


硬核处理器系统 (HPS) 提供一个安全数字 / 多媒体卡 (SD/MMC) 控制器，用于连接外部 SD 和 MMC 闪存卡，安全数字 I/O (SDIO) 器件和消费电子高级传输体系结构 (CE-ATA) 硬盘。SD/MMC 控制器使您能够存储器引导映像并从可移动闪存卡引导处理器系统。对于大型应用或者用户数据，您也可以使用闪存卡扩展板上存储容量。其它应用包括连接嵌入式 SD (eSD) 和嵌入式 MMC (eMMC) 不可移动闪存器件。

SD/MMC 控制器基于 Synopsys® DesignWare® 移动存储主机 (DWC_mobile_storage) 控制器。

 本文档使用 SD/SDIO 命令，在 *Physical Layer Simplified Specification, Version 3.01* 和第 11 - 67 页的 “参考文献” *SDIO Simplified Specification Version 2.00* 中有详细介绍。

SD/MMC 控制器的特性

HPS SD/MMC 控制器具有以下特性：

- 支持从移动存储器的 HPS 引导
- 支持下面的标准或卡类型：
 - SD，包括 eSD—3.0 版本
 - SDIO，包括嵌入式 SDIO (eSDIO)—3.0 版本
 - CE-ATA—1.1 版本
 - MMC，包括 eMMC—4.41 版本，1-bit，4-bit 和 8-bit（在某些封装中，如第 11 - 2 页的表 11 - 2 中所描述）
- 集成的基于描述符的直接存储器访问 (DMA)
- 内部的 4 KB 接收和发送 FIFO 缓存

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Portions © 2011 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.



表 11-1 显示了多种 SD 卡器件类型和支持电压，总线模式和速度。

SD/MMC 控制器不直接支持 eSDIO 卡器件的电压切换，卡中断或者后端电源控制 (back-end power control)。然而，这些信号可以连接到通用 I/O (GPIO)。

表 11-1. SD 卡使用情况

卡器件类型	支持的电压		支持的总线模式				支持的总线速度模式			
							默认速度	高速	SDR12	SDR25 ⁽¹⁾
	3.3 V	1.8 V	SPI	1 bit	4 bit	8 bit	12.5 MBps 25 MHz	25 MBps 50 MHz	12.5 MBps 25 MHz	25 MBps 50 MHz
SDSC (SD)	✓	—	✓	✓	—	—	✓	✓	—	—
SDHC	✓	✓ ⁽²⁾	✓	✓	✓	—	✓	✓	✓	✓
SDXC	✓	✓ ⁽²⁾	✓	✓	✓	—	✓	✓	✓	✓
eSD	✓	✓ ⁽²⁾	✓	✓	✓	—	✓	✓	✓	✓
SDIO	✓	✓ ⁽²⁾	✓	✓	✓	—	✓	✓	✓	✓
eSDIO	✓	✓ ⁽²⁾	✓	✓	✓	✓ ⁽³⁾	✓	✓	✓	✓

表 11-1 注释：

- (1) SDR25 速度模式需要 1.8-V 信号。要注意即便一个卡支持 UHS-I 模式（例如：SDR50，SDR104，DDR50）但它仍然在较低速度上进行通信（例如 SDR12，SDR25）。
- (2) 控制电压切换输出以支持 1.8-V 信号用于 SD。
- (3) 所有的 FPGA 封装都不支持 eSDIO 可选的 8-bit 总线模式。

 卡形状因素（例如小型和微型）没有在表 11-1 中列举，因为它们不影响卡接口功能。

表 11-2 显示了多种 MMC 卡器件类型和支持电压，总线模式和总线速度。

作为外部卡接口一部分，SD/MMC 控制器不包括复位输出。考虑使用一个通用输出管脚来复位闪存卡器件。

表 11-2. MMC 使用情况

卡器件类型	最大时钟速度 (MHz)	最大数据速率 (MBps)	支持的电压		支持的总线模式				支持的总线速度模式	
			3.3 V	1.8 V	SPI ⁽¹⁾	1 bit	4 bit	8 bit	默认速度	高速度
MMC	20	2.5	✓	—	✓	✓	—	—	✓	—
RSMMC	20	10	✓	—	✓	✓	✓	—	✓	✓
MMCPlus	50 ⁽³⁾	25	✓	—	—	✓	✓	✓ ⁽²⁾	✓	✓
MMCMobile	50	6.5	✓	✓	—	✓	—	—	✓	✓
eMMC	50	25	✓	✓	—	✓	✓	—	✓	✓

表 11-2 注释：

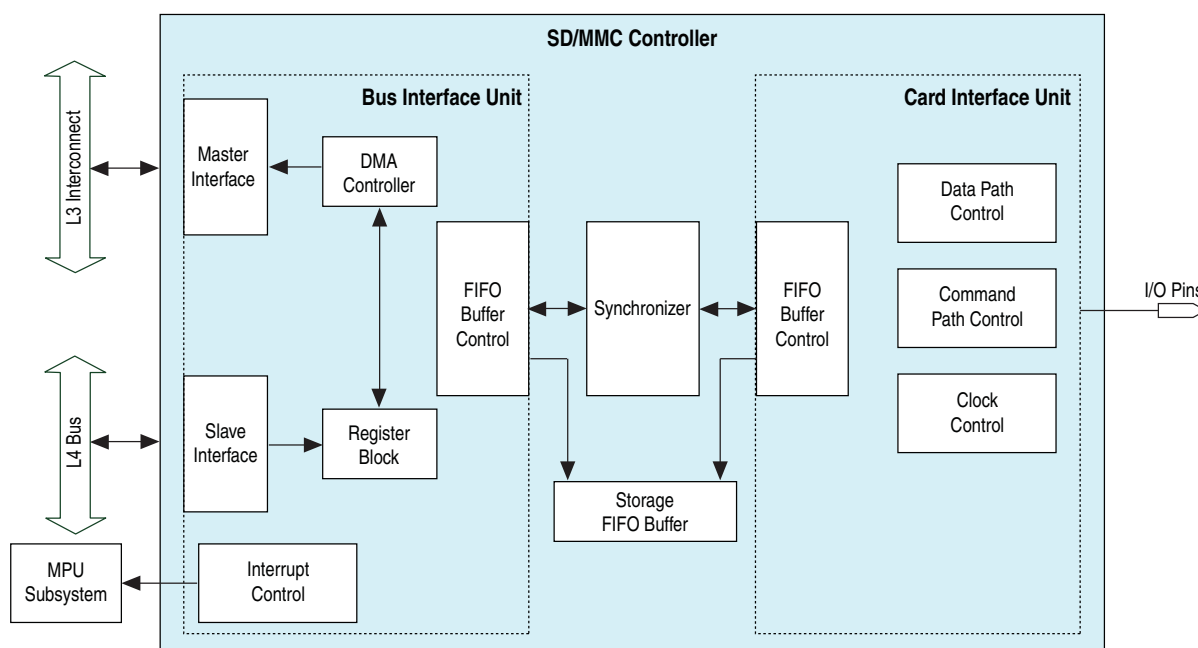
- (1) SPI 模式在 MMC 4.41 规范中不再使用。
- (2) 所有的 FPGA 封装都不支持可选的 8-bit 总线模式。
- (3) 支持 50 MHz 的最大时钟率，而不是 52 MHz（在 MMC 规范中指定的）。

SD/MMC 控制器结构图和系统集成

SD/MMC 控制器包括一个总线接口单元 (BIU) 和一个卡接口单元 (CIU)。BIU 提供一个从接口，用于主机存取控制和状态寄存器 (CSR)。此外，该单元通过一个 DMA 接口提供独立的 FIFO 缓存存取。DMA 控制器用于交换系统存储器与 FIFO 缓存之间的数据。DMA 寄存器可被主机访问来控制 DMA 操作。CIU 在控制器上支持 SD, MMC 和 CE-ATA 协议，并通过时钟控制模块提供时钟管理。中断控制模块（用于生成中断）连接到 ARM® Cortex™ A9 微处理器单元 (MPU) 子系统内的通用中断控制器。

图 11-1 显示了 SD/MMC 控制器的结构图以及如何集成在 HPS 中。

图 11-1. SD/MMC 控制器连接



SD/MMC 控制器的功能描述

本小节介绍了 SD/MMC 控制器组件及控制器如何运行。

SD/MMC/CE-ATA 协议

SD/MMC/CE-ATA 协议基于命令和数据比特流，该比特流由起始比特启动，并由停止比特终止。此外，SD/MMC 控制器也提供参考时钟，SD/MMC 控制器也是唯一能启动传输的主接口。

- Command— 在 CMD 管脚上串行发送的一个标记 (token)，开始一个操作。
- Response— 作为对某些命令的响应，从卡在 CMD 管脚串行发送一个标记 (token)。
- Data— 对数据移动命令使用数据管脚串行传递。

