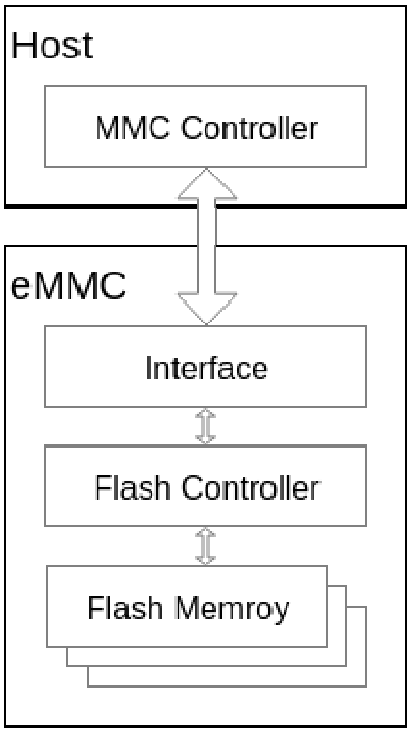
一、eMMC 簡介

eMMC 是 **embedded MultiMediaCard** 的簡稱。MultiMediaCard，即 **MMC， 是一種閃存卡（Flash Memory Card）標準**，它定義了 MMC 的架構以及訪問 Flash Memory 的接口和協議。而 **eMMC 則是對 MMC 的一個拓展**，以滿足更高標準的性能、成本、體積、穩定、易用等的需求。

eMMC 的整體架構如下圖片所示：

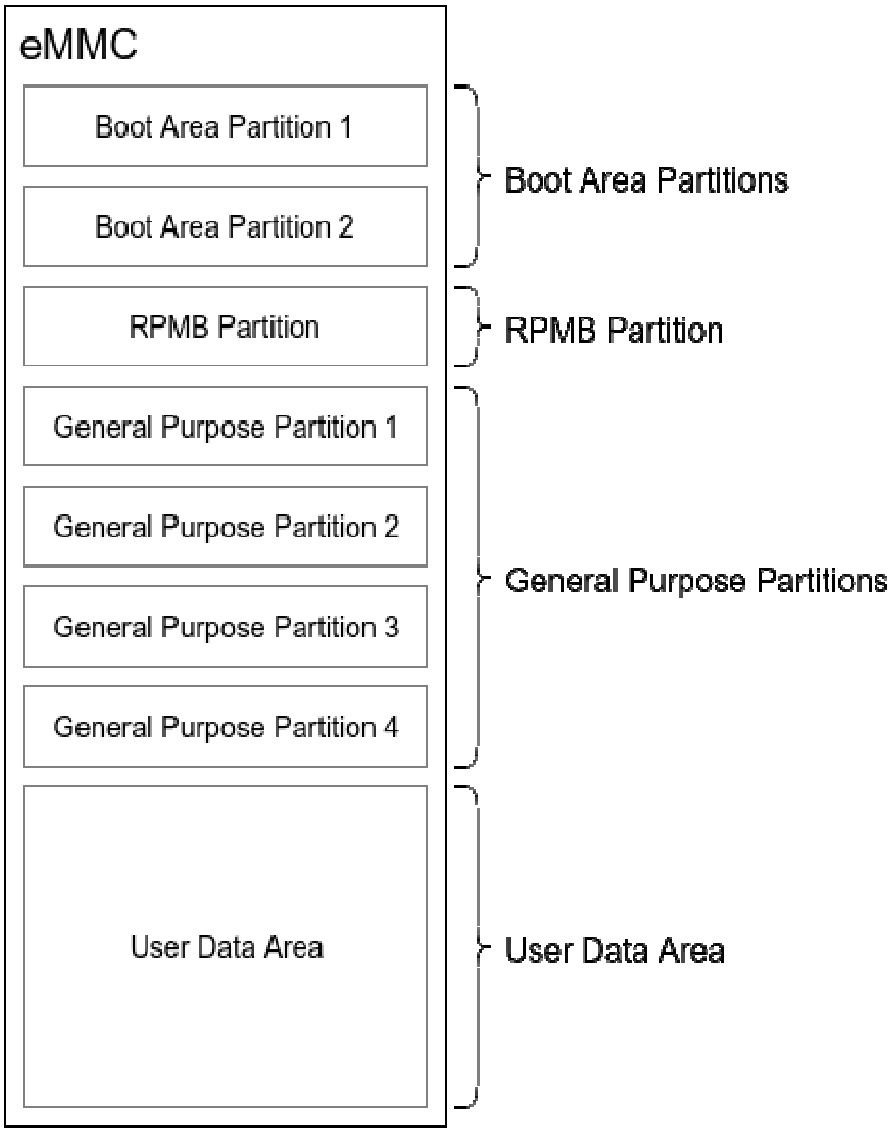


圖片： eMMC 整體架構

eMMC 內部主要可以分為 **Flash Memory**、**Flash Controller** 以及 **Host Interface** 三大部分。

1、Flash Memory

Flash Memory 是一種非易失性的存儲器，通常在嵌入式系統中用於存放系統、應用和數據等，類似於 PC 系統中的硬盤。
目前，**絕大部分**手機和平板等移動設備中所使用的 **eMMC 內部的 Flash Memory 都屬於 NAND Flash**，關於 NAND Flash 的更多細節可以參考 Flash Memory 章節。
eMMC 在內部對 Flash Memory 劃分了幾個主要區域，如下圖所示：



圖片：*eMMC* 內部分區

1.1 BOOT Area Partition 1 & 2

此分區主要是**為了支持從 *eMMC* 啟動系統**而設計的。
該分區的數據，在 *eMMC* 上電後，可以通過很簡單的協議就可以讀取出來。同時，**大部分的 *SOC* 都可以通過 *GPIO* 或者 *FUSE* 的配置，讓 *ROM* 代碼在上電後，將 *eMMC BOOT* 分區的內容加載到 *SOC* 內部的 *SRAM* 中執行**。

1.2 RPMB Partition

RPMB 是 **Replay Protected Memory Block** 的簡稱，它通過 HMAC SHA-256 和 Write Counter 來保證保存在 RPMB 內部的數據不被非法篡改。

在實際應用中，**RPMB 分區通常用來保存安全相關的數據**，例如指紋數據、安全支付相關的密鑰等。

1.3 General Purpose Partition 1～4

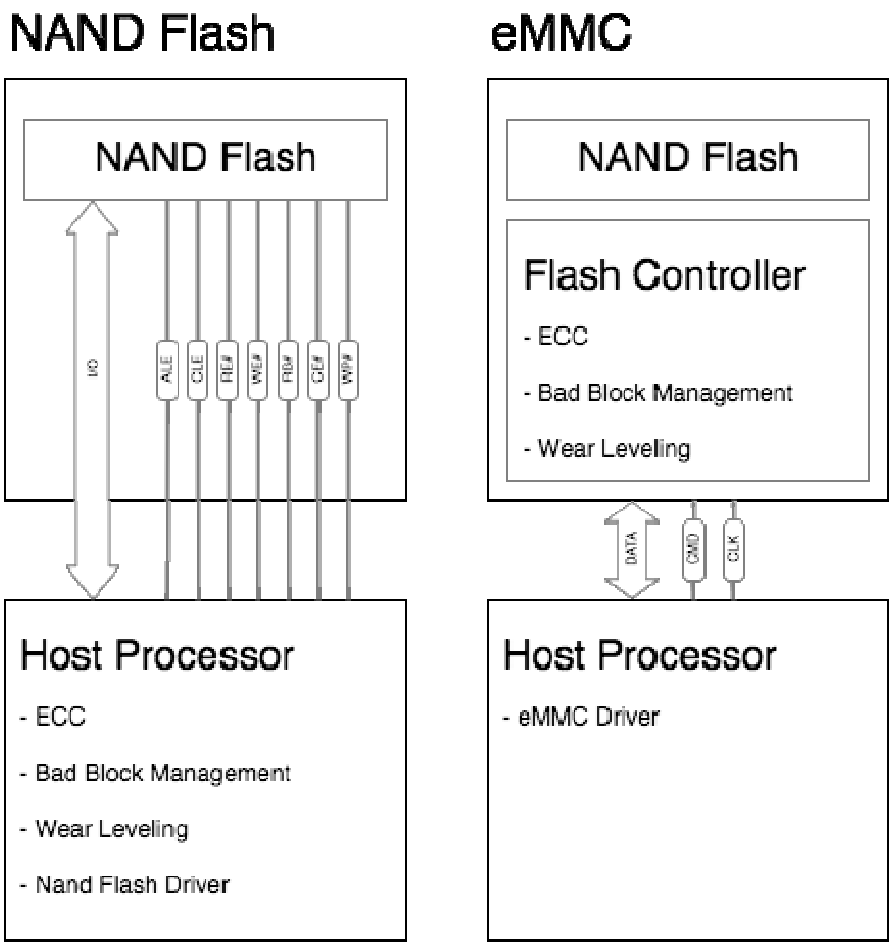
此區域則主要**用於存儲系統或者用戶數據**。 General Purpose Partition 在芯片出廠時，通常是不存在的，需要主動進行配置後，才會存在。

1.4 User Data Area

此區域則主要**用於存儲系統和用戶數據**。
User Data Area 通常會進行再分區，例如 Android 系統中，通常在此區域分出 boot、system、userdata 等分區。
更多 eMMC 分區相關的細節，請參考 **eMMC 分區管理** 章節。

2、Flash Controller

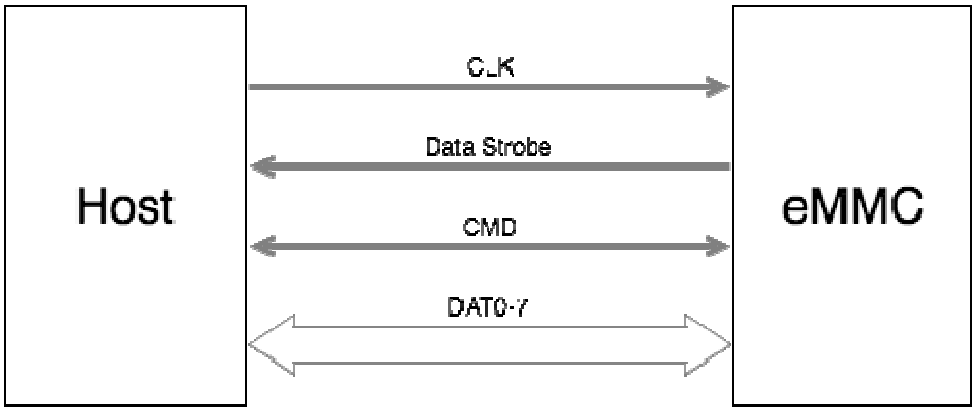
NAND Flash 直接接入 Host 時，Host 端通常需要有 NAND Flash Translation Layer，即 NFTL 或者 NAND Flash 文件系統來做壞塊管理、ECC 等的功能。
eMMC 則在其內部集成了 Flash Controller，用於完成擦寫均衡、壞塊管理、ECC 校驗等功能。相比於直接將 NAND Flash 接入到 Host 端，eMMC 屏蔽了 NAND Flash 的物理特性，可以減少 Host 端軟件的複雜度，讓 Host 端專注於上層業務，省去對 NAND Flash 進行特殊的處理。同時，eMMC 通過使用 Cache、Memory Array 等技術，在讀寫性能上也比 NAND Flash 要好很多。



