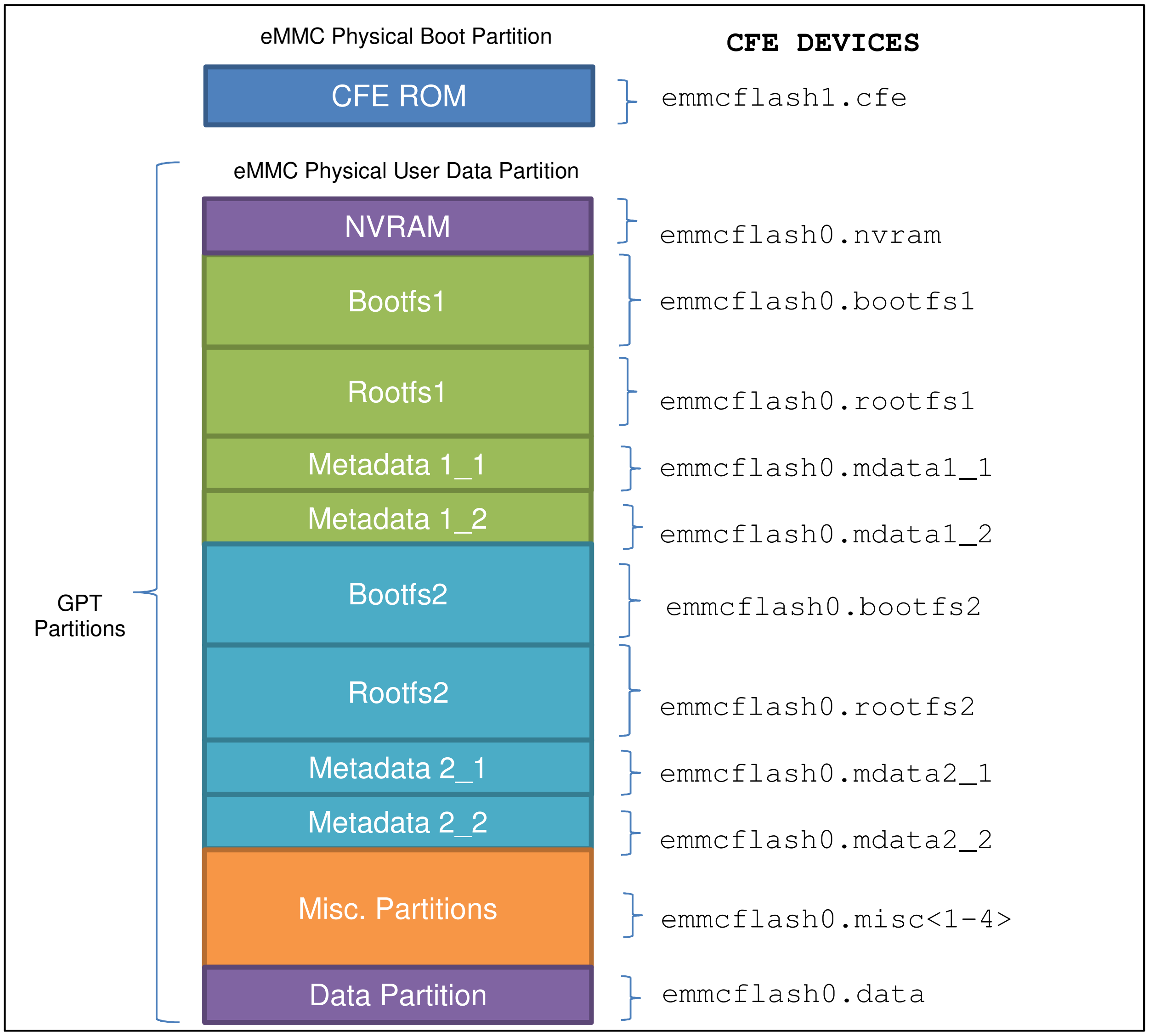

Figure 5: Mapping eMMC partitions to CFE devices

4.1 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 special CFE devices are automatically created and are mapped to the first and last few blocks on the eMMC's user data physical partition, thus ensuring that the partition tables are always written to the proper locations on the eMMC.

4.2 eMMC CFE Commands

Along with general commands for flashing new images and setting CFE parameters, new eMMC commands shown in Table 2 have been added to the CFE menu.

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)

5 eMMC Support in Linux

5.1 Build

eMMC enabled linux images can be built by using build profiles generated via the `maketargets` utility along with the `EMMC.arch` file.

5.2 Source Code and Utilities

eMMC devices are natively supported in linux via 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 via 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`.

5.3 eMMC Linux Partitions

Linux will automatically detect and create block devices for physical and logical eMMC partitions. Figure 6 shows how eMMC partitions are mapped to block devices under Linux.