图 11 - 2 显示了一个多数据块 (multiple-block) 读操作的实例，其中的时钟只是一个表示，并不表示确切的时钟周期数。

图 11 - 2. 多数据块读操作

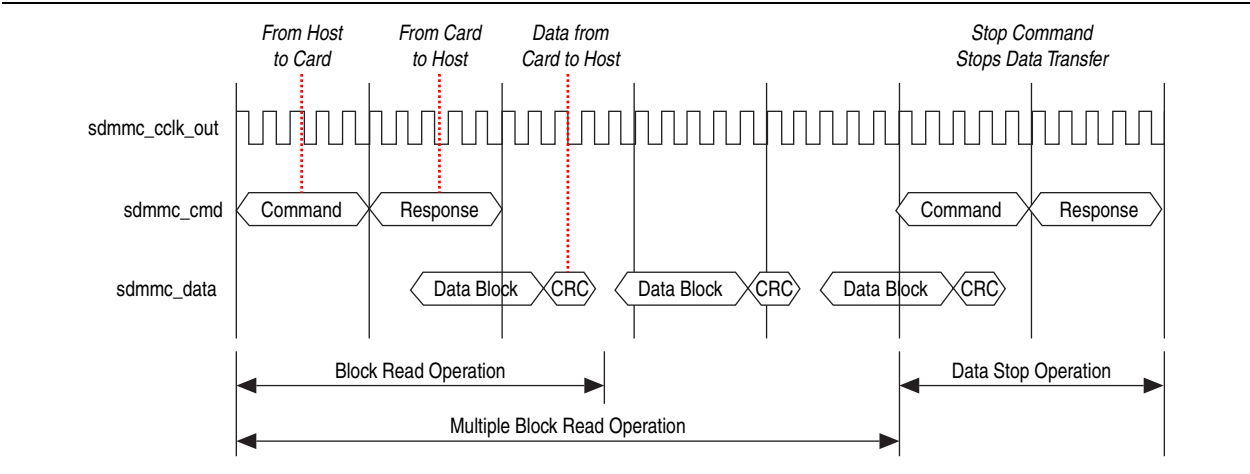
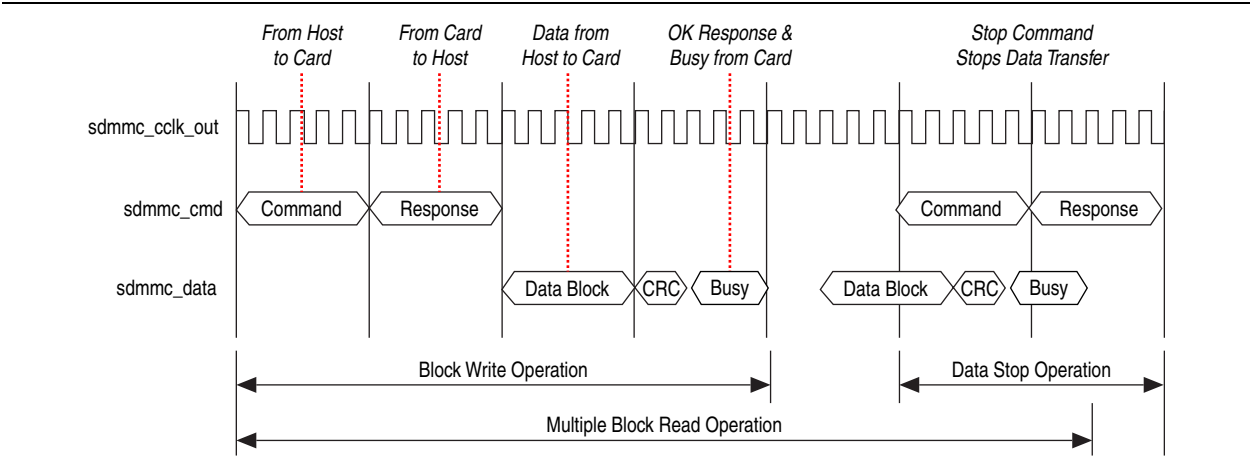


图 11 - 3 显示了多数据块写操作中由主机发送的一个命令标记的实例。

图 11 - 3. 多数据块写操作



BIU

BIU 接口连接到 CIU，并连接到 level 3 (L3) 互联和 level 4 (L4) 外围总线。BIU 包含下面的主要功能模式：

- 从接口
- 寄存器模块
- FIFO 缓存
- 内部 DMA 控制器

从接口

主处理器通过从接口访问 SD/MMC 控制器寄存器和数据 FIFO 缓存。

寄存器模块

寄存器模块是 BIU 的一部分，提供对 CSR 的读写访问。

所有寄存器都位于 BIU 时钟域中。当发送命令到卡时，通过将命令寄存器 (cmd) 的起始命令比特 (start_cmd) 设为 1，CIU 操作所需要的全部相关寄存器都被传递到 CIU 模块。在此期间，软件一定不要对这些从 BIU 传递到 CIU 的寄存器进行写操作。软件再次写入这些寄存器之前，必须等待硬件将 start_cmd 比特复位成 0。寄存器单元有一个硬件锁定功能 (hardware locking feature) 来防止对寄存器的非法写入。

通过设置 cmd 寄存器的 start_cmd 比特来发出一个 start 命令后，在 CIU 接收到此命令前不能写入以下寄存器：

- 命令 (cmd)
- 命令自变量 (cmdarg)
- 字节数 (bytcnt)
- 数据块大小 (blksiz)
- 时钟分频器 (clkdiv)
- 时钟使能 (clkena)
- 时钟源 (clksrc)
- 超时 (tmout)
- 卡类型 (ctype)

一旦 CIU 接收命令，硬件就会复位 start_cmd 比特。如果在此锁定期间试图对这些寄存器进行主机写操作，那么该写操作会被忽略，原始中断状态寄存器 (rintsts) 中的硬件锁定写错误比特 (hle) 被设为 1。此外，如果中断使能，并且未被硬件锁定错误屏蔽，那么一个中断会发送到主机。

接收到一个命令后，在下面情况中可以发送另一个命令到 CIU (有一个 one-deep 命令队列)：

- 如果前一个命令不是数据传递命令，那么一旦前一个命令完成就发送新的命令到 SD/MMC/CE-ATA 卡。
- 如果前一个命令是数据传递命令，并且对于新命令 cmd 寄存器的等待前一个数据完成比特 (wait_prvdata_complete) 设为 1，那么仅当数据传递完成时才会发送这一新命令到 SD/MMC/CE-ATA 卡。
- 如果 wait_prvdata_complete 比特设为 0，那么前一个命令一发送，新的命令就会发送到 SD/MMC/CE-ATA 卡。通常利用这一特性来停止或终止前一个数据传递或者查询数据传递中的卡状态。

中断控制器单元 (Interrupt Controller Unit)

中断控制器单元生成中断，此中断取决于 `rintsts` 寄存器，中断屏蔽寄存器 (`intmask`) 和控制寄存器 (`ctrl`) 的中断使能比特 (`int_enable`)。一旦检测到中断情况，控制器就会设置 `rintsts` 寄存器中的相应中断比特。`rintsts` 寄存器中的该比特保持为 1，直到软件通过对中断比特写入一个 1 来将该比特复位成 0；写入 0 对该比特没有影响。

中断端口是一个高电平有效 (active-high)，电平敏感的中断。仅当 `rintsts` 寄存器中至少一个比特设为 1，相应的 `intmask` 寄存器比特是 1 以及 `ctrl` 寄存器的 `int_enable` 比特是 1 时，中断端口才是有效的。

下面的比特可用作顶层端口用于调试的目的：

- `intmask` 寄存器中的全部比特
- `rintsts` 寄存器中的全部比特
- `ctrl` 寄存器中的 `int_enable` 比特

`ctrl` 寄存器的 `int_enable` 比特上电时设为 0，`intmask` 寄存器比特设为 0x0000000，屏蔽全部中断。

以下情况能导致中断的出现：

- 读操作中的结束比特错误
- 写操作上没有循环冗余代码 (CRC)
- 自动命令完成
- 起始比特错误
- 硬件锁定写错误
- FIFO 存储器下溢或上溢错误
- 主机超时导致的数据缺乏
- 数据读超时或者引导数据开始
- 响应超时或接收到引导 ACK
- 数据 CRC 错误
- 响应 CRC 错误
- 接收 FIFO 缓存数据请求
- 发送 FIFO 缓存数据请求
- 数据传递结束
- 命令完成
- 响应错误

Receive FIFO Data Request 和 Transmit FIFO Data Request 中断由电平敏感的中断源控制。因此，中断源首先必须清零，然后才能将 `rintsts` 寄存器中的中断相应比特设为 0。

例如，当接收 Receive FIFO Data Request 中断时，必须清空 FIFO 缓存，这样 FIFO 缓存数才不会大于 RX 水印（导致触发中断）。

其余的中断由一个时钟脉冲宽度（clock-pulse-width）源触发。

FIFO 缓存

TSDD/MMC 控制器有一个 4-KB 数据 FIFO 缓存用于存储发送和接收数据。FIFO 缓存存储器支持纠错码 (ECC)。与 FIFO 缓存连接的两个接口都支持单比特和双比特错误注入 (error injection)。使能和错误注入管脚是由系统管理程序驱动的输出，状态管脚是驱动到 MPU 子系统的输出。

SD/MMC 控制器提供输出，当检测到（和纠正）单比特可纠正错误和双比特（不可纠正）错误时通知系统管理器。当检测到 ECC 错误时，系统管理器生成一个对 GIC 的中断。

 关于更多信息，请参考 *Cyclone® V 器件手册* 卷 3 中的 *System Manager* 章节。

内部 DMA 控制器

内部 DMA 控制器有一个 CSR 和一个发送引擎或接收引擎，用于在系统存储器与卡之间传递数据。该控制器使用一种描述符机制，在最小的主处理器干预下能够有效地从源到目的移动数据。通过设置控制器可以在向卡上发送和接收数据传递完成的情况下以及其它正常或错误情况下中断主处理器。DMA 控制器和主驱动器通过单数据结构进行通信。

内部 DMA 控制器传递来自卡的数据到系统存储器中的数据缓存中，并传递发送数据，从存储器中的数据缓存到控制器的 FIFO 缓存。位于系统存储器中的描述符用作这些缓存的指针。

数据缓存位于系统存储器的物理存储空间，包括完整和部分的的数据。缓存状态保持在描述符中。数据链 (data chaining) 是指跨越多个数据缓存的数据。然而，单一描述符不能跨越多个数据缓存。