圖片：NAND Flash 與 eMMC

3、Host Interface

eMMC 與 Host 之間的連接如下圖所示：



圖片：eMMC Interface

各個信號的用途如下所示：

CLK

用於同步的時鐘信號

Data Strobe

此信號是從 Device 端輸出的時鐘信號，頻率和 CLK 信號相同，用於同步從 Device 端輸出的數據。**該信號在 eMMC 5.0 中引入**。

CMD

此信號用於發送 Host 的 command 和 Device 的 response。

DAT0-7

用於傳輸數據的 8 bit 總線。

Host 與 eMMC 之間的通信都是 Host 以一個 Command 開始發起的。針對不同的 Command，Device 會做出不同的響應。詳細的通信協議相關內容，請參考 **eMMC 總線協議** 章節。

原文：eMMC 簡介

二、eMMC 分區管理

1、Partitions Overview

eMMC 標準中，將內部的 Flash Memory 劃分為 4 類區域，最多可以支持 8 個硬件分區，分區圖見上節*圖片 eMMC 內部分區*。

1.1 概述

一般情況下，Boot Area Partitions 和 RPMB Partition 的容量大小通常都為 4MB，部分芯片廠家也會提供配置的機會。General Purpose Partitions (GPP) 則在出廠時默認不被支持，即不存在這些分區，需要用戶主動使能，並配置其所要使用的 GPP 的容量大小，GPP 的數量可以為 1 - 4 個，各個 GPP 的容量大小可以不一樣。User Data Area (UDA) 的容量大小則為總容量大小減去其他分區所佔用的容量。更多各個分區的細節將在後續小節中描述。

1.2 分區編址

eMMC 的**每一個硬件分區的存儲空間都是獨立編址的，即訪問地址為 0 - partition size**。具體的數據讀寫操作實際訪問哪一個硬件分區，是由 eMMC 的 Extended CSD register 的 PARTITION_CONFIG Field 中的 Bit[2:0]: PARTITION_ACCESS 決定的，用戶可以通過配置 PARTITION_ACCESS 來切換硬件分區的訪問。也就是說，用戶在訪問特定的分區前，需要先發送命令，配置 PARTITION_ACCESS，然後再發送

相關的數據訪問請求。更多數據讀寫相關的細節，請參考 **eMMC 總線協議** 章節。
eMMC 的各個硬件分區有其自身的功能特性，多分區的設計，為不同的應用場景提供了便利。

2、Boot Area Partitions

Boot Area 包含兩個 Boot Area Partitions，**主要用於存儲 Bootloader，支持 SOC 從 eMMC 啟動系統**。

2.1 容量大小

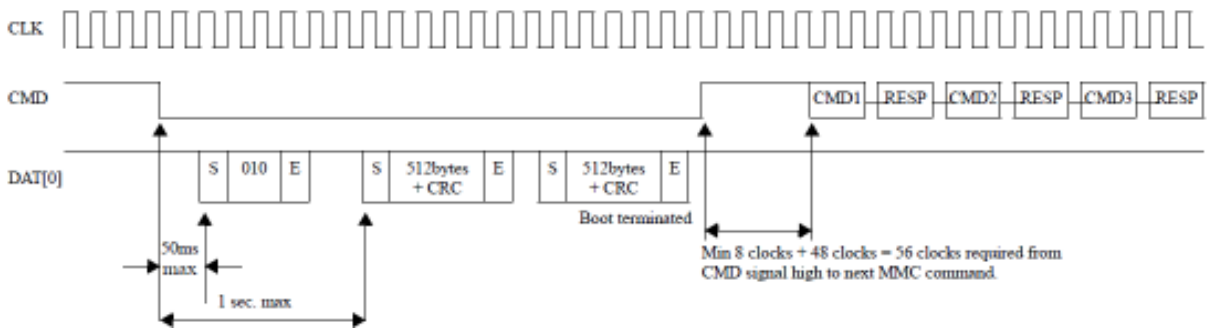
兩個 Boot Area Partitions 的大小是完全一致的，由 Extended CSD register 的 BOOT_SIZE_MULT Field 決定，大小的計算公式如下：
Size = 128Kbytes x BOOT_SIZE_MULT
一般情況下，Boot Area Partition 的大小都為 4 MB，即 BOOT_SIZE_MULT 為 32，部分芯片廠家會提供改寫 BOOT_SIZE_MULT 的功能來改變 Boot Area Partition 的容量大小。BOOT_SIZE_MULT 最大可以為 255，即 Boot Area Partition 的最大容量大小可以為 255 x 128 KB = 32640 KB = 31.875 MB。

2.2 從 Boot Area 啟動

eMMC 中定義了 Boot State，在 Power-up、HW reset 或者 SW reset 後，如果滿足一定的條件，eMMC 就會進入該 State。進入 Boot State 的條件如下：

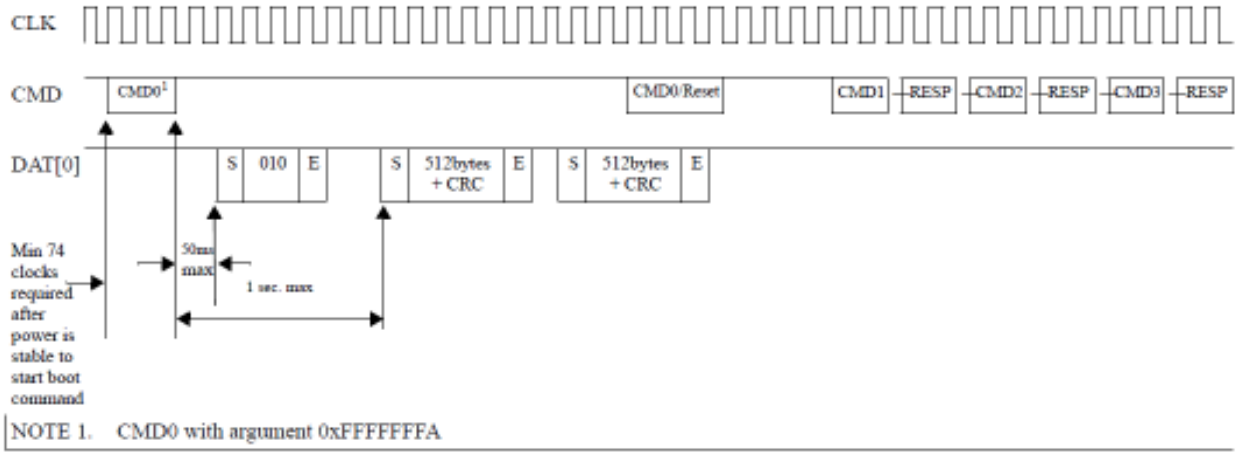
Original Boot Operation

CMD 信號保持低電平不少於 74 個時鐘週期，會觸發 Original Boot Operation，進入 Boot State。



Alternative Boot Operation

在 74 個時鐘週期後，在 CMD 信號首次拉低或者 Host 發送 CMD1 之前，Host 發送參數為 0xFFFFFFFA 的 CMD0 時，會觸發 Alternative Boot Operation，進入 Boot State。



在 Boot State 下，如果有配置 BOOT_ACK，eMMC 會先發送 「010」 的 ACK 包，接著 eMMC 會將最大為 128Kbytes x BOOT_SIZE_MULT 的 Boot Data 發送給 Host。傳輸過程中，Host 可以通過拉高 CMD 信號 (Original Boot 中)，或者發送 Reset 命令 (Alternative Boot 中) 來中斷 eMMC 的數據發送，完成 Boot Data 傳輸。
Boot Data 根據 Extended CSD register 的 PARTITION_CONFIG Field 的 Bit[5:3]:BOOT_PARTITION_ENABLE 的設定，可以從 Boot Area Partition 1、Boot Area Partition 2 或者 User Data Area 讀出。

Boot Data 存儲在 Boot Area 比在 User Data Area 中要更加的安全，可以減少意外修改導致系統無法啟動，同時無法更新系統的情況出現。
（更多 Boot State 的細節，請參考 **eMMC 工作模式** 的 Boot Mode 章節）

2.3 寫保護

通過設定 Extended CSD register 的 BOOT_WP Field，可以**為兩個 Boot Area Partition 獨立配置寫保護功能,以防止數據被意外改寫或者擦出**。
eMMC 中定義了兩種 Boot Area 的寫保護模式：

Power-on write protection

使能後，如果 eMMC 掉電，寫保護功能失效，需要每次 Power on 後進行配置。

Permanent write protection

使能後，即使掉電也不會失效，主動進行關閉才會失效。

3、RPMB Partition

RPMB（Replay Protected Memory Block）Partition 是 eMMC 中的一個**具有安全特性的分區**。
eMMC 在寫入數據到 RPMB 時，會***校驗數據的合法性***，只有指定的 Host 才能夠寫入，同時在讀數據時，也提供了***簽名機制***，保證 Host 讀取到的數據是 RPMB 內部數據，而不是攻擊者偽造的數據。

RPMB 在實際應用中，通常用於存儲一些有防止非法篡改需求的數據，例如手機上指紋支付相關的公鑰、序列號等。RPMB 可以對寫入操作進行鑑權，但是讀取並不需要鑑權，任何人都可以進行讀取的操作，因此存儲到 RPMB 的數據通常會進行加密後再存儲。

3.1 容量大小

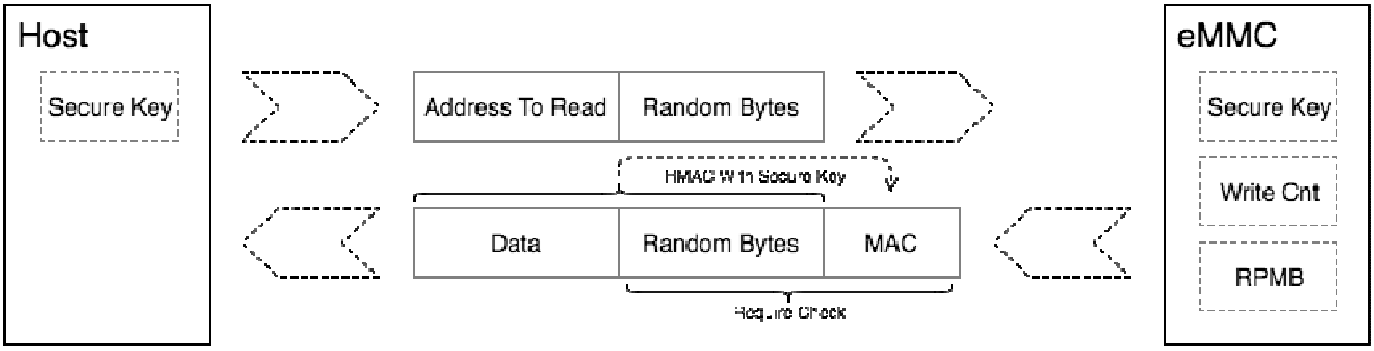
RPMB Partition 的大小是由 Extended CSD register 的 BOOT_SIZE_MULT Field 決定，大小的計算公式如下：
Size = 128Kbytes x BOOT_SIZE_MULT
一般情況下，Boot Area Partition （筆誤？RPMB Partition）的大小為 4 MB，即 RPMB_SIZE_MULT 為 32，部分芯片廠家會提供改寫 RPMB_SIZE_MULT 的功能來改變 RPMB Partition 的容量大小。RPMB_SIZE_MULT 最大可以為 128，即 Boot Area Partition（筆誤？RPMB Partition） 的最大容量大小可以為 128 x 128 KB = 16384 KB = 16 MB。

3.2 Replay Protect 原理

使用 eMMC 的產品，在產線生產時，會為每一個產品生產一個唯一的 256 bits 的 Secure Key，燒寫到 eMMC 的 **OTP 區域（只能燒寫一次的區域）**，同時 Host 在安全區域中（例如：TEE）也會保留該 Secure Key。
在 eMMC 內部，還有一個 RPMB Write Counter。RPMB 每進行一次合法的寫入操作時，Write Counter 就會自動加一 。
通過 Secure Key 和 Write Counter 的應用，RMPB 可以實現數據讀取和寫入的 Replay Protect。