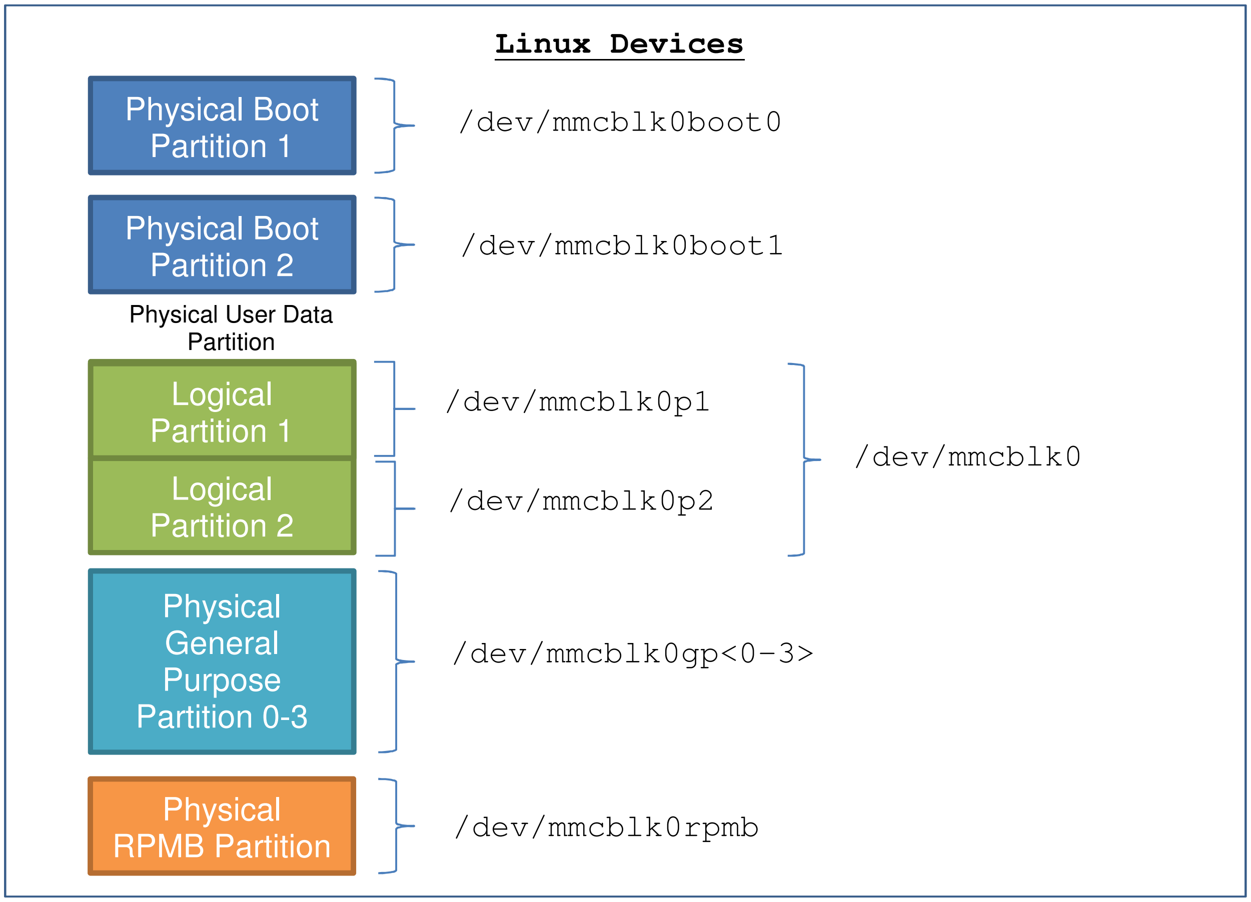

Figure 6: Mapping eMMC partitions to Linux block devices

For ease of use, softlinks are made using the GPT partition labels to the actual eMMC linux devices e.g:

```
#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
```

6 Getting started with eMMC

This section describes various methods that are available to flash initial images on boards with blank eMMC devices.

6.1 Loading Images Via JTAG and Ethernet

This section describes the steps involved in loading initial images via JTAG and Ethernet. JTAG is used to load the CFERAM bootloader, and once the bootloader is running, the full image can be downloaded via Ethernet. The instructions are as follows:

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 via eMMC

3. Attach to target via 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 booted up successfully you should see the following eMMC related logs (sample):

```
    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 will show the complete eMMC GPT formatting command which will result 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 current eMMC partitions are not compatible with the image being loaded (too small), re-partition eMMC. Following example sets bootfs, rootfs and data partitions to 100Mb and all misc partitions to 0Mb :

```
      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

6.2 Loading Images Via eMMC Programmer

If a eMMC flash programmer is available then whole flash images can be written directly to the device. The instructions are as follows:

1. Ensure the the 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 BOOT partition
3. Flash the `bcm<chip>_emmc_datapartition_flash_image_<board_id>.w` image to the to the USERDATA 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

6.3 Loading Images Via NAND

If the board has both NAND and eMMC installed **AND** the SoC allows concurrent usage of both NAND and eMMC, we can boot into NAND and program eMMC via the linux filesystem. The instructions are as follows:

1. Strap 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 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 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 Limitations

CommEngine eMMC support has the following limitations:

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