单一描述符用于接收和发送。列表的基地址被写入描述符列表基地址寄存器 (`dbaddr`)。描述符列表是正向连接的。最后一个描述符能指回到第一个入口，以创建一个环状结构。描述符列表位于主机的物理存储器地址空间。每个描述符可以最多指向两个数据缓存。

内部 DMA 控制器描述符

内部 DMA 控制器使用以下类型的描述符结构：

- 双缓存结构 — 两个描述符的间距由写入到总线模式寄存器 (bmod) 的描述符跳过长度字段 (dsl) 的跳过长度值决定的。
- 链式结构 — 每个描述符指向一个独立的缓存和链接列表中的下一个描述符。

图 11 - 4 和图 11 - 5 分别显示了内部 DMA 控制器双缓存描述符结构和链式描述符结构。

图 11 - 4. 双缓存描述符结构

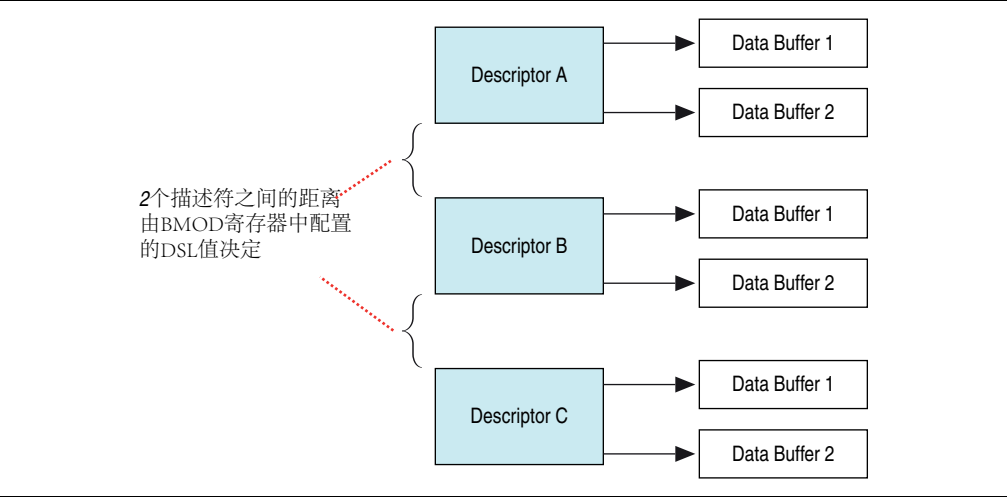


图 11 - 5. 链式描述符结构

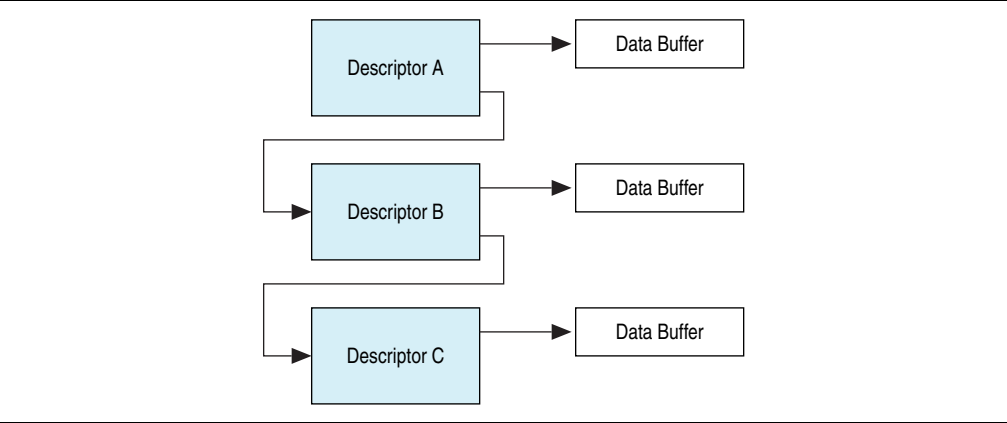


表 11 - 3 显示了一个描述符的内部格式。描述符地址必须与 32-bit 总线对齐。每个描述符包括 16 字节的控制和状态信息。关于描述符每个比特的详细信息，请参考第 11 - 9 页的表 11 - 4 到第 11 - 10 页的表 11 - 7。

表 11 - 3. 描述符格式 (1/2)

名称	偏移	31	30	29	...	26	25	...	13	12	...	6	5	4	3	2	1	0
DES0	0	OWN	CES	—										ER	CH	FS	LD	DIC
DES1	4	—				BS2						BS1						

表 11 - 3. 描述符格式 (2/2)

名称	偏移	31	30	29	...	26	25	...	13	12	...	6	5	4	3	2	1	0
DES2	8	BAP1																
DES3	12	BAP2 或者下一个描述符地址																

内部 DMA 控制器描述符中的 DES0 字段包含控制和状态信息。表 11 - 4 列出了此描述符中的比特信息。

表 11 - 4. 内部 DMA 控制器 DES0 描述符字段

比特	名称	说明
31	OWN	设为 1 时，该比特表明描述符被内部 DMA 控制器拥有。 设为 0 时，该比特表明描述符被主机拥有。内部 DMA 控制器完成数据传递时置位该比特为 0。
30	Card Error Summary (CES)	CES 比特表明是否出现传输错误。CES 比特是 <code>rintsts</code> 寄存器中下面错误比特的逻辑或。 ■ End-bit error (<code>ebe</code>) (结束比特错误) ■ Response timeout (<code>rto</code>) (响应超时) ■ Response CRC (<code>rcrc</code>) (响应 CRC) ■ Start-bit error (<code>sbe</code>) (起始比特错误) ■ Data read timeout (<code>drtio</code>) (数据读超时) ■ Data CRC for receive (<code>dcrc</code>) (接收中的数据 CRC) ■ Response error (<code>re</code>) (响应错误)
29:6	保留	—
5	End of Ring (ER)	设为 1 时，该比特表明描述符列表达到其最终描述符。内部 DMA 控制器返回到列表的基地址，创建描述符环。ER 对于一个双缓存描述符结构是有意义的。
4	Second Address Chained (CH)	设为 1 时，该比特表明描述符中的第二个地址是下一个描述符地址，而不是第二个缓存地址。该比特设为 1 时，BS2 (DES1[25:13]) 必须全为零。
3	First Descriptor (FS)	设为 1 时，该比特表明此描述符包含数据的第一个缓存。如果第一个缓存的大小为 0，那么下一个描述符包含数据的起始部分。
2	Last Descriptor (LD)	设为 1 时，该比特表明此描述符指向的缓存是数据的最后缓存。
1	Disable Interrupt on Completion (DIC)	设为 1 时，对于结束于此描述符指向的缓存的数据，该比特防止内部 DMA 控制器状态寄存器 (<code>idsts</code>) 的 TI/RI 比特的设置。
0	保留	—

DES1 描述符域包含缓存大小。表 11-5 列出了该描述符中的比特信息。

表 11-5. 内部 DMA 控制器 DES1 描述符域

比特	名称	说明
31:26	保留	—
25:13	Buffer 2 Size (BS2)	这些比特表明第二个数据缓存字节大小。该缓存大小必须是四的倍数，否则会导致未定义的行为。若 DES0[4] 设为 1，则该字段无效。
12:0	Buffer 1 Size (BS1)	表明数据缓存大小，该缓存大小必须是四字节的倍数，否则会导致未定义的行为。如果该字段为 0，DMA 会忽略此缓存，继续链式结构中的下一个描述符。 如果只有一个描述符和一个缓存被编程，那么只使用缓存 1，不使用缓存 2。

DES2 描述符域包含数据缓存的地址指针。表 11-6 列出了此描述符中的比特信息。

表 11-6. 内部 DMA 控制器 DES2 描述符域

比特	名称	说明
31:0	Buffer Address Pointer 1 (BAP1)	这些比特表明第一个数据缓存的物理地址。内部 DMA 控制器忽略 DES2 [1:0]，因为它只执行 32 比特对齐 (32-bit-aligned) 的访问。

如果当前描述符不是链式描述符结构中的最后一个描述符，也不是双缓存结构中的第二个缓存地址，那么 DES3 描述符域包含下一个描述符的地址指针。表 11-7 列出了该描述符中的比特。

表 11-7. 内部 DMA 控制器 DES3 描述符域

比特	名称	说明
31:0	Buffer Address Pointer 2 (BAP2) or Next Descriptor Address	这些比特表明使用双缓存结构时的第二个缓存的物理地址。如果 Second Address Chained (DES0[4]) 比特设为 1，那么此地址包含物理存储器的指针，在此物理存储器中出现下一个描述符。 如果这不是最后的描述符，那么下一个描述符地址指针必须与 32 比特对齐。忽略 Bit 1 和 0。

主机总线突发访问 (Host Bus Burst Access)

如果使用 bmod 寄存器的固定突发比特 (fb) 配置，那么内部 DMA 控制器试图在主接口执行固定长度的突发传输。最大突发长度由 bmod 寄存器的可编程突发长度 (pbl) 指示和限制。当获取描述符时，主接口总是呈现以 4 为突发长度的互联。

仅当 FIFO 缓存中有足够的空间来存储配置的突发，或者到传递末端的字节数少于配置的突发长度时，内部 DMA 控制器才启动数据传递。当 DMA 主接口配置为固定长度突发时，DMA 主接口使用最有效的 INCR4/8/16 和 SINGLE 传输组合来传递数据。如果 DMA 主接口没有配置为固定长度突发，那么 DMA 主接口就使用 INCR（未定义长度）和 SINGLE 传输来传递数据。

主机数据缓存对齐 (Host Data Buffer Alignment)