3.3 RPMB 數據讀取

RPMB 數據讀取的流程如下：



a.Host 向 eMMC 發起讀 RPMB 的請求，同時生成一個 16 bytes 的隨機數，發送給 eMMC。

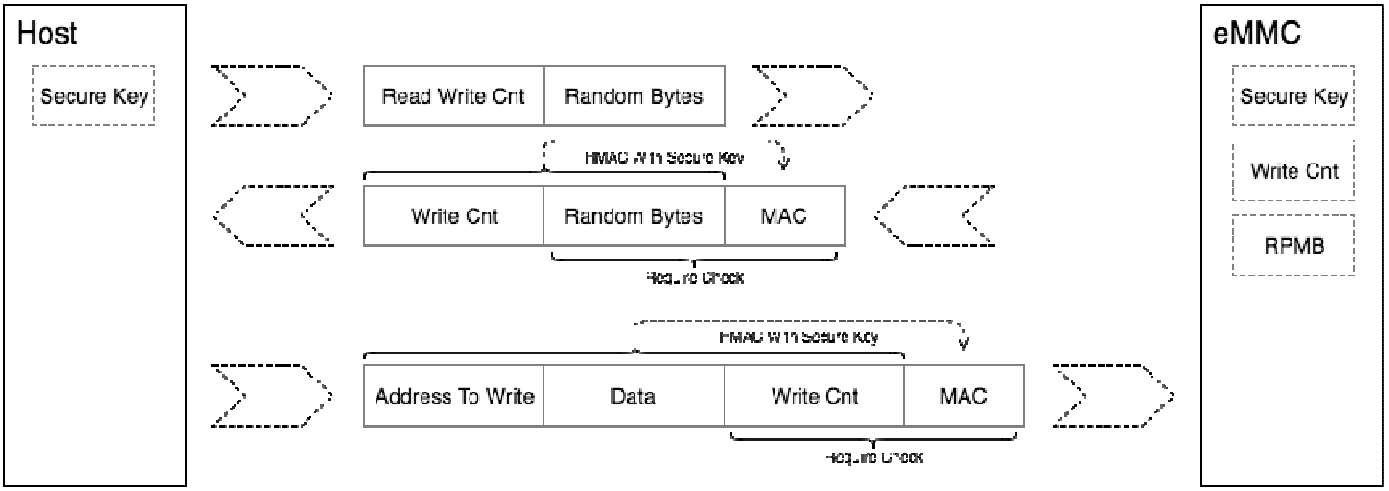
b.eMMC 將請求的數據從 RPMB 中讀出，並使用 Secure Key 通過 HMAC SHA-256 算法，計算讀取到的數據和接收到的隨機數拼接到一起後的簽名。然後，eMMC 將讀取到的數據、接收到的隨機數、計算得到的簽名一併發送給 Host。

c.Host 接收到 RPMB 的數據、隨機數以及簽名後，首先比較隨機數是否與自己發送的一致，如果一致，再用同樣的 Secure Key 通過 HMAC SHA-256 算法對數據和隨機數組合到一起進行簽名，如果簽名與 eMMC 發送的簽名是一致的，那麼就可以確定該數據是從 RPMB 中讀取到的正確數據，而不是攻擊者偽造的數據。

通過上述的讀取流程，可以保證 Host 正確的讀取到 RPMB 的數據。

3.4 RPMB 數據寫入

RPMB 數據寫入的流程如下：



a.Host 按照上面的讀數據流程，讀取 RPMB 的 Write Counter。

b.Host 將需要寫入的數據和 Write Counter 拼接到一起並計算簽名，然後將數據、Write Counter 以及簽名一併發給 eMMC。

c.eMMC 接收到數據後，先對比 Write Counter 是否與當前的值相同，如果相同那麼再對數據和 Write Counter 的組合進行簽名，然後和 Host 發送過來的簽名進行比較，如果簽名相同則鑑權通過，將數據寫入到 RPMB 中。

通過上述的寫入流程，可以保證 RPMB 不會被非法篡改。
更多 RPMB 相關的細節，可以參考 **eMMC RPMB** 章節。

4、General Purpose Partitions

eMMC 提供了 General Purpose Partitions (GPP)，主要用於存儲系統和應用數據。在很多使用 eMMC 的產品中，GPP 都沒有被啟用，因為它在功能上與 **UDA 類似**，產品上直接使用 UDA 就可以滿足需求。

4.1 容量大小

eMMC 最多可以支持 4 個 GPPs，每一個 GPP 的大小可以單獨配置。用戶可以通過設定 Extended CSD register 的以下三個 Field 來設 GPPx (x=1~4) 的容量大小：

- GP_SIZE_MULT_x_2
- GP_SIZE_MULT_x_1
- GP_SIZE_MULT_x_0

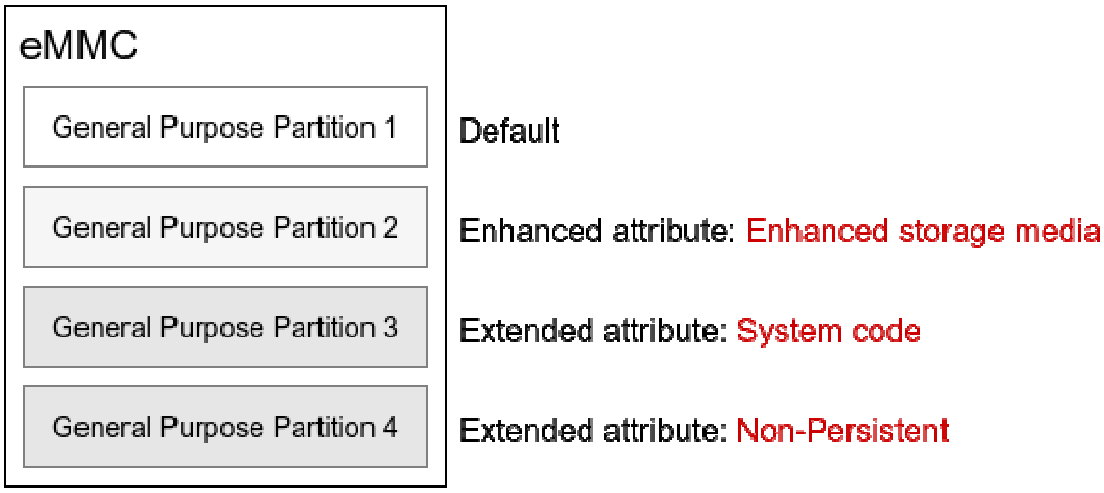
GPPx 的容量計算公式如下：

Size = (GP_SIZE_MULT_x_2 * 2^16 + GP_SIZE_MULT_x_1 * 2^8 + GP_SIZE_MULT_x_0 * 2^0) * (Write protect group size)
Write protect group size = 512KB * HC_ERASE_GRP_SIZE * HC_WP_GRP_SIZE
註：

- eMMC 中，擦除和寫保護都是按塊進行的，上述表達式中的 HC_WP_GRP_SIZE 為寫保護的操作塊大小，HC_ERASE_GRP_SIZE 則為擦除操作的快的大小。
- eMMC 芯片的 GPP 的配置通常是只能進行一次 (OTP)，一般會在產品量產階段，在產線上進行。

分區屬性

eMMC 標準中，為 GPP 定義了兩類屬性，**Enhanced attribute** 和 **Extended attribute**。每個 GPP 可以設定兩類屬性中的一種屬性，不可以同時設定多個屬性。



Enhanced attribute

- Default, 未設定 Enhanced attribute。
- Enhanced storage media， 設定 GPP 為 Enhanced storage media。

在 eMMC 標準中，實際上並未定義設定 Enhanced attribute 後對 eMMC 的影響。Enhanced attribute 的具體作用，由芯片製造商定義。
在實際的產品中，設定 Enhanced storage media 後，一般是把該分區的存儲介質從 MLC 改變為 SLC，提高該分區的讀寫性能、壽命以及穩定性。由於 1 個存儲單元下，MLC 的容量是 SLC 的兩倍，所以在總的存儲單元數量一定的情況下，如果把原本為 MLC 的分區改變為 SLC，會減少 eMMC 的容量，就是說，此時 eMMC 的實際總容量比標稱的總容量會小一點。（MLC 和 SLC 的細節可以參考 Flash Memory 章節內容）

Extended attribute

- Default, 未設定 Extended attribute。
- System code， 設定 GPP 為 System code 屬性，該屬性主要用在存放操作系統類的、很少進行擦寫更新的分區。
- Non-Persistent，設定 GPP 為 Non-Persistent 屬性，該屬性主要用於存儲臨時數據的分區，例如 tmp 目錄所在分區、 swap 分區等。

在 eMMC 標準中，同樣也沒有定義設定 Extended attribute 後對 eMMC 的影響。Extended attribute 的具體作用，由芯片製造商定義。Extended attribute 主要是跟分區的應用場景有關，廠商可以為不用應用場景的分區做不同的優化處理。

5、User Data Area

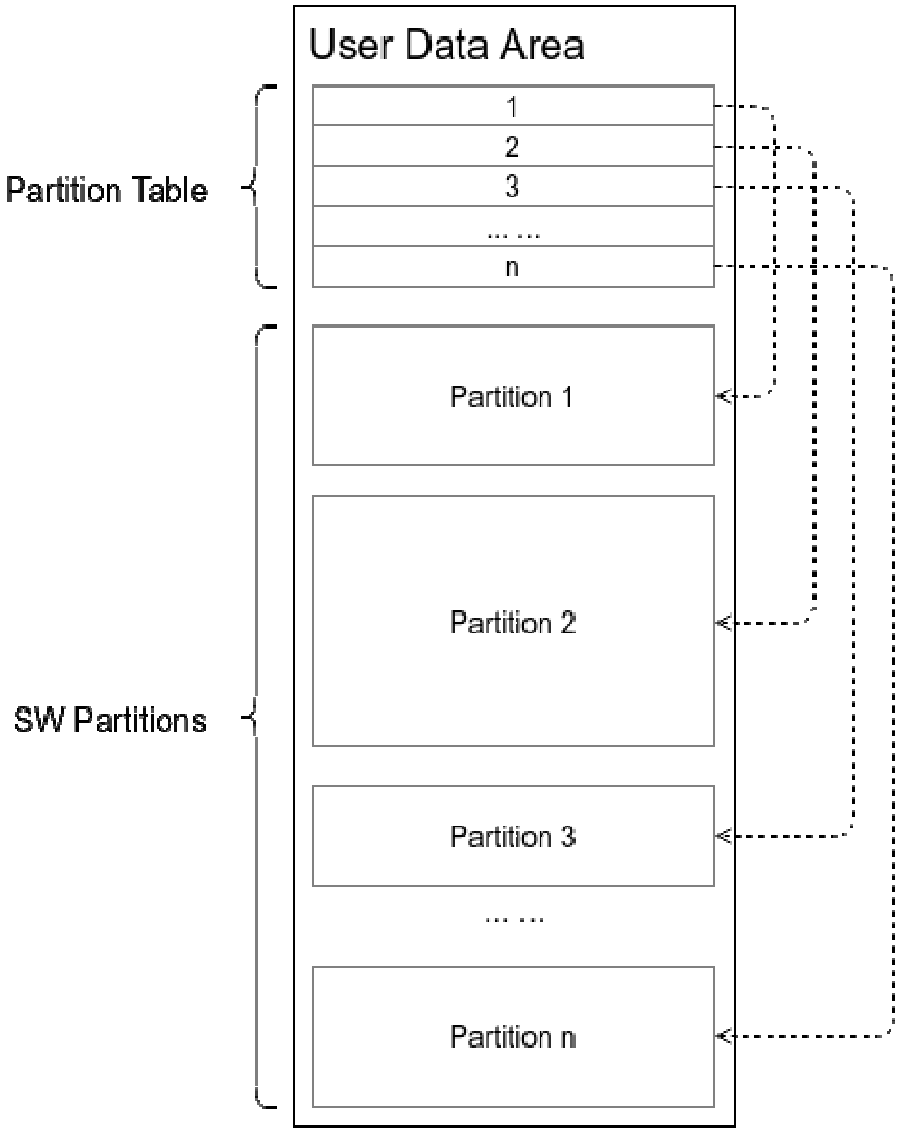
User Data Area (UDA) 通常是 **eMMC 中最大的一個分區，是實際產品中，最主要的存儲區域**。