系统存储器中的发送和接收数据缓存必须对齐于 32 比特边界。

缓存大小计算

驱动器知道发送和接收的数据量。发送到卡时，内部 DMA 控制器传递 FIFO 缓存的确切字节数，该字节数由 DES1 描述符字段的缓存大小字段指定。

如果一个描述符没有标记成最后一个描述符 (DES0 字段的 LD 比特设为 0)，那么描述符的相应缓存应该是满的，并且缓存中的有效数据量由其缓存大小域准确地指明。如果一个描述符标记成最后一个描述符，那么缓存可能是满的，也可能不是满的，由 DES1 域中的缓存大小表明。驱动器知道有效位置的数量。驱动器应该会忽略剩下的无效字节。

内部 DMA 控制器中断

中断能由多种事件导致。idsts 寄存器包含可能导致中断的全部比特。内部 DMA 控制器中断使能寄存器 (idinten) 包含一个使能比特用于能够导致出现中断的每个事件。

idsts 寄存器中有两个汇总中断 (summary interrupt) — 正常中断汇总比特 (nis) 和异常中断汇总比特 (ais) 寄存器。nis 比特是 idsts 寄存器中的发送中断 (ti) 与接收中断 (ri) 比特逻辑或的结果。ais 比特是 idsts 寄存器中的致命总线错误中断 (fbe)，描述符不可用中断 (du) 和卡错误总结中断 (ces) 比特的逻辑或结果。

通过对相应的比特位置写入 1 来清除中断。如果对一个中断的比特位置写入 0，那么写操作被忽略，并且不会清除中断。当清除组中全部使能中断时，相应的汇总比特 (summary bit) 设为 0。当两个汇总比特都设为 0 时，中断信号被置低。

中断不排队。如果在驱动器已经响应前一个中断之前出现另一个中断事件，那么不会生成其它的中断。例如，idsts 寄存器的 ri 比特表明一个或多个数据被传递到主缓存。

对于同步的多个事件，只生成一个中断。驱动器必须扫描 idsts 寄存器以确定中断原因。控制器的最终中断信号是 BIU 与内部 DMA 控制器中断的逻辑或。

内部 DMA 控制器 FSM

下面步骤显示了内部 DMA 控制器功能状态机 (FSM) 操作：

1. 内部 DMA 控制器执行四次访问来获取一个描述符。
2. DMA 控制器从内部存储描述符信息。如果它是第一个描述符，那么控制器会发出一个 FIFO 缓存复位，并等待直到复位完成。
3. 内部 DMA 控制器检查描述符的每个比特是否正确。如果发现比特失配，那么相应的错误比特被设为 1，并通过设置 DES0 域中的 OWN 比特为 1 来关闭描述符。

rintsts 寄存器表明下面其中一个情况：

- 响应超时
- 响应 CRC 错误
- 数据接收超时
- 响应错误

4. 在写入数据到系统存储器之前，DMA 要等待到达 RX 水印，或者从系统存储器中读取数据之前，DMA 要等待到达 TX 水印。RX 水印代表 DMA 写入存储器前存储在本地 FIFO 缓存中的字节数。TX 水印代表 DMA 读取存储器中的数据前存储在本地 FIFO 缓存中的空闲字节数。
5. 如果可编程突发长度 (PBL) 字段的值大于缓存中剩余数据量，则启动单一传递 (single transfer)。如果使用双缓存，并且第二个缓存中没有数据 (缓存大小 = 0)，那么将跳过缓存并关闭描述符。
6. 在一个描述符的数据传递完成后，内部 DMA 控制器将描述符中的 OWN 比特设为 0。如果传递覆盖多个描述符，那么 DMA 控制器将获取下一个描述符。如果传递结束于当前描述符，那么设置 `idsts` 寄存器的 `ri` 比特或者 `ti` 比特后内部 DMA 控制器进入空闲模式。根据描述符结构 (双缓存或者链式) 加载描述符的相应起始地址。如果是双缓存描述符的第二个数据缓存，那么不会再次获取描述符。

内部 DMA 传递期间的终止

数据传递期间，如果主机发出一个 SD/SDIO STOP_TRANSMISSION 命令 (CMD12) 到卡中，那么内部 DMA 控制器在完成数据传递后将关闭当前的描述符直到 Data Transfer Over (DTO) 中断被置位。一旦发出 STOP_TRANSMISSION 命令，DMA 控制器就会执行单一突发传递。

1. 对于卡写操作，内部 DMA 控制器在从系统存储器中读取数据后不断将该数据写入 FIFO 缓存，直到 DTO 中断被置位。这样，卡时钟一直保持运行，将 STOP_TRANSMISSION 命令可靠地发送到卡中。
2. 对于卡读操作，内部 DMA 控制器不断读取 FIFO 缓存中的数据并将这些数据写入到系统存储器中，直到生成 DTO 中断。这样做是必需的，因为直到所有的 FIFO 缓存数据被清空才会生成 DTO 中断。



对于卡写操作的终止，只有当前描述符 (期间发出一个 STOP_TRANSMISSION 命令) 被内部 DMA 控制器关闭，剩下的未读描述符没被内部 DMA 控制器关闭。



对于卡读操作终止，内部 DMA 控制器从 FIFO 缓存中读取数据并将这些数据写入到相应的描述符数据缓存中。剩下的未读描述符没被关闭。

FIFO 缓存上溢和下溢

在正常数据传递情况下不会出现 FIFO 缓存上溢和下溢。然而，如果存在编程错误，那么就能够导致 FIFO 缓存上溢或者下溢。例如，考虑以下情况。

发送：

- PBL=4
- TX watermark = 1

对于这些编程值，如果 FIFO 缓存只有一个位置是空的，那么 DMA 会试图从存储器中读取四个字，即使只有一个字的存储空间可用。这会导致 FIFO 缓存上溢中断。

接收：

- PBL=4
- RX watermark = 1

对于这些编程值，如果 FIFO 缓存只有一个位置被填充，那么 DMA 会试图写入四个字，即使只有一个字可用。这会导致 FIFO 缓存下溢中断。

驱动器必须确保要传递的字节数（在描述符中指示的）是 4 字节的倍数。例如，如果 `bytcnt` 寄存器 = 13，那么描述符中指示的字节数据必须取舍到 16，因为长度字段的值必须是 4 字节的倍数。

表 11-8 列出了内部 DMA 控制器数据传递操作中的合法 PBL 和 FIFO 缓存水印值。

表 11-8. PBL 和水印值

PBL（传递数）	TX/RX 水印值
1	大于或等于 1
4	大于或等于 4
8	大于或等于 8
16	大于或等于 16
32	大于或等于 32
64	大于或等于 64
128	大于或等于 128
256	大于或等于 256

致命总线错误情况

致命总线错误 (fatal bus error) 是由通过主接口的错误响应造成的。这是一个系统错误，因此软件驱动器一定不能在控制器上继续执行任何设置。要从此情况中恢复，需要执行下面其中一项任务：

- 通过复位管理器 (reset manager) 发出一个复位到控制器。
- 通过写入 `ctrl` 寄存器的控制器复位比特来发出一个程序控制器复位。

CIU

CIU 连接 BIU 和 SD/MMC 卡或器件。主处理器将命令参数写入到 SD/MMC 控制器的 BIU 控制寄存器中，这些参数然后被传递到 CIU 中。根据控制寄存器的值，CIU 根据 SD/MMC 协议生成 SD/MMC 命令和卡总线上的数据流量。控制寄存器值也决定了命令和数据流量是否导向 CE-ATA 卡，以及 SD/MMC 控制器是否相应地控制命令和数据通路。

下面部分描述了 CIU 操作限制：

- 命令发出后，CIU 接收其它命令只检查读操作状态，或者停止传递。
- 一次只能发出一个数据传递命令。
- 在开端式的卡 (open-ended card) 读操作期间，如果由于 FIFO 缓存为空导致了卡时钟的停止，那么软件必须首先将数据填充进 FIFO 缓存并开始运行卡时钟。然而它只能发出一个 SD/SDIO `STOP_TRANSMISSION` (CMD12) 命令到卡。
- 在 SDIO/COMBO 卡传递期间，如果卡功能暂时停止，软件想继续暂停的传递，那么它必须首先复位 FIFO 缓存并执行 `resume` 命令，如同一个新的数据传递命令。
- 在卡数据传递过程中，当发出 SD/SDIO 卡复位命令 (`GO_IDLE_STATE`, `GO_INACTIVE_STATE` 或者 `CMD52_reset`) 时，软件必须将 `cmd` 寄存器中的停止终止命令比特 (`stop_abort_cmd`) 置为 1，这样控制器就能够在发出卡复位命令后停止数据传递。
- 如果由于卡读操作期间 FIFO 缓存已满导致了卡时钟的停止，那么软件必须读取至少两个 FIFO 缓存位置来开始运行卡时钟。

- 如果 CE-ATA 卡器件中断使能 (ATA 控制寄存器中的 `nIEN` 比特设为 0), 那么当 `RW_BLK` 命令挂起时一定不要发送新的 `RW_BLK` 命令到同一卡器件 (本文档中使用的 `RW_BLK` 命令是由 CE-ATA 规范定义的 `RW_MULTIPLE_BLOCK` MMC 命令)。等待 Command Completion Signal (CCS) 过程中只能发送 Command Completion Signal Disable (CCSD) 命令。
- 对于相同的卡器件, 如果中断在 CE-ATA 卡中禁用 (ATA 控制寄存器中的 `nIEN` 比特设为 1), 则可以使用一个新命令读取状态信息。
- CE-ATA 卡器件不支持开式传递 (open-ended transfer)。
- CE-ATA 传递不支持 `send_auto_stop` 信号 (软件一定不要设置 `cmd` 寄存器中的 `send_auto_stop` 比特)。

CIU 包含以下主要功能模块:

- 命令路径
- 数据路径
- 时钟控制

命令路径 (Command Path)

命令路径执行以下功能:

- 加载卡命令参数
- 发送命令到卡总线
- 接收来自卡总线的响应
- 发送响应到 BIU
- 加载时钟参数
- 在命令管脚上驱动 P-bit

通过写入 BIU 寄存器和设置 `cmd` 寄存器中的 `start_cmd` 比特发送一个新命令到控制器。命令路径加载新命令 (command, command argument, timeout) 并发送一个确认接收信号 (acknowledgement) 到 BIU。

新命令加载后, 命令路径状态机发送一个命令到卡总线 — 包括内部生成的七项 (seven-term) CRC (CRC-7) — 和接收响应 (如果有)。状态机然后发送接收的响应和信号到 BIU (命令完成), 然后在加载新命令之前等待八个时钟周期。在 CE-ATA 数据负载传递 (`RW_MULTIPLE_BLOCK`) 命令中, 如果卡器件中断使能 (ATA 控制寄存器中的 `nIEN` 比特设为 0), 那么状态机接收到响应后执行下面操作:

- 不驱动 P-bit; 等待 CCS, 解码并返回到空闲状态, 然后驱动 P-bit。
- 如果主机要发送 CCSD 命令, 并且八个时钟周期在响应后终止, 那么它将发送命令管脚上的 CCSD 码型。

加载命令参数 (Load Command Parameter)

在以下情况下, 命令或响应被加载到命令路径中:

- 来自 BIU 的新命令—当 BIU 发送一个新命令到 CIU 时，cmd 寄存器中的 start_cmd 比特设为 1。
- 内部生成的 send_auto_stop—当数据路径结束时，加载 SD/SDIO STOP 命令请求。
- 基于相对卡地址 (RCA) 0x000 的中断请求 (IRQ) 响应—当命令路径等待来自 MMC 的 IRQ 响应并且由 BIU 发一个 “send irq response” 请求，ctrl 寄存器中的发送 IRQ 请求比特 (send_irq_response) 设为 1。

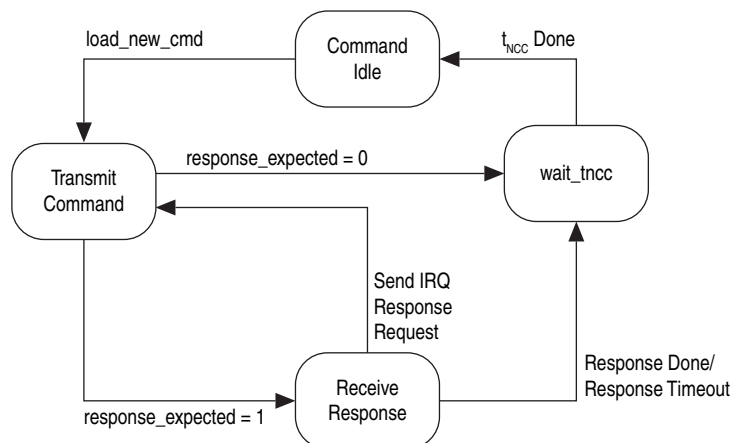
在命令路径中加载一个来自 BIU 的新命令取决于以下 cmd 寄存器比特设置：

- update_clock_registers_only—如果 cmd 寄存器中的该比特设为 1，那么命令路径只更新 clkena，clkdiv 和 clksrc 寄存器。如果该比特设为 0，那么命令路径将加载 cmd，cmdar 和 tmout 寄存器。然后，命令路径将处理发送到卡中的新命令。
- wait_prvdata_complete—如果该比特设为 1，命令路径在下面其中一个条件下加载新命令：
 - 如果数据路径是空闲的（也就是说没有数据传递发生），或者正在进行开端数据传递 (bytcnt = 0)，那么即刻加载新命令。
 - 如果正在进行预定义的数据传递，那么在当前数据传递完成后加载新命令。

发送命令和接收响应

一个新命令加载到命令路径中后 (cmd 寄存器中的 update_clock_registers_only 比特设为 0)，命令路径状态机在卡总线上发出一个命令。图 11-6 显示了命令路径状态机。

图 11-6. 命令路径状态机



命令路径状态机根据 cmd 寄存器比特值执行下面功能：

1. send_initialization—在发送命令之前发送 80 个时钟周期的初始化序列。
2. response_expected—命令所需要的一个响应。命令发出后，命令路径状态机接收一个 48-bit 或 136-bit 响应并将其发送到 BIU。如果在 (tmout 寄存器中设置的) 指定数量的周期内没有接收到卡响应的起始比特，那么 rintsts 寄存器中的 rto 比特和命令完成 (CD) 比特被设为 1，并信号给 BIU。如果预期响应比特设为 0，那么命令路径发出一个命令和发一个响应完成信号给 BIU，这会导致 rintsts 寄存器中的 cmd 比特设为 1。

- 3. response_length— 如果该比特设为 1，则会接收到一个 136-bit 的长响应；如果设为 0，则会接收到一个 48-bit 的短响应。
- 4. check_response_crc— 如果该比特设为 1，那么命令路径对在响应中接收到的 CRC-7 与内部生成的 CRC-7 作比较。如果两者不匹配，那么响应 CRC 错误会发送到 BIU，也就是说，rintsts 寄存器中的 rcrc 比特设为 1。

发送响应到 BIU

如果 cmd 寄存器中的 response_expected 比特设为 1，那么接收到的响应会被发送到 BIU 中。对于短响应，更新响应寄存器 0(resp0)。对于长响应，更新响应寄存器 3(resp3)，响应寄存器 2(resp2)，响应寄存器 (resp1) 和 resp0 寄存器，随后 rintsts 寄存器中的 cmd 比特被置为 1。如果响应用于 CIU 发出的一个 AUTO_STOP 命令，那么该响应被写入到 resp1 寄存器中，随后 rintsts 寄存器中的自动命令完成比特 (acd) 被置为 1。

一个正确的卡响应包含表 11 - 9 中所列出的字段。命令路径验证卡响应的内容。

表 11 - 9. 卡响应字段

字段	内容
Response transmission bit(响应传输比特)	0
Command index(命令索引)	已发送命令的命令索引
End bit (结束比特)	1

对于 136-bit 响应，或者当 cmd 寄存器中的 check_response_crc 比特设为 0 时，不检查命令索引。对于 136-bit 响应和保留的 CRC 48-bit 响应，命令索引被保留，也就是 0b111111。

关于响应值的更多信息，请参考第 11 - 67 页的“参考文献”中介绍的 *Physical Layer Simplified Specification, Version 3.01*。

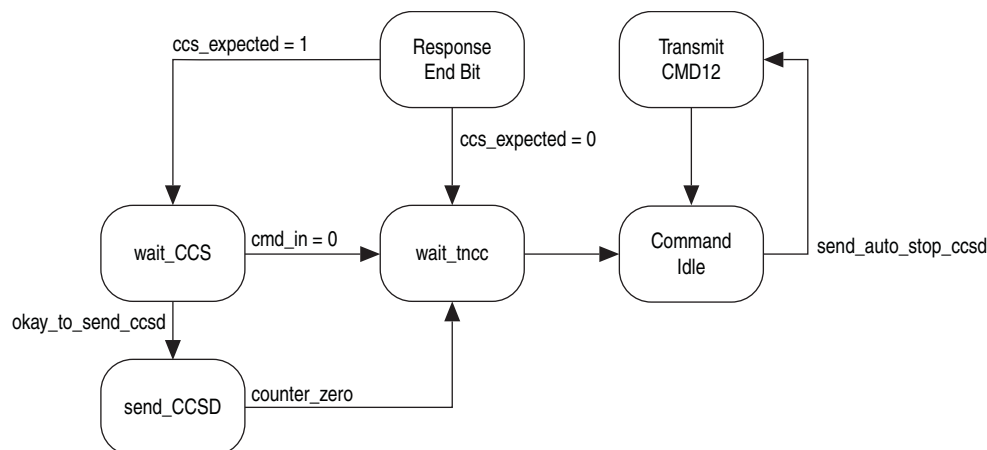
在 CMD 行上驱动 P-bit

命令路径在在两个命令之间的 CMD 行上驱动单周期上拉比特 (P-bit) 为 1(如果不需要响应)。如果需要响应，那么在接收到响应后和在下一个命令开始前驱动 P-bit。在访问 CE-ATA 卡器件期间，对于需要 CCS 的命令，仅在 CE-ATA 卡中的中断被禁用 (ATA 控制寄存器中的 nIEN 比特设为 1)，也就是 cmd 寄存器中的 CCS 期望比特 (ccs_expected) 设为 0 时，才会在响应后驱动 P-bit。如果命令需要 CCS，那么仅在接收到 CCS 后驱动 P-bit。

轮询 CCS

CE-ATA 卡器件生成 CCS 以通知主控制器正常的 ATA 命令完成或者 ATA 命令终止。接收到来自卡的响应后，命令路径状态机根据 cmd 寄存器比特值来执行图 11-7 中显示的功能。

图 11-7. CE-ATA 命令路径状态机



以下描述了图 11-7 中的一些细节：

1. 响应结束比特状态 (response end bit state) — 状态机接收来自卡器件的响应结束比特。如果 cmd 寄存器的 ccs_expected 比特设为 1，那么状态机将进入 CCS 状态。
2. 等待 CCS — 状态机等待来自 CE-ATA 卡器件的 CCS。等待期间能够发生下面情况：
 - a. 软件设置 ctrl 寄存器中的发送 CCSD 比特 (send_ccsd)，指示不等待 CCS 和在命令行上发送 CCSD 码型。
 - b. 在 CMD 行上接收 CCS。
3. 发送 CCSD 命令 — 在 CMD 行上发送 CCSD 码型 (0b00001)。