5.1 容量大小

UDA 的容量大小不需要設置，在配置完其他分區大小後，再扣除設置 Enhanced attribute 所損耗的容量，剩下的容量就是 UDA 的容量。

5.2 軟件分區

為了更合理的管理數據，滿足不同的應用需求，UDA 在實際產品中，會進行軟件再分區。目前主流的軟件分區技術有 **MBR（Master Boot Record）**和 **GPT（GUID Partition Table）**兩種。這兩種分區技術的基本原理類似，如下圖所示：



軟件**分區技術一般是將存儲介質劃分為多個區域，既 SW Partitions，然後通過一個 Partition Table 來維護這些 SW Partitions**。在 Partition Table 中，每一個條目都保存著一個 SW Partition 的起始地址、大小等的屬性信息。軟件系統在啟動後，會去掃瞄 Partition Table，獲取存儲介質上的各個 SW Partitions 信息，然後根據這些信息，將各個 Partitions 加載到系統中，進行數據存取。

MBR 和 GPT 此處不展開詳細介紹，更多的細節可以參考 wikipedia 上 MBR 和 GPT 相關介紹。

5.3 區域屬性

eMMC 標準中，支持為 UDA 中一個特定大小的區域設定 Enhanced attribute。與 GPP 中的 Enhanced attribute 相同，eMMC 標準也沒有定義該區域設定 Enhanced attribute 後對 eMMC 的影響。Enhanced attribute 的具體作用，由芯片製造商定義。

Enhanced attribute

- Default, 未設定 Enhanced attribute。

- Enhanced storage media， 設定該區域為 Enhanced storage media。

在實際的產品中，UDA 區域設定為 Enhanced storage media 後，一般是把該區域的存儲介質從 MLC 改變為 SLC。通常，產品中可以將某一個 SW Partition 設定為 Enhanced storage media，以獲得更好的性能和健壯性。

6、eMMC 分區應用實例

在一個 **Android 手機系統**中，各個分區的呈現形式如下：

- mmcblk0 為 eMMC 的塊設備;
- mmcblk0boot0 和 mmcblk0boot1 對應兩個 Boot Area Partitions;
- mmcblk0rpmb 則為 RPMB Partition，
- mmcblk0px 為 UDA 劃分出來的 SW Partitions;
- 如果存在 GPP，名稱則為 mmcblk0gp1、mmcblk0gp2、mmcblk0gp3、mmcblk0gp4;

```
root@xxx:/ # ls /dev/block/mmcblk0*
/dev/block/mmcblk0
/dev/block/mmcblk0boot0
/dev/block/mmcblk0boot1
/dev/block/mmcblk0rpmb
/dev/block/mmcblk0p1
/dev/block/mmcblk0p2
/dev/block/mmcblk0p3
/dev/block/mmcblk0p4
/dev/block/mmcblk0p5
/dev/block/mmcblk0p6
/dev/block/mmcblk0p7
/dev/block/mmcblk0p8
/dev/block/mmcblk0p9
/dev/block/mmcblk0p10
/dev/block/mmcblk0p11
/dev/block/mmcblk0p12
/dev/block/mmcblk0p13
/dev/block/mmcblk0p14
/dev/block/mmcblk0p15
```

```
/dev/block/mmcblk0p16
/dev/block/mmcblk0p17
/dev/block/mmcblk0p18
/dev/block/mmcblk0p19
/dev/block/mmcblk0p20
```

每一個分區會根據實際的功能來設定名稱。

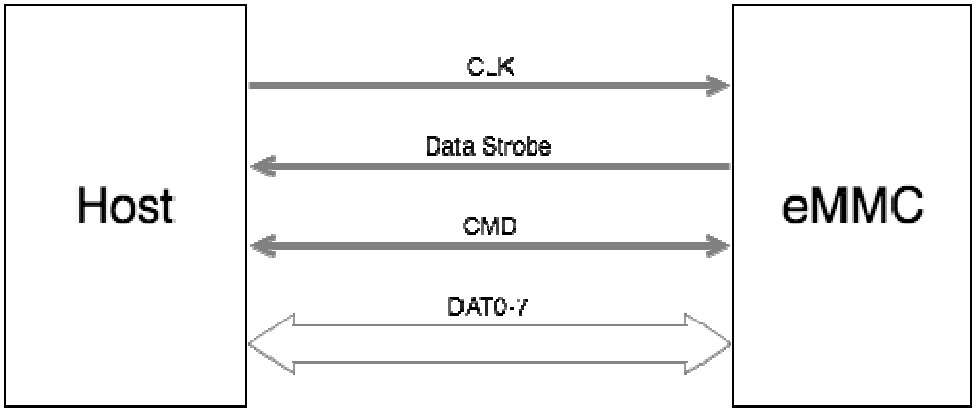
```
root@xxx:/ # ls -l /dev/block/platform/mtk-msdc.0/11230000.msdc0/by-name/
lrwxrwxrwx root root 2015-01-03 04:03 boot -> /dev/block/mmcblk0p22
lrwxrwxrwx root root 2015-01-03 04:03 cache -> /dev/block/mmcblk0p30
lrwxrwxrwx root root 2015-01-03 04:03 custom -> /dev/block/mmcblk0p3
lrwxrwxrwx root root 2015-01-03 04:03 devinfo -> /dev/block/mmcblk0p28
lrwxrwxrwx root root 2015-01-03 04:03 expdb -> /dev/block/mmcblk0p4
lrwxrwxrwx root root 2015-01-03 04:03 flashinfo -> /dev/block/mmcblk0p32
lrwxrwxrwx root root 2015-01-03 04:03 frp -> /dev/block/mmcblk0p5
lrwxrwxrwx root root 2015-01-03 04:03 keystore -> /dev/block/mmcblk0p27
lrwxrwxrwx root root 2015-01-03 04:03 lk -> /dev/block/mmcblk0p20
lrwxrwxrwx root root 2015-01-03 04:03 lk2 -> /dev/block/mmcblk0p21
lrwxrwxrwx root root 2015-01-03 04:03 logo -> /dev/block/mmcblk0p23
lrwxrwxrwx root root 2015-01-03 04:03 md1arm7 -> /dev/block/mmcblk0p17
lrwxrwxrwx root root 2015-01-03 04:03 md1dsp -> /dev/block/mmcblk0p16
lrwxrwxrwx root root 2015-01-03 04:03 md1img -> /dev/block/mmcblk0p15
lrwxrwxrwx root root 2015-01-03 04:03 md3img -> /dev/block/mmcblk0p18
lrwxrwxrwx root root 2015-01-03 04:03 metadata -> /dev/block/mmcblk0p8
lrwxrwxrwx root root 2015-01-03 04:03 nvdata -> /dev/block/mmcblk0p7
lrwxrwxrwx root root 2015-01-03 04:03 nvram -> /dev/block/mmcblk0p19
lrwxrwxrwx root root 2015-01-03 04:03 oemkeystore -> /dev/block/mmcblk0p12
lrwxrwxrwx root root 2015-01-03 04:03 para -> /dev/block/mmcblk0p2
lrwxrwxrwx root root 2015-01-03 04:03 ppl -> /dev/block/mmcblk0p6
lrwxrwxrwx root root 2015-01-03 04:03 proinfo -> /dev/block/mmcblk0p13
lrwxrwxrwx root root 2015-01-03 04:03 protect1 -> /dev/block/mmcblk0p9
lrwxrwxrwx root root 2015-01-03 04:03 protect2 -> /dev/block/mmcblk0p10
lrwxrwxrwx root root 2015-01-03 04:03 recovery -> /dev/block/mmcblk0p1
```

原文：[eMMC 分區管理]

三、eMMC 總線協議

1、 eMMC 總線接口

eMMC 總線接口定義如下圖所示：



各個信號的描述如下：

- **CLK**

CLK 信號用於從 Host 端輸出時鐘信號，**進行數據傳輸的同步和設備運作**的驅動。

在一個時鐘週期內，CMD 和 DAT0-7 信號上都可以支持傳輸 1 個比特，即 **SDR (Single Data Rate) 模式**。此外，DAT0-7 信號還支持配置為 **DDR (Double Data Rate) 模式**，在一個時鐘週期內，可以傳輸 2 個比特。

Host 可以在通訊過程中動態調整時鐘信號的頻率（注，頻率範圍需要滿足 Spec 的定義）。通過調整時鐘頻率，可以實現省電或者數據流控（避免 Over-run 或者 Under-run）功能。 在一些場景中，Host 端還可以關閉時鐘，例如 eMMC 處於 Busy 狀態時，或者接收完數據，進入 Programming State 時。

- CMD

CMD 信號主要用於 **Host 向 eMMC 發送 Command 和 eMMC 向 Host 發送對於的 Response**。Command 和 Response 的細節會在後續章節中介紹。

- DAT0-7

DAT0-7 信號主要用於 **Host 和 eMMC 之間的數據傳輸**。在 eMMC 上電或者軟復位後，只有 DAT0 可以進行數據傳輸，完成初始化後，可配置 DAT0-3 或者 DAT0-7 進行數據傳輸，即數據總線可以配置為 4 bits 或者 8 bits 模式。

- Data Strobe

Data Strobe 時鐘信號**由 eMMC 發送給 Host，頻率與 CLK 信號相同，用於 Host 端進行數據接收的同步**。Data Strobe 信號只能在 HS400 模式下配置啟用，啟用後可以提高數據傳輸的穩定性，省去總線 tuning 過程。

更詳細的工作原理請參考 **eMMC 工作模式** 章節。

2、 eMMC 總線模型

eMMC 總線中**一個 Host 可以有多個 eMMC Devices**。總線上的所有通訊都由 Host 端以一個 Command 開發發起，**Host 一次只能與一個 eMMC Device 通訊**。
系統在上電啟動後，**Host 會為所有 eMMC Device 逐個分配地址（RCA，Relative device Address）**。當 Host 需要和某一個 eMMC Device 通訊時，會先根據 RCA 選中該 eMMC Device，只有被選中的 eMMC Device 才會響應 Host 的 Command。
更詳細的工作原理請參考 **eMMC 工作模式** 章節。

2.1 速率模式

隨著 eMMC 協議的版本迭代，**eMMC 總線的速率越來越高**。為了兼容舊版本的 eMMC Device，所有 Devices 在上電啟動或者 Reset 後，都會先進入兼容速率模式（Backward Compatible Mode）。在完成 eMMC Devices 的初始化後，Host 可以通過特定的流程，讓 Device 進入其他高速率模式，目前支持以下的幾種速率模式。

Mode	Data Rate	Bus Width	Frequency	Max Data Transfer (x8)
Backward Compatible	Single	x1, x4, x8	0-26 MHz	26 MB/s
High Speed SDR	Single	x1, x4, x8	0-52 MHz	52 MB/s
High Speed DDR	Dual	x4, x8	0-52 MHz	104 MB/s
HS200	Single	x4, x8	0-200 MHz	200 MB/s
HS400	Dual	x8	0-200 MHz	400 MB/s