主处理器的 CCS 检测和中断

如果 cmd 寄存器中的 ccs_expected 比特设为 1，那么通过设置 rintsts 寄存器中的数据传递结束比特 (dto) 来指示来自 CE-ATA 卡器件的 CCS。控制器生成一个 DTO 中断（如果该中断没被屏蔽）。

对于 RW_MULTIPLE_BLOCK 命令，如果 CE-ATA 卡器件中断被禁用（ATA 控制寄存器中的 nIEN 比特设为 1）——也就是 cmd 寄存器中的 ccs_expected 比特设为 0——那么没有来自卡的 CCS。当数据传递结束——也就是当所要求数量的字节传递完成时——rintsts 寄存器中的 dto 比特设为 1。

CCS 超时

如果命令需要来自卡器件的 CCS (cmd 寄存器中的 ccs_expected 比特设为 1)，那么命令状态机将等待 CCS 并保持等待 CCS 状态。如果 CE-ATA 卡无法发送 CCS，那么主软件 (host software) 必须实现一个超时机制来释放命令和数据路径。控制器不实现硬件计时器；主硬件负责维护软件计时器。


如果发生 CCS 超时，那么主机必须通过设置 `ctrl` 寄存器中的 `send_ccsd` 比特来发出一个 CCSD 命令。控制器命令路径状态机发送 CCSD 命令到 CE-ATA 卡器件并进入空闲状态。发送 CCSD 命令后，主机也必须发送一个 SD/SDIO STOP_TRANSMISSION 命令到 CE-ATA 卡，以终止未处理的 ATA 命令。

发送 CCSD 命令

如果 `ctrl` 寄存器中的 `send_ccsd` 比特设为 1，那么控制器在 CMD 线上发送一个 CCSD 码型。在等待 CCS 过程中，或者发生 CCS 超时后，主机能够发送 CCSD 命令。

发送 CCSD 码型后，控制器设置 `rintsts` 寄存器中的 `cmd` 比特，并生成对主机的中断（如果 Command Done 中断没被屏蔽）。


 在 CIU 模块中，如果在与采集的 CCS 相同的时钟周期上 `ctrl` 寄存器中的 `send_ccsd` 比特设为 1，那么 CIU 模块不在 CMD 线上发送 CCSD 码型。在此情况下，`rintsts` 寄存器中的 `dto` 和 `cmd` 比特设为 1。

 由于异步边界，可能已经发生 CCS，并且 `send_ccsd` 比特设为 1。在此情况下，CCSD 命令没有发送到 CE-ATA 卡器件，并且 `send_ccsd` 比特没有设为 0。在下一个命令发出前，主机必须将 `send_ccsd` 比特复位成 0。

如果 `ctrl` 寄存器中的发送自动停止 CCSD (`send_auto_stop_ccsd`) 比特设为 1，那么在发送 CCSD 码型后控制器将发送一个内部生成的 STOP_TRANSMISSION 命令 (CMD12)。控制器设置 `rintsts` 寄存器中的 `acd` 比特。

I/O 传输延迟 (N_{ACIO} 超时)

主机软件维持超时机制以处理读取 CE-ATA 卡器件期间的 I/O 传输延迟 (N_{ACIO} 周期) 超时。控制器既不维持超时机制也不在等待数据标记 (data token) 的起始比特期间指示 N_{ACIO} 周期已经过去。I/O 传输延迟适用于使用 RW_REG 和 RW_BLK 命令的读传递；本文档中使用的 RW_REG 和 RW_BLK 命令指的是 CE-ATA 规范中定义的 RW_MULTIPLE_REGISTER 和 RW_MULTIPLE_BLOCK MMC 命令。

 N_{ACIO} 超时后，应用程序必须通过发送 CCSD 和 STOP 命令或者 STOP 命令来终止命令。当 STOP_TRANSMISSION 命令从控制器发出时，Data Read Timeout (DRT0) 中断可能被置为 1，在此情况下，`rintsts` 寄存器中的数据读超时引导数据起始比特 (`bds`) 和 `dto` 比特被置为 1。

数据路径

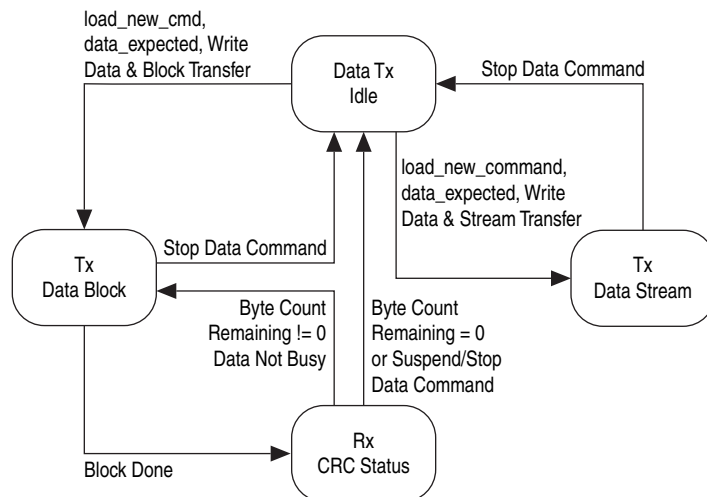
数据路径模块读取数据 FIFO 缓存，并在写数据传递期间发送卡总线上的数据，或者在读数据传递期间接收数据并将其写入到 FIFO 缓存中。每当未执行数据传递命令时，数据路径就加载新的数据参数 — 需要的数据，读 / 写数据传递，流 / 块传递，块大小，字节数，卡类型，超时寄存器。如果 `cmd` 寄存器中的数据传输预期比特 (`data_expected`) 设为 1，那么新的命令是一个数据传递命令，数据路径开始下面其中一个操作：

- 若 read/write bit = 1，则发送数据
- 若 read/write bit = 0，则接收数据

数据发送

图 11-8 所示的数据发送状态机在接收到数据写命令的响应后的两个周期开始数据传输，即便命令路径检测到一个响应错误或者响应 CRC 错误。如果由于响应超时没有接收到来自卡的响应，那么数据没有被发送。根据 cmd 寄存器中的的传递模式比特 (transfer_mode) 的值，数据发送状态机将数据流或数据块放到卡数据总线上。

图 11-8. 数据发送状态机



流数据发送 (Stream Data Transmit)

如果 cmd 寄存器中的 transfer_mode 比特设为 1，那么传递是一个流写入 (stream-write) 数据传递。数据路径读取 BIU 中 FIFO 缓存的数据，并将这些数据以数据流的方式发送到卡数据总线。如果 FIFO 缓存变空，那么卡时钟停止，一旦 FIFO 缓存中有数据卡时钟又重新开始。

如果 bytcnt 寄存器复位成 0，那么传递是一个开端流写入 (open-ended stream-write) 数据传递。在此数据传递期间，数据路径继续以流的方式发送数据直到主机软件发出一个 SD/SDIO STOP 命令。当 STOP 命令的结束比特和数据的结束比特匹配超过两个时钟周期时，流数据传递终止。

如果写入 bytcnt 寄存器一个非零值，并且 cmd 寄存器中的 send_auto_stop 比特设为 1，那么就会从内部生成 STOP 命令，并在流写传递 (stream write transfer) 的最后一个字节匹配后出现 STOP 命令的结束比特时在命令路径中加载该 STOP 命令。如果主机在所有数据字节传递到卡总线之前发出一个 STOP 命令，那么数据传输也会终止。

单数据块数据 (Single Block Data)

如果 cmd 寄存器中的 transfer_mode 比特设为 0，并且 bytcnt 寄存器的值等于 block_size 寄存器的值，那么会出现单数据块写数据传递 (single-block write-data transfer)。数据发送状态机以单数据块的形式发送数据，其中的字节数等于数据块大小，包括内部生成的 16 项 CRC (CRC-16)。

如果 `ctype` 寄存器被设置用于一个 1-bit, 4-bit 或者 8-bit 数据传递, 那么数据分别在 1, 4 或者 8 数据行发送, 单独生成 CRC-16, 并分别在 1, 4 或者 8 数据行发送。

一个单数据块发送后, 数据发送状态机接收来自卡的 CRC 状态, 并发送数据传递信号到 BIU 中。当 `rintsts` 寄存器中的 `dto` 比特设为 1 时这才会发生。

如果接收到一个来自卡的负 CRC 状态, 那么数据路径通过设置 `rintsts` 寄存器中的 `dcrc` 比特发送一个数据 CRC 错误到 BIU。

此外, 如果 CRC 状态的起始比特没有在数据块结束后的两个时钟周期接收到, 那么通过设置 `rintsts` 寄存器中的 `sbe` 比特, CRC 状态起始比特错误 (SBE) 信号会被发送到 BIU。

多数据块数据

如果 `cmd` 寄存器中的 `transfer_mode` 比特设为 0, 并且 `bytcnt` 寄存器中的值不等于 `block_size` 寄存器的值, 那么会出现多数据块写数据传递。数据发送状态机以数据块的形式发送数据, 其中数据块中的字节数等于数据块大小, 包括内部生成的 CRC-16。

如果 `ctype` 寄存器设为 1-bit, 4-bit 或者 8-bit 数据传递, 那么数据分别在 1, 4 或 8 数据行上发送, CRC-16 被单独生成并分别在 1, 4 或 8 数据行上发送。

一个数据块发送后, 数据发送状态机接收来自卡的 CRC 状态。如果剩余字节数变成 0, 那么数据路径发送数据传递完成的信号到 BIU。这在 `rintsts` 寄存器中的 `dto` 比特设为 1 时出现。