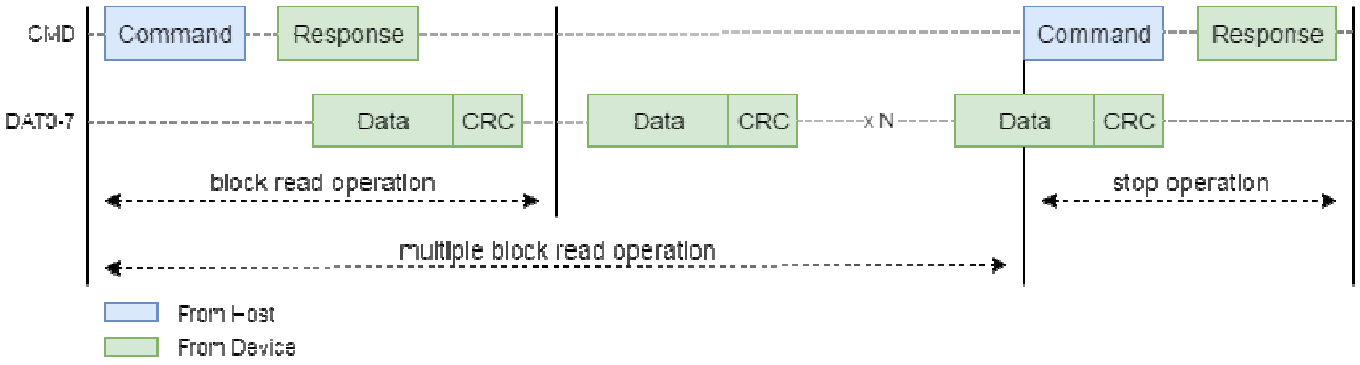
註：

- Extended CSD byte[185] HS_TIMING 寄存器可以配置總線速率模式
- Extended CSD byte[183] BUS_WIDTH 寄存器用於配置總線寬度和 Data Strobe

2.2 通信模型

Host 與 eMMC Device 之間的**通信都是由 Host 以一個 Command 開始發起**的，eMMC Device 在完成 Command 所指定的任務後，則返回一個 Response。

- Read Data

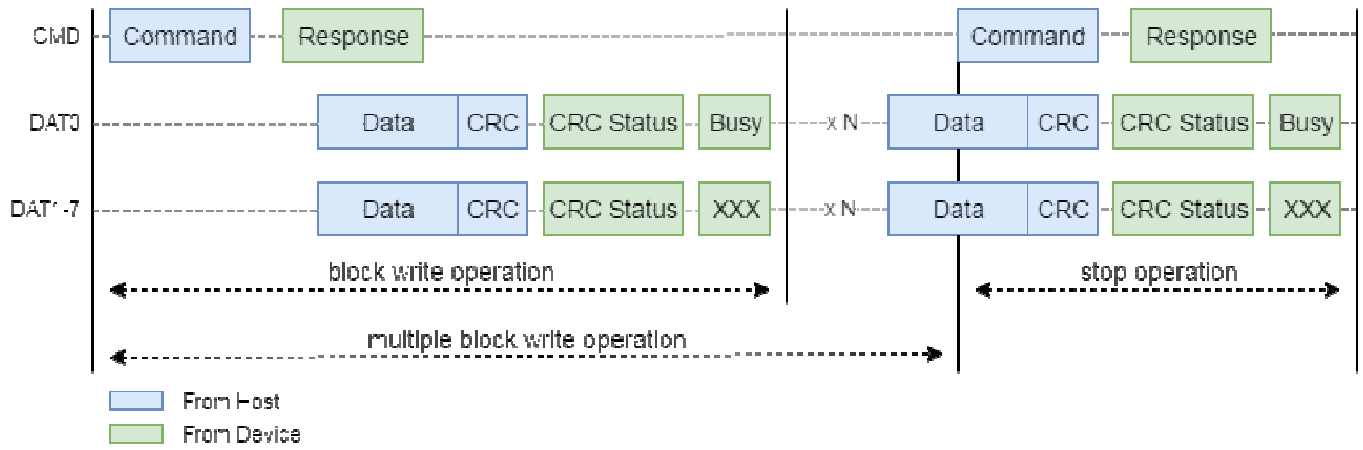


Host 從 eMMC Device 讀取數據的流程如上圖所示。
如果 Host 發送的是 **Single Block Read** 的 Command，那麼 eMMC Device 只會發送一個 Block 的數據。
如果 Host 在發送 **Multiple Block Read** 的 Command 前，先發送一個設定需要讀取的 Block Count 的 Command。eMMC Device 在完成指定 Block Count 的數據發送後，就自動結束數據傳輸，不需要 Host 主動發送 Stop Command。
如果 Host 沒有發送設定需要讀取的 Block Count 的 Command，發送 Multiple Block Read 的 Command 後，eMMC Device 會持續發送數據，直到 Host 發送 Stop Command 停止數據傳輸。

注：

從 eMMC Device 讀數據都是按 Block 讀取的。 Block 大小可以由 Host 設定，或者固定為 512 Bytes，不同的速率模式下有所不同。

- Write Data



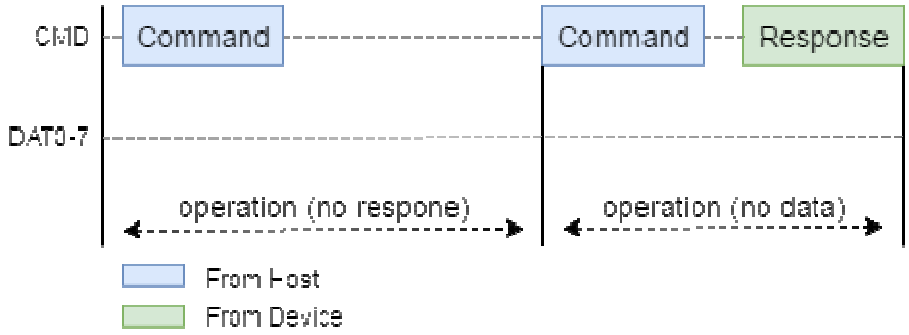
Host 向 eMMC Device 寫入數據的流程如上圖所示。
如果 Host 發送的是 **Single Block Write Command**，那麼 eMMC Device 只會將後續第一個 Block 的數據寫入的存儲器中。
如果 Host 在發送 **Multiple Block Write** 的 Command 前，先發送一個設定需要讀取的 Block Count 的 Command。eMMC Device 在接收到指定 Block Count 的數據後，就自動結束數據接收，不需要 Host 主動發送 Stop Command。
如果 Host 沒有發送設定需要讀取的 Block Count 的 Command，發送 Multiple Block Write 的 Command 後，eMMC Device 會持續接收數據，直到 Host 發送 Stop Command 停止數據傳輸。
eMMC Device 在接收到一個 Block 的數據後，會進行 CRC 校驗，然後將校驗結果通過 CRC Token 發送給 Host。
發送完 CRC Token 後，如果 CRC 校驗成功，eMMC Device 會將數據寫入到內部存儲器時，此時 DAT0 信號會拉低，作為 Busy 信號。Host 會持續檢測 DAT0 信號，直到為高電平時，才會接著發送下一個 Block 的數據。如果 CRC 校驗失敗，那麼 eMMC Device 不會進行數據寫入，此次傳輸後續的數據都會被忽略。

注：

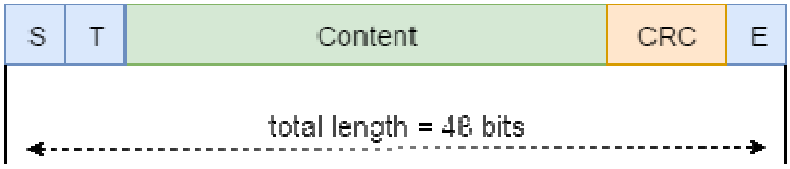
向 eMMC Device 寫數據都是按 Block 寫入的。 Block 大小可以由 Host 設定，或者固定為 512 Bytes，不同的速率模式下有所不同。

- No Data

在 Host 與 eMMC Device 的通信中，有部分交互是不需要進行數據傳輸的，還有部分交互甚至不需要 eMMC Device 的回覆 Response。



- Command



如上圖所示，**eMMC Command** 由 **48 Bits** 組成，各個 **Bits** 的解析如下所示：

Description	Start Bit	Transmission Bit	Command Index	Argument	CRC7	End Bit
Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	"0"	"1"	x	x	x	"1"

Start Bit 固定為 "0"，在沒有數據傳輸的情況下，**CMD 信號保持高電平**，當 Host 將 Start Bit 發送到總線上時，**eMMC Device** 可以很方便檢測到該信號，並開始接收 Command。

Transmission Bit 固定為 "1"，指示了該數據包的傳輸方向為 Host 發送到 eMMC Device。

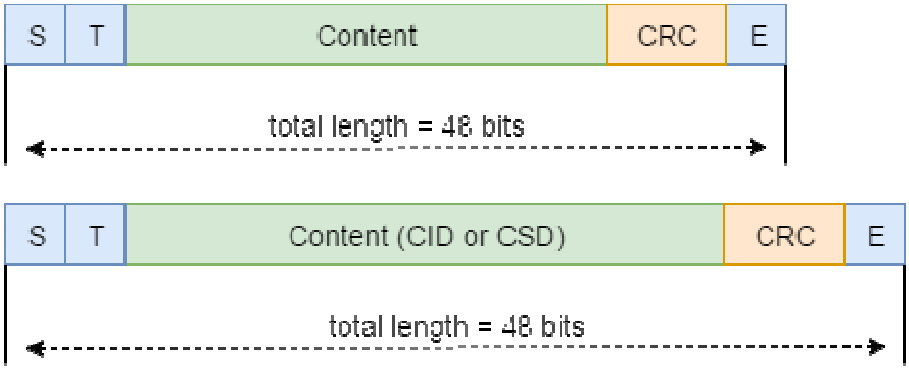
Command Index 和 Argument 為 Command 的具體內容，不同的 Command 有不同的 Index，不同的 Command 也有各自的 Argument。 更多的細節，請參考 **eMMC Commands** 章節。

CRC7 是包含 Start Bit、Transmission Bit、 Command Index 和 Argument 內容的 CRC 校驗值。
End Bit 為結束標誌位，固定為"1"。

注：

CRC 校驗簡單來說，是發送方將需要傳輸的數據「除於」（模 2 除）一個約定的數，並將得到的餘數附在數據上一併發送出去。接收方收到數據後，再做同樣的「除法」，然後校驗得到餘數是否與接收的餘數相同。如果不相同，那麼意味著數據在傳輸過程中發生了改變。更多的細節不在本文展開描述，感興趣的讀者可以參考 CRC wiki 中的介紹。

- Response



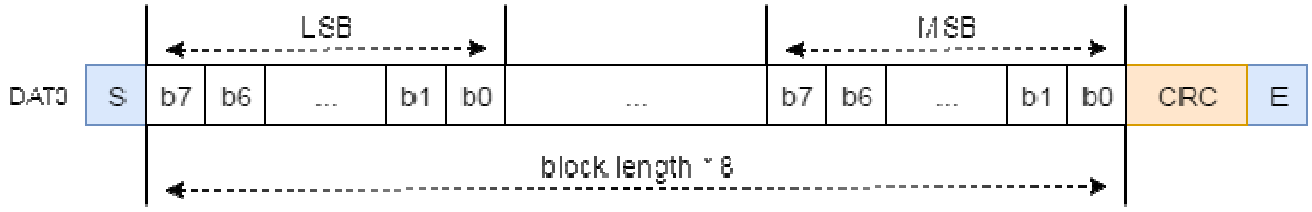
eMMC Response 有兩種長度的數據包，分別為 48 Bits 和 136 Bits。
Start Bit 與 Command 一樣，固定為 "0"，在沒有數據傳輸的情況下，CMD 信號保持高電平，當 eMMC Device 將 Start Bit 發送到總線上時，Host 可以很方便檢測到該信號，並開始接收 Response。