如果剩余数据字节大于 0, 那么数据路径状态机开始发送另一个数据块。

如果接收到一个来自卡的负 CRC 状态, 那么数据路径通过设置 `rintsts` 寄存器中的 `dcrc` 比特发送一个数据 CRC 错误到 BIU, 并继续数据发送直达所有字节发送完成。

如果 CRC 状态的起始比特没有在数据块结束后的两个时钟周期接收到, 那么通过设置 `rintsts` 寄存器中的 `sbe` 比特, CRC 状态起始比特错误 (SBE) 信号会被发送到 BIU, 并终止数据传递。

如果 `cmd` 寄存器中的 `send_auto_stop` 比特设为 1, 那么在最后一个数据块的传递过程中从内部生成 SD/SDIO STOP 命令, 其中没有额外字节传递到卡中。STOP 命令的结束比特可能不完全匹配最后一个数据块中的 CRC 状态的结束比特。

对于 1 比特, 4 比特或者 8 比特, 如果数据块尺寸分别小于 4, 16 或 32, 那么数据发送状态机在所有数据传递后将终止数据传递, 与此同时, 内部生成的 STOP 命令加载到命令路径。

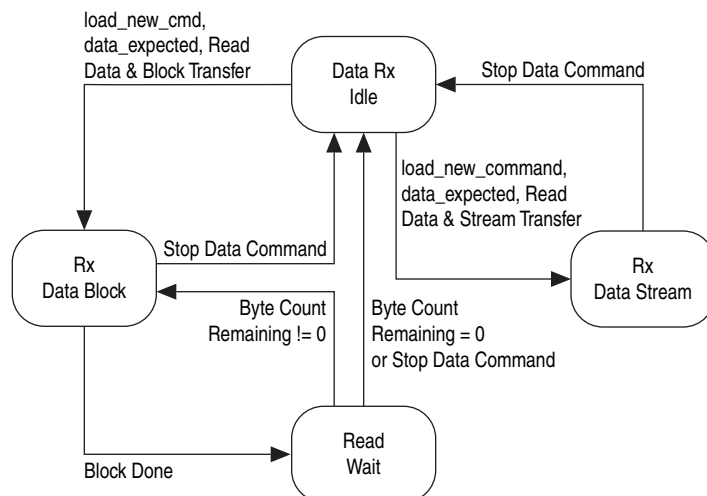
如果 `bytcnt` 是零 (数据块大小必须大于零), 那么传递是一个开放式传递。此类型的数据传递的数据发送状态机继续 block-write 数据传递, 直到主机软件发出一个 SD/SDIO STOP 或者 STOP_TRANSMISSION (CMD12) 命令。

数据接收

图 11-9 中所示的数据接收状态机在数据读命令的结束比特后的两个时钟周期接收数据, 即使命令路径检测到一个响应错误或者响应 CRC 错误。如果由于出现响应超时而没有接收到来自卡的响应, 那么 BIU 不会接收到数据传递完成的信号。这发生在由控制器发送的命令是一个卡非法操作时, 使卡不能开始读数据传递。

如果数据超时前没有接收到数据，那么数据路径会发送一个数据超时信号到 BIU 和完成数据传输的结束信号。根据 cmd 寄存器中的 transfer_mode 比特，数据接收状态机会以数据流或者数据块的形式接收来自卡的数据。

图 11 - 9. 数据接收状态机



流数据读操作 (Stream Data Read)

如果 cmd 寄存器中的 transfer_mode 比特设为 1，那么会出现 stream-read 数据传递，此时数据路径接收来自卡的数据并将其写入 FIFO 缓存中。如果 FIFO 缓存变满，那么一旦 FIFO 缓存不再是满的，卡时钟就会停止并重新开始。

如果 bytcnt 寄存器设为 0，那么开放式流读取数据方式开始传输数据。在此类型的数据传递过程中，数据路径持续接收数据流，直到主机发出一个 SD/SDIO STOP 命令。流数据传递在 STOP 命令的结束比特后的两个时钟周期终止。

如果 bytcnt 寄存器包含一个非零值，并且 cmd 寄存器中的 send_auto_stop 比特设为 1，那么 STOP 命令从内部生成并加载到数据路径中，其中在接收到流数据传递的最后一个字节后出现 STOP 命令的结束比特。如果在接收到卡中所有的数据字节之前主机发出一个 SD/SDIO STOP 或者 STOP_TRANSMISSION (CMD12) 命令，那么此数据传将终止。

单数据块数据读操作 (Single-block Data Read)

如果 ctype 寄存器设为 1-bit, 4-bit 或 8-bit 数据传递，那么分别从 1, 4 或 8 数据行接收数据，独立生成 CRC-16，并分别检查 1, 4 或 8 数据行上的 CRC-16。如果有 CRC-16 失配，那么数据通路会发送一个数据 CRC 错误信号到 BIU。如果接收到的结束比特不是 1，那么 BIU 会接收到一个结束比特错误 (EBE)。

多数据块数据读操作 (Multiple-block Data Read)

如果 cmd 寄存器中的 transfer_mode 比特设为 0，并且 bytcnt 寄存器的值不等于 block_size 寄存器的值，那么这是一个多数据块读数据传递。数据接收状态机接收块数据，其中块中的字节数等于数据块大小，包括内部生成的 CRC-16。

如果 ctype 寄存器设为 1-bit, 4-bit 或者 8-bit 数据传递，那么分别从 1, 4 或 8 数据行接收数据，单独生成 CRC-16，并分别检查 1, 4 或 8 数据行上的 CRC-16。接收到数据块后，如果剩余字节数变成零，那么数据路径会发送一个数据传递信号到 BIU。

如果剩余数据字节大于零，那么数据路径状态机会导致另一个数据块被接收。如果一个接收到的数据块的 CRC-16 与内部生成的 CRC-16 不匹配，那么 BIU 的数据 CRC 错误和数据接收继续数据发送，直到发送全部数据。此外，如果接收到的数据块的结束比特不是 1，那么数据路径上的数据信号将终止 CIU 的比特错误，数据接收状态机终止数据接收，等待数据超时，并且发送数据传递完成信号到 BIU。

如果 cmd 寄存器中的 send_auto_stop 比特设为 1，那么当传递最后一个数据块时从内部生成 SD/SDIO STOP 命令，没有传递卡中额外的字节。STOP 命令的结束比特可能不会完全匹配最后一个数据块中的结束比特。

对于 1-bit，4-bit 或者 8-bit 数据传递模式，如果卡数据传递所要求的数据块大小分别小于 4，16 或 32 字节，那么传递所有数据后数据发送状态机将终止数据传递，此时内部生成的 STOP 命令加载到命令路径。之后接收到的卡数据被数据路径忽略。

如果 bytcnt 寄存器是 0（数据块大小必须大于零），那么这是一个开放式数据块传递。对于此类数据传递，数据接收状态机继续 block-read 数据传递，直到主机软件发出一个 SD/SDIO STOP 或者 STOP_TRANSMISSION (CMD12) 命令。

自动停止 (Auto Stop)

当寄存器中的 `send_auto_stop` 比特设为 1 时，控制器从内部生成 SD/SDIO STOP 命令并加载到命令路径中。通过对 MMC 使用数据流读 (stream read) 和对 SD 卡的 SD 存储器传递使用多数据块读或写 (multiple-block read or write)，AUTO_STOP 命令帮助发送确切数量的数据字节。软件必须根据表 11 - 10 中的明细设置 `send_auto_stop` 比特。

表 11 - 10. 自动停止生成

ird type	Transfer type	Byte Count	send_auto_stop bit set	Comments
VC	Stream read	0	No	Open-ended stream
VC	Stream read	> 0	Yes	Auto-stop after all bytes transfer
VC	Stream write	0	No	Open-ended stream
VC	Stream write	> 0	Yes	Auto-stop after all bytes transfer
VC	Single-block read	> 0	No	Byte count = 0 is illegal
VC	Single-block write	> 0	No	Byte count = 0 is illegal
VC	Multiple-block read	0	No	Open-ended multiple block
VC	Multiple-block read	> 0	Yes **	Pre-defined multiple block
VC	Multiple-block write	0	No	Open-ended multiple block
VC	Multiple-block write	> 0	Yes **	Pre-defined multiple block
MEM	Single-block read	> 0	No	Byte count = 0 is illegal
MEM	Single-block write	> 0	No	Byte count = 0 illegal
MEM	Multiple-block read	0	No	Open-ended multiple block
MEM	Multiple-block read	> 0	Yes	Auto-stop after all bytes transfer
MEM	Multiple-block write	0	No	Open-ended multiple block
MEM	Multiple-block write	> 0	Yes	Auto-stop after all bytes transfer
SDIO	Single-block read	> 0	No	Byte count = 0 is illegal
SDIO	Single-block write	> 0	No	Byte count = 0 illegal
SDIO	Multiple-block read	0	No	Open-ended multiple block
SDIO	Multiple-block read	> 0	No	Pre-defined multiple block
SDIO	Multiple-block write	0	No	Open-ended multiple block
SDIO	Multiple-block write	> 0	No	Per-defined multiple block

** The condition under which the transfer mode is set to block transfer and *byte_count* is equal to block size is treated as a single-block data transfer command for both MMC and SD cards. If *byte_count* = *n***block_size* (*n* = 2, 3, ...), the condition is treated as a predefined multiple-block data transfer command. In the case of an MMC card, the host software can perform a predefined data transfer in two ways: 1) Issue the CMD23 command before issuing CMD18/CMD25 commands to the card – in this case, issue CMD18/CMD25 commands without setting the `send_auto_stop` bit. 2) Issue CMD18/CMD25 commands without issuing CMD23 command to the card, with the `send_auto_stop` bit set. In this case, the multiple-block data transfer is terminated by an internally-generated auto-stop command after the programmed byte count.