Transmission Bit 固定為 "0"，指示了該數據包的傳輸方向為 eMMC Device 發送到 Host。

Content 為 Response 的具體內容，不同的 Command 會有不同的 Content。 更多的細節，請參考 **eMMC Responses** 章節。
CRC7 是包含 Start Bit、Transmission Bit 和 Content 內容的 CRC 校驗值。
End Bit 為結束標誌位，固定為"1"。

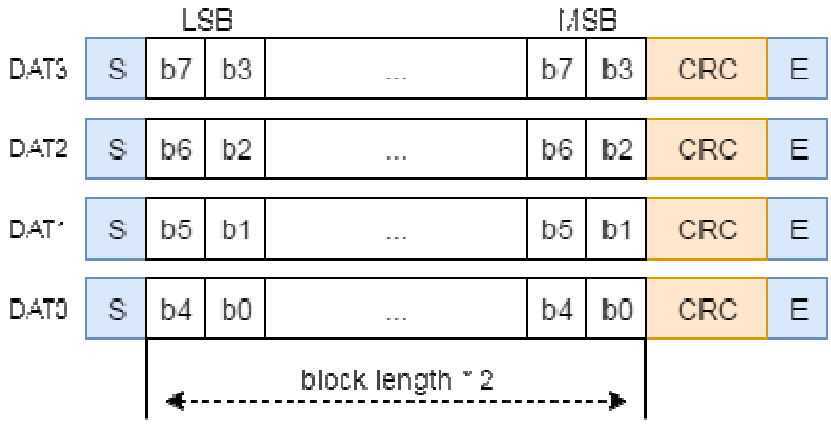
- Data Block

Data Block 由 Start Bit、Data、CRC16 和 End Bit 組成。以下是不同總線寬度和 Data Rate 下，Data Block 詳細格式。
1 Bit Bus SDR



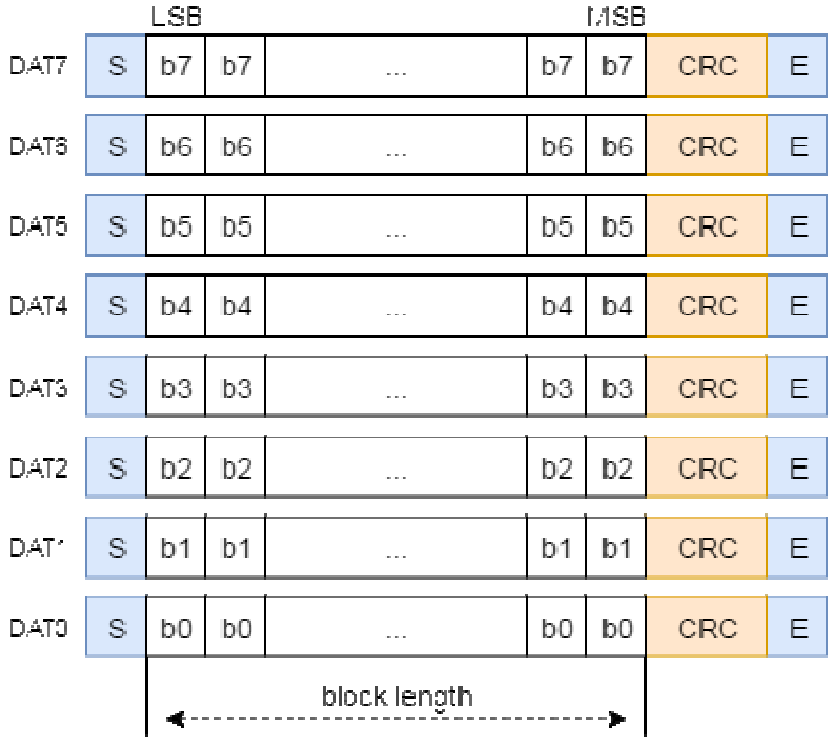
CRC 為 Data 的 16 bit CRC 校驗值，不包含 Start Bit。

4 Bits Bus SDR



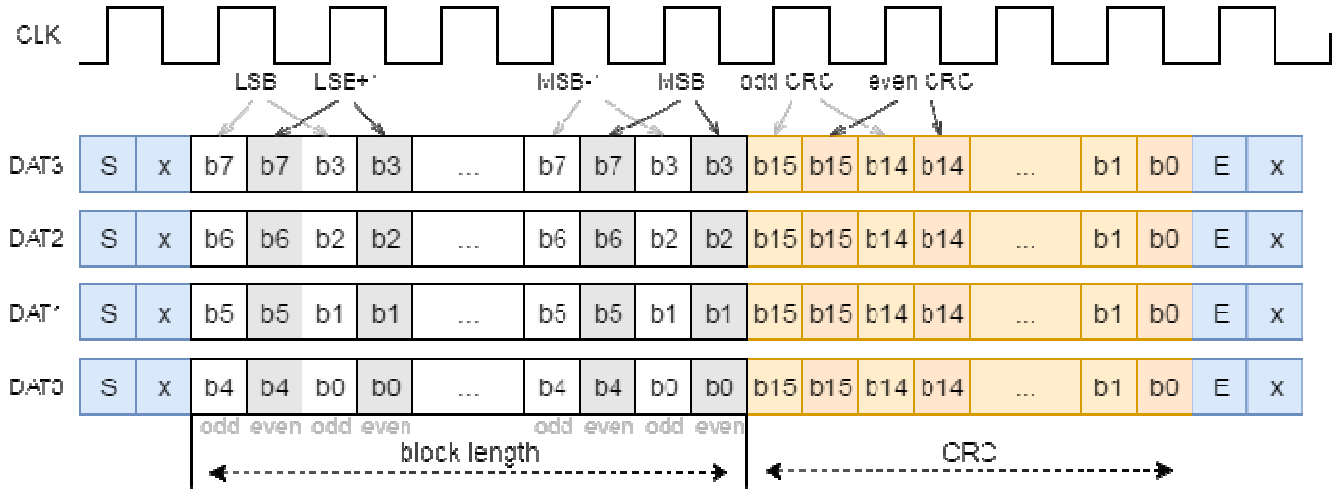
各個 Data Line 上的 CRC 為對應 Data Line 的 Data 的 16 bit CRC 校驗值。

8 Bits Bus SDR

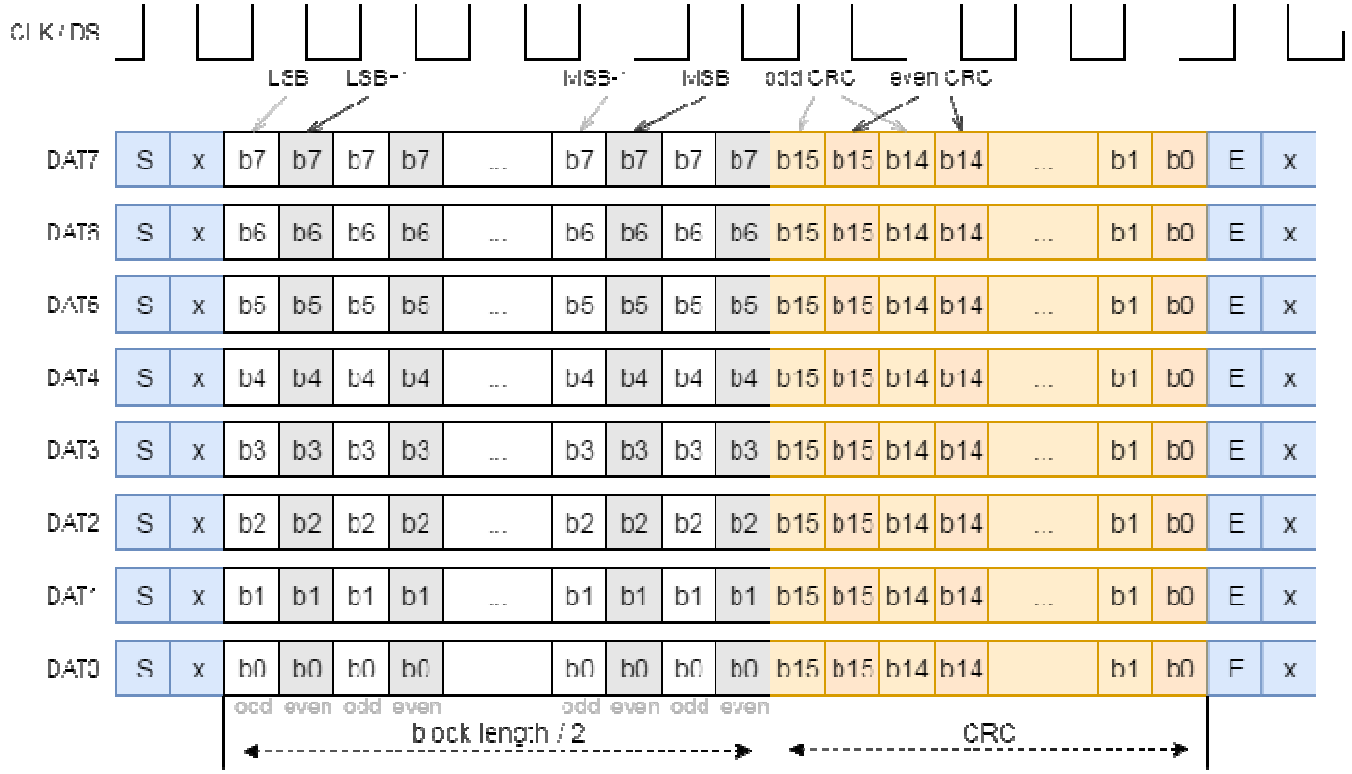


各個 Data Line 上的 CRC 為對應 Data Line 的 Data 的 16 bit CRC 校驗值。

4 Bits Bus DDR



8 Bits Bus DDR



在 **DDR 模式下，Data Line 在時鐘的上升沿和下降沿都會傳輸數據**，其中上升沿傳輸數據的奇數字節 （Byte 1,3,5 ...），下降沿則傳輸數據的偶數字節（Byte 2,4,6 ...）。
此外，在 DDR 模式下，1 個 Data Line 上有兩個相互交織的 CRC16，上升沿的 CRC 比特組成 odd CRC16，下降沿的 CRC 比特組成 even CRC16。odd CRC16 用於校驗該 Data Line 上所有上升沿比特組成的數據，even CRC16 則用於校驗該 Data Line 上所有下降沿比特組成的數據。

註：

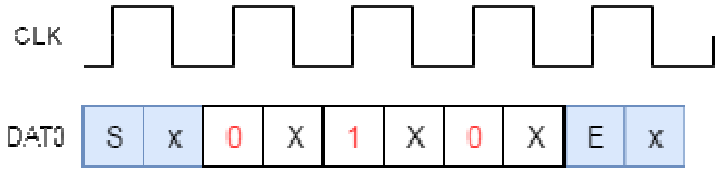
DDR 模式下使用兩個 CRC16 作為校驗，可能是為了更可靠的校驗，選用 CRC16 而非 CRC32 則可能是出於兼容性設計的考慮。

- CRC Status Token

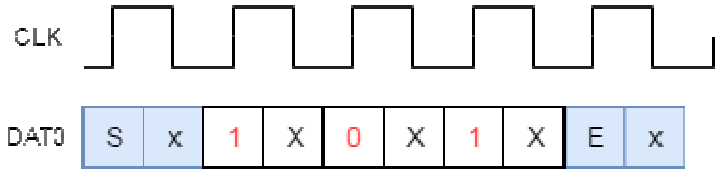
在寫數據傳輸中，eMMC Device 接收到 Host 發送的一個 Data Block 後，會進行 CRC 校驗，如果校驗成功，eMMC 會在對應的 Data Line 上向 Host 發回一個 Positive CRC status token (010)，如果校驗失敗，則會在對應的 Data Line 上發送一個 Negative CRC status token (101)。

讀數據時，Host 接收到 eMMC Device 發送的 Data Block 後，也會進行 CRC 校驗，但是不管校驗成功或者失敗，都不會向 eMMC Device 發送 CRC Status Token。

Positive CRC status token



Negative CRC status token

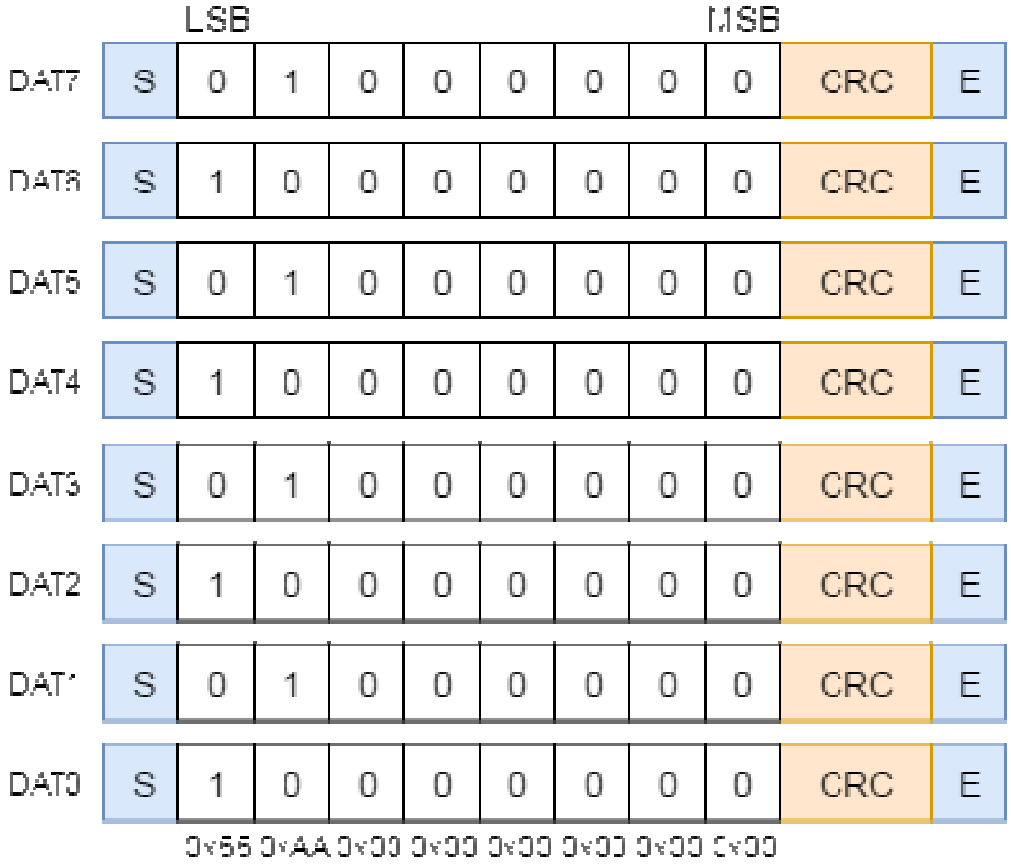


3、eMMC 總線測試過程

當 eMMC Device 處於 SDR 模式時，**Host 可以發送 CMD19 命令，觸發總線測試過程（Bus testing procedure）**，測試總線硬件上的連通性。如果 eMMC Device 支持總線測試，那麼 eMMC Device 在接收到 CMD19 後，會發回對應的 Response，接著 eMMC Device 會發送一組固定的測試數據給 Host。Host 接收到數據後，檢查數據正確與否，即可得知總線是否正確連通。

如果 eMMC Device 不支持總線測試，那麼接收到 CMD19 時，不會發回 Response。
總線測試不支持在 DDR 模式下進行。

測試數據如下所示：



總線寬度為 1 時，只發送 DAT0 上的數據，總線寬度為 4 時，則只發送 DAT0-3 上的數據

4、eMMC 總線 Sampling Tuning

由於芯片製造工藝、PCB 走線、電壓、溫度等因素的影響，數據信號從 eMMC Device 到達 Host 端的時間是存在差異的，Host 接收數據時採樣的時間點也需要相應的進行調整。而 **Host 端最佳採樣時間點，則是通過 Sampling Tuning 流程得到**。

不同 eMMC Device 最佳的採樣點可能不同，同一 eMMC Device 在不同的環境下運作時的最佳採樣點也可能不同。
在 eMMC 標準中，定義了在 HS200 模式下可以進行 Sampling Tuning。

4.1 Sampling Tuning 流程

Sampling Tuning 是用於計算 Host 最佳採樣時間點的流程，大致的流程如下：

1. Host 將採樣時間點重置為默認值
2. Host 向 eMMC Device 發送 Send Tuning Block 命令
3. eMMC Device 向 Host 發送固定的 Tuning Block 數據

4. Host 接收到 Tuning Block 並進行校驗
5. Host 修改採樣時點，重新從第 2 步開始執行，直到 Host 獲取到一個有效採樣時間點區間
6. Host 取有效採樣時間點區間的中間值作為採樣時間點，並推出 Tuning 流程

上述流程僅僅是一個示例。Tuning 流程執行的時機、頻率和具體的步驟是由 Host 端的 eMMC Controller 具體實現而定的。

4.2 Tuning Block 數據

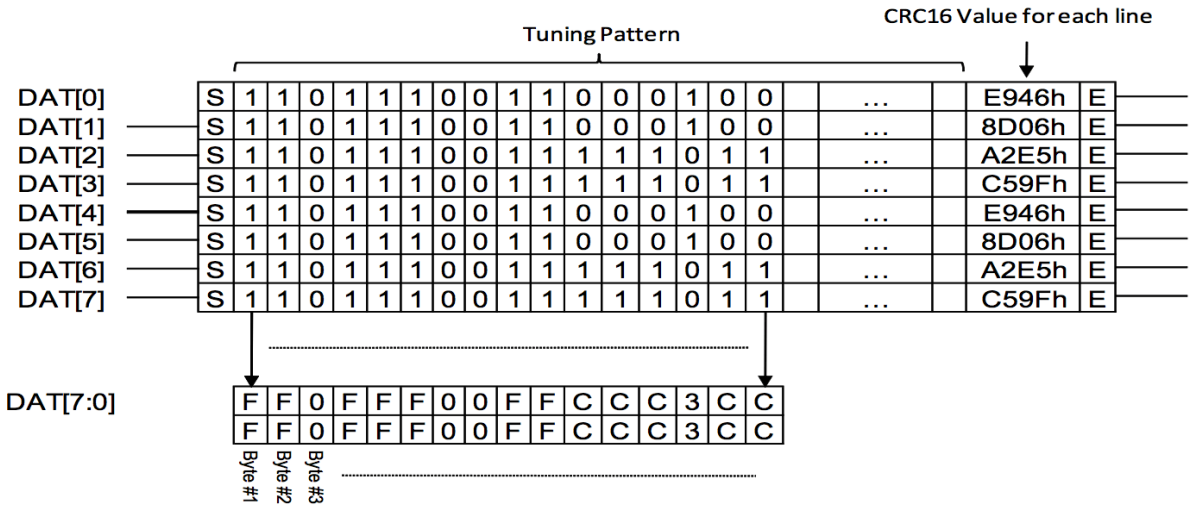
Tuning Block 是專門為了 Tuning 而設計的一組特殊數據。相對於普通的數據，這組特殊數據在傳輸過程中，會更高概率的出現 high SSO noise、deterministic jitter、ISI、timing errors 等問題。這組數據的具體內容如下所示：

Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #9	Byte #10	Byte #11	Byte #12	Byte #13	Byte #14	Byte #15	Byte #16	Byte #17	Byte #18	Byte #19	Byte #20	Byte #21	Byte #22	Byte #23	Byte #24	Byte #25	Byte #26	Byte #27	Byte #28	Byte #29	Byte #30	Byte #31	Byte #32
F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E	F
F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E	F

Byte #33	Byte #34	Byte #35	Byte #36	Byte #37	Byte #38	Byte #39	Byte #40	Byte #41	Byte #42	Byte #43	Byte #44	Byte #45	Byte #46	Byte #47	Byte #48	Byte #49	Byte #50	Byte #51	Byte #52	Byte #53	Byte #54	Byte #55	Byte #56	Byte #57	Byte #58	Byte #59	Byte #60	Byte #61	Byte #62	Byte #63	Byte #64
F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E	F
F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E	F

Byte #65	Byte #66	Byte #67	Byte #68	Byte #69	Byte #70	Byte #71	Byte #72	Byte #73	Byte #74	Byte #75	Byte #76	Byte #77	Byte #78	Byte #79	Byte #80	Byte #81	Byte #82	Byte #83	Byte #84	Byte #85	Byte #86	Byte #87	Byte #88	Byte #89	Byte #90	Byte #91	Byte #92	Byte #93	Byte #94	Byte #95	Byte #96
F	F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E
F	F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E

Byte #97	Byte #98	Byte #99	Byte #100	Byte #101	Byte #102	Byte #103	Byte #104	Byte #105	Byte #106	Byte #107	Byte #108	Byte #109	Byte #110	Byte #111	Byte #112	Byte #113	Byte #114	Byte #115	Byte #116	Byte #117	Byte #118	Byte #119	Byte #120	Byte #121	Byte #122	Byte #123	Byte #124	Byte #125	Byte #126	Byte #127	Byte #128
F	F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E
F	F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E



總線寬度為 1 時，只發送 DAT0 上的數據，總線寬度為 4 時，則只發送 DAT0-3 上的數據

原文：eMMC 總線協議

四、eMMC 工作模式

1、Overview

eMMC Device 在 Power On、HW Reset 或者 SW Reset 時，Host 可以觸發 eMMC Boot，讓 eMMC 進入 **Boot Mode**。在此模式下，eMMC Device 會將 Boot Data 發送給 Host，這部分內容通常為系統的啟動代碼，如 BootLoader。

如果 Host 沒有觸發 Boot 流程或者 Boot 流程完成後，eMMC Device 會進入 **Device Identification Mode**。在此模式下，eMMC Device 將進行初始化，Host 會為 eMMC Device 設定工作電壓、協商尋址模式以及分配 RCA 設備地址。

Device Identification Mode 結束後，就會進入 **Data Transfer Mode**。在此模式下，Host 可以發起數據讀寫流程。

進入 Data Transfer Mode 後，Host 可以發起命令，讓 eMMC Device 進入 **Interrupt Mode**。在此模式下，eMMC Device 會等待內部的中斷事件，例如，寫數據完成等。eMMC Device 在收到內部中斷事件時，會向 Host 發送 Response，然後切換到 Data Transfer Mode，等待 Host 後續的數據讀寫命令。

2、 Boot Operation Mode

2.1 Boot From eMMC Device

在 Power On、HW Reset 或者 SW Reset 後，如果 eMMC Device 有使能 Boot Mode（即，寄存器位 BOOT_PARTITION_ENABLE (EXT_CSD byte [179]) 指定了啟動分區），那麼 Host 有兩種方式可以讓 eMMC Device 進入 Boot Mode，分別定義為 Original Boot 和 Alternative Boot，如下：

1. Original Boot：拉低 CMD 信號並保持不少於 74 個時鐘週期
2. Alternative Boot：保持 CMD 信號為高電平，74 個時鐘週期後，發送參數為 0xFFFFFFFA 的 CMD0 命令

進入 Boot Mode 後，eMMC Device 會根據寄存器位 BOOT_PARTITION_ENABLE 的設定，從兩個 Boot partitions 和 UDA 中選擇一個分區讀取大小為 128KB × BOOT_SIZE_MULT (EXT_CSD byte [226]) 的 Boot Data 通過 Data Lines 發送給 Host。

在 Boot Data 數據傳輸過程中，Host 可以打斷數據傳輸，提前結束 Boot Mode，方法如下：

1. Original Boot：傳輸過程中，拉高 CMD 信號
2. Alternative Boot：傳輸過程中，發送參數為 0xF0F0F0F0 的 CMD0 命令

Host 發送參數為 0xF0F0F0F0 的 CMD0 命令，可以讓 eMMC Device 進行 SW Reset。

Host 拉高 RST_n 信號可以觸發 eMMC Device 進行 HW Reset。

2.2 Boot Acknowledge

如果寄存器位 BOOT_ACK (EXT_CSD byte [179]) 被設定為 1， eMMC Device 會在 Host 觸發 Boot Mode 的 50 ms 內，在 DAT0 上發送一個 "010" Boot ACK 給 Host。

2.3 Boot Bus 配置

EXT_CSD byte [177] BOOT_BUS_CONDITIONS 寄存器用於配置在 Boot Mode 時，數據傳輸的總線狀態。
通過 BOOT_BUS_CONDITIONS 寄存器配置，在 Boot Mode 時，總線可以支持以下幾種模式：

Mode	Data Rate	Bus Width	Frequency	Max Data Transfer (x8)
Backward Compatible	Single	x1, x4, x8	0-26 MHz	26 MB/s
High Speed SDR	Single	x1, x4, x8	0-52 MHz	52 MB/s
High Speed DDR	Dual	x4, x8	0-52 MHz	104 MB/s

BOOT_BUS_CONDITIONS 寄存器還可以配置退出 Boot Mode 後，是復位還是保留當前總線配置。如果配置為復位，那麼退出 Boot Mode 後，總線會被覆位為 Backward Compatible SDR x1 模式，如果配置為保留，那麼退出 Boot Mode 後，總線會保留 Boot Mode 時的總線模式。

BOOT_BUS_CONDITIONS 寄存器為 nonvolatile 屬性，配置內容掉電不會丟失。
如果 eMMC Device 沒有經過 Boot Mode，BOOT_BUS_CONDITIONS 寄存器不會改變總線模式。
退出 Boot Mode 後，還可以通過 HS_TIMING 和 BUS_WIDTH 寄存器配置總線模式。

2.4 Boot Data 更新

eMMC Device 在從廠商出貨時，沒有存儲內容，也沒有使能 Boot Mode。使用 eMMC Devcie 產品需要先通過其他的方式（例如，通過 USB、UART 等）啟動一個下載系統，將 Boot Data 以及其他的系統數據寫入到 eMMC 中，同時使能 Boot Mode 並設定 Boot Bus 模式。而後，產品才能從 eMMC Device 上啟動軟件系統。
Boot Data 的更新與其他數據的寫入類似，更多的數據寫入細節，請參考 **Data Transfer Mode** 小節。

3、Device Identification Mode

如果 Host 沒有觸發 Boot 流程或者 Boot 流程完成後，eMMC Device 會進入 Device Identification Mode。
eMMC Device 在退出 Boot Mode 後或者沒使能 Boot Mode 時 Power On、HW Reset 或者 SW Reset 後，會進入 Device Identification Mode 的 **Idle State**。
在 Idle State 下，eMMC Device 會進行內部初始化，Host 需要持續發送 CMD1 命令，查詢 eMMC Device 是否已經完成初始化，同時進行工作電壓和尋址模式協商。
Host 發送的 CMD1 命令的參數中，包含了 Host 所支持的工作電壓和尋址模式信息，eMMC Device 在接收到這些信息後，會進行匹配。如果 eMMC Devcie 和 Host 所支持的工作電壓和尋址模式不匹配，那麼 eMMC Device 會進入 **Inactive State**。
eMMC Device 在接收到 CMD1 命令後，會將 OCR register 的內容作通過 Response 返回給 Host，其中包含了 eMMC Device 是否完成初始化的標誌位、設備工作電壓範圍 Voltage Range 和存儲訪問模式 Memory Access Mode 信息。
eMMC Device 完成初始化後，就會進入 **Ready State**。在該 State 下，Host 會發送 CMD2 命令，獲取 eMMC Device 的 CID。

CID，即 Device identification number，用於標識一個 eMMC Device。它包含了 eMMC Device 的製造商、OEM、設備名稱、設備序列號、生產年份等信息，每一個 eMMC Device 的 CID 都是唯一的，不會與其他的 eMMC Device 完全相同。
eMMC Device 接收到 CMD2 後，會將 127 Bits 的 CID register 的內容通過 Response 返回給 Host。
發送完 CID 後，eMMC Device 接著就會進入 Identification State。而後，Host 會發送參數包含 16 Bits RCA 的 CMD3 命令，為 eMMC Device 分配 RCA。
設定完 RCA 後，eMMC Devcie 就完成了 Devcie Identification，進入 Data Transfer Mode。

節只描述了單個 eMMC Device 的 Devcie Identification 過程，多個 eMMC 的 Device Identification 過程與此類似，更多的細節可以參考 eMMC Spec。

3.1 Voltage Range

eMMC Device 支持 3.3v 和 1.8v 兩種工作電壓模式。在 1.8v 模式下，eMMC Device 會更加的省電。

3.2 Memory Access Mode

Memory Access Mode 決定了 eMMC Device 在響應 Host 的數據讀寫請求時，是如何訪問內部存儲器的。在 eMMC 標準中存在兩種 Memory Access Mode：**Byte Access Mode** 和 **Sector Access Mode**。

在數據讀寫的 Command 中，Host 會將讀寫的地址 A 作為 Command 的參數發送給 eMMC Device，在 Byte Access Mode 下，eMMC Device 將從第 A 個 Byte 開始進行讀寫操作，而在 Sector Access Mode 下，eMMC Device 將會從第 A 個 Sector 開始進行讀寫操作，一個 Sector 的大小為 512 Bytes 或者 4 KBytes，更大的 Sector 支持更大容量的存儲器訪問。

使用 Byte Access Mode 更加的靈活高效，但是由於尋址位數的限制，不能訪問超過 2GB 的存儲內容。Sector Access Mode 則支持大容量存儲的訪問，其中 512 Bytes Sector 可以支持最大 256 GB 容量的存儲訪問，更大容量的需求則可以使用 4 KBytes Sector。

3.3 RCA - Relative device Address

RCA 是在 **Devcie Identification 過程中，由 Host 分配的 16 Bits 的設備地址**，主要用於 Data Transfer Mode 下進行通信時，選定具體要進行操作的 eMMC Devcie。

Host 分配的 RCA 通常從 1 開始遞增，0 地址作為廣播地址。eMMC Devcie 的 RCA register 保存了 Host 分配的 RCA。

3.4 Data Transfer Mode

eMMC Device 完成 Device Identification 後，就會進入到 Data Transfer Mode 的 Standby State。

在 Standby State 時，Host 可以通過發送 CMD5 命令，讓 eMMC Devcie 進入 低功耗的 Sleep State，而後再發送 CMD5 命令則可以讓 eMMC Device 退出 Sleep State。

在 Standby State 時，Host 可以通過發送 CMD7 命令，讓 eMMC Devcie 進入 Transfer State，而後再發送 CMD7 命令則可以讓 eMMC Device 退出 Transfer State。

3.5 Read Data

在 Transfer State 時，Host 可以發送以下的命令，觸發數據讀取流程：

命令	描述
CMD8	读取 EXT_CSD 寄存器数据
CMD17	从指定的地址开始，读取一个 Block 的数据
CMD18	从指定的地址开始，读取多个 Block 的数据
CMD21	读取 Tuning Block 的数据

eMMC Device 在接收到上述幾個 CMD 時，就會進入 **Sending-data State**。在此 State 下，eMMC Device 會持續將數據發送給 Host，直到指定數量的數據 Block 傳輸完成或者接收到 Host 發送的 CMD12 傳輸停止命令。eMMC Device 在停止發送數據後，會返回到 Transfer State。
如果 Host 在發送 CMD18 前，先發送一個設定需要讀取的 Block Count 的 CMD23。eMMC Device 在完成指定 Block Count 的數據發送後，就自動結束數據傳輸，不需要 Host 主動發送停止命令 CMD12。
如果 Host 沒有發送設定需要讀取的 Block Count 的 Command，發送 Multiple Block Read 的 Command 後，eMMC Device 會持續發送數據，直到 Host 發送 Stop Command 停止數據傳輸。

如果在發送 CMD18 前，先發送 CMD23 設定需要讀取的 Block Count，那麼 eMMC Device 會在發送完指定數量的 Block 後，自動停止發送數據。

3.6 Write Data

在 Transfer State 時，Host 可以發送以下的命令，觸發數據寫入流程：

命令	描述
CMD24	写入一个 Block 的数据
CMD25	写入多个 Block 的数据
CMD26	写入 CID 寄存器值
CMD27	写入 CSD 寄存器值

CID 寄存器值通常是只能寫一次，由廠家在生產時確定並寫入 CSD 寄存器值的部分位則可以多次改寫。
eMMC Device 在接收到上述幾個 CMD 時，就會進入 **Receive-data State**，在此 State 下，eMMC Devcie 會持續從 Host 接收數據，並存儲到內部的 Buffer 或者寄存器中。
如果 Host 在發送 CMD25 前，先發送一個設定需要寫入的 Block Count 的 CMD23。eMMC Device 在完成指定 Block Count 的數據接收後，就自動結束數據傳輸，不需要 Host 主動發送停止命令 CMD12。
如果 Host 沒有發送設定需要寫入的 Block Count 的 Command，發送 Multiple Block Write 的 Command 後，eMMC Device 會持續接收數據，直到 Host 發送 Stop Command 停止數據傳輸。
eMMC Device 在開始進行寫入操作時，會先將接收到的數據存儲在內部 Buffer 中，然後在後台將 Buffer 中的數據寫入到 Flash 中。通常情況下，Host 發送數據的速度會比

eMMC 寫入 Flash 的速度快，所以內部的 Buffer 會出現寫滿的狀態，此時 eMMC Devcie 會將 DAT0 信號線拉低作為 Busy 信號。Host 收到 Busy 信號後，就會暫停發送數據，等到 eMMC Device 將 Buffer 中的數據處完一部分並解除 Busy 信號後，再重新發送數據。

當 eMMC Device 完成數據接收後，就會進入到 Programming State，將內部 Buffer 中剩餘未寫入的數據寫入到 Flash 中。 在該 State 下，eMMC Device 會持續將 DAT0 拉低，作為 Busy 信號。如果在完成寫入前，有收到新的寫入命令，那麼 eMMC Device 會立刻退回到 Receive-data State，進行數據接收；如果在完成寫入前，沒有收到新的寫入命令，則會在完成寫入後，退回到 Transfer State。

如果 eMMC Devcie 在 Programming State 時，還沒有完成寫入操作，就收到參數不等於自身 RCA 的 CMD7 命令，那麼 eMMC Device 會進入到 Disconnect State。在該 State 下，eMMC Device 會繼續進行寫入操作，寫入完成後則進入到 Stand-by State。

如果 eMMC Device 在 Disconnect State 時，還沒有完成寫入操作，就收到參數等於自身 RCA 的 CMD7 命令，那麼 eMMC Devcie 會從新回到 Programming State。

3.7 Packed Commands - Packed Read and Packed Write

eMMC 標準中，支持將對多個不連續地址的數據讀取或者寫入。

原文：[eMMC 工作模式]