下面列表描述了 AUTO_STOP 命令的条件：

- 字节数大于零的 MMC 的流读操作 — 控制器生成一个内部 STOP 命令并将其加载到命令路径中，这样当从卡中读取数据的最后一个字节和没有接收到额外的数据字节时，发送 STOP 命令的结束比特。如果字节数少于六（48 比特），那么在发送 STOP 命令的结束比特之前将从卡中接收一些额外的数据字节。
- 字节数大于零的 MMC 的流写操作 — 制器生成一个内部 STOP 命令并将其加载到命令路径中，这样当数据的最后一个字节在卡总线上发送以及无额外的字节被发送时，发送 STOP 命令的结束比特。如果字节数少于六（48 比特），那么数据路径将发送数据 (data last) 以满足这些条件。
- 字节数大于零的 SD 卡的多数据块读存储器 — 如果模块大小少于四 (single-bit 数据总线)，16 (4-bit 数据总线) 或者 32 (8-bit 数据总线)，那么读取全部字节后 AUTO_STOP 命令被加载到命令路径。否则，STOP 命令被加载到命令路径，这样在接收到最后一个数据块后会发送 STOP 命令的结束比特。
- 字节数大于零的 SD 卡的多数据块写存储器 — 如果模块大小少于三 (single-bit 数据总线)，12 (4-bit 数据总线) 或者 24 (8-bit 数据总线)，那么发送全部数据块后 AUTO_STOP 命令被加载到命令路径。否则，STOP 命令被加载到命令路径，这样在接收到 CRC 状态的结束比特后会发送 STOP 命令的结束比特。
- auto-stop 期间对主机软件的预先警告 — 当一个 AUTO_STOP 命令发出时，主机软件必须在控制器发送 AUTO_STOP 命令和数据传递完成后才能发出一个新的命令到控制器。如果主机在采用 AUTO_STOP 命令进行数据传输过程中需要处理一个新的命令时，在发送新的命令和接收到其响应后，可能会发送 AUTO_STOP 命令。这会导致延迟发送 STOP 命令，从而传递额外的数据字节。对于数据流写操作 (stream write)，额外的数据字节是错误数据，能够损坏卡数据。如果主机要在数据传递完成前终止数据传递，那么它能够发出一个 SD/SDIO STOP 或者 STOP_TRANSMISSION (CMD12) 命令，在此情况下，控制器不会生成 AUTO_STOP 命令。

使用数据路径的非数据传递命令

某些 SD/SDIO 非数据传递命令（读写命令以外的其它命令）也使用数据路径。表 11-11 列出了这些命令及它们的寄存器设置要求。

表 11 - 11. 非数据传递命令和要求

	PROGRAM_CSD (CMD27)	SEND_WRITE_PROT (CMD30)	LOCK_UNLOCK (CMD42)	SD_STATUS (ACMD13)	SEND_NUM_WR_BLOCKS (ACMD22)	SEND_SCR (ACMD51)
cmd 寄存器设置						
Cmd_index	0x1B=27	0x1E=30	0x2A=42	0x0D=13	0x16=2	0x33=51
Response_expect	1	1	1	1	1	1
Response_length	0	0	0	0	0	0
Check_response_crc	1	1	1	1	1	1
Data_expected	1	1	1	1	1	1
Read/write	1	0	1	0	0	0
Transfer_mode	0	0	0	0	0	0
Send_auto_stop	0	0	0	0	0	0
Wait_prevdata_complete	0	0	0	0	0	0
Stop_abort_cmd	0	0	0	0	0	0
cmdarg 寄存器设置						
	填充位	32-bit 写保护 数据地址	填充位	填充位	填充位	填充位
blksiz 寄存器设置						
	16	4	Num_bytes (1)	64	4	8
bytcnt 寄存器设置						
	16	4	Num_bytes (1)	64	4	8

表 11 - 11 注释：

(1) Num_bytes = 锁卡数据结构中指定的字节数量。请参考 SD 规范和 MMC 规范。

表 11 - 12.

	CMD27	CMD30	CMD42	ACMD13	ACMD22	ACMD51
Command register programming						
Cmd_index	6'h1B	6'h1E	6'h2A	6'h0D	6'h16	6'h33
Response_expect	1	1	1	1	1	1
Response_length	0	0	0	0	0	0
Check_response_crc	1	1	1	1	1	1
Data_expected	1	1	1	1	1	1
Read/write	1	0	1	0	0	0
Transfer_mode	0	0	0	0	0	0
Send_auto_stop	0	0	0	0	0	0
Wait_prevdata_complete	0	0	0	0	0	0
Stop_abort_cmd	0	0	0	0	0	0
Command Argument register programming						
	Stuff bits	32-bit write protect data address	Stuff bits	Stuff bits	Stuff bits	Stuff bits
Block Size register programming						
	16	4	Num_bytes*	64	4	8
Byte Count register programming						
	16	4	Num_bytes*	64	4	8
*: Num_bytes = No. of bytes specified as per the lock card data structure (Refer to the SD specification and the MMC specification).						

时钟控制模块

时钟控制模块提供了 SD/MMC/CE-ATA 卡所要求的不同时钟频率。时钟控制模块有一个时钟分频器，用于生成不同的卡时钟频率。

卡的时钟频率取决于下面的时钟 `ctrl` 寄存器设置：

- **clkdiv 寄存器** — 内部时钟分频器用于生成卡所需要的不同时钟频率。通过写入 clkdiv 寄存器能够设置时钟分频器的分频因子。时钟分频器是一个 8-bit 值，提供 1 到 510 的时钟分频因子；时钟分频因子 0 代表时钟分频器旁路，1 代表 2 分频，2 代表 4 分频，以此类推。
- **clksrc 寄存器** — 将此寄存器设为 0，因为时钟被 0 分频。
- **clkena 寄存器** — cclk_out 卡输出时钟能够在下面条件下使能或禁用：
 - clkena 寄存器中的 cclk_enable 比特设为 1 时，cclk_out 被使能，设为 0 时被禁用。
 - 通过设置 clkena 寄存器中的 cclk_low_power 比特设为 1 能够使能低功耗 (low-power) 模式。如果使能低功耗模式来节省卡电能，那么当卡处于空闲状态至少八个卡时钟周期时要禁用 cclk_out 信号。当加载一个新命令和命令路径进入非空闲状态时，使能低功耗模式。

在下面情况中，卡时钟停止或禁用：

- 通过写入 clkena 寄存器能够禁用时钟。
- 当选择了低功耗模式和卡处于空闲状态至少八个时钟周期时。
- FIFO 缓存是满的，数据路径不能接收更多的卡数据和数据传递未完成 — 避免 FIFO 缓存上溢。
- FIFO 缓存是空的，数据路径不能发送更多的数据到卡中和数据传递未完成 — 避免 FIFO 缓存下溢。



在主机软件修改 clkdiv 和 clksrc 寄存器的值之前，卡时钟必须通过 clkena 寄存器禁用。

错误检测

在下面情况中，CIU 中的卡操作过程中能够出现错误。

响应

- **响应超时** — 在超时寄存器中的指定数量的时钟周期内没有接收到带有响应起始比特的预期响应。
- **响应CRC错误** — 响应是预期的，需要检查响应CRC；响应CRC-7与内部生成的CRC-7不匹配。
- **响应错误** — 响应传输比特不是 0，命令索引与发送命令的命令索引不匹配，或者响应结束比特不是 1。

数据发送

- **无 CRC 状态** — 写数据传递过程中，如果在数据块的结束比特发出后的两个时钟周期内没有接收到 CRC 状态起始比特，那么数据路径执行下面操作：
 - 发送无 CRC 状态信号到 BIU
 - 终止进一步的数据传递
 - 发送数据传递完成信号到 BIU

- 负 CRC—如果在写数据块之后接收到的 CRC 是负的（也就是说不是 0b010），那么数据路径将发送一个数据 CRC 错误信号到 BIU，然后继续数据传递。
- 由 FIFO 缓存空导致的数据缺乏 (data starvation)—如果在写数据传输过程中 FIFO 缓存变空，或者卡时钟停止，FIFO 缓存保持一个数据超时数量时钟周期的空状态，那么数据路径将发送一个数据缺乏 (data-starvation) 错误信号到 BIU 并且继续等待 FIFO 缓存中的数据。

数据接收 (Data Receive)

- 数据超时 — 读数据传递过程中，如果在超时寄存器中指定数量的周期前没有接收到数据起始比特，那么数据路径执行下面操作：
 - 发送一个数据超时错误信号到 BIU
 - 终止进一步的数据传递
 - 发送数据传递完成信号到 BIU
- 数据 SBE— 在 4-bit 或者 8-bit 读数据传递过程中，如果全部比特 (all-bit) 数据行没有起始比特，那么数据路径将发送一个数据 SBE 信号到 BIU，并等待数据超时，之后发送一个数据传递完成信号。
- 数据 CRC 错误— 在读数据块传递过程中，如果接收到的 CRC-16 与内部生成的 CRC-16 不匹配，那么数据路径将发送一个数据 CRC 错误信号到 BIU，然后继续数据传递。
- 数据 EBE— 在读数据传递过程中，如果接收到的数据的结束比特不是 1，那么数据路径将发送一个 EBE 信号到 BIU，终止进一步数据传递，然后发送数据传递完成信号到 BIU。
- 由 FIFO 缓存满导致的数据缺乏 (data starvation)— 读数据传输过程中，当 FIFO 缓存变满时，卡时钟会停止。如果 FIFO 缓存保持一个数据超时数量时钟周期的满状态，那么通过设置 rintsts 寄存器中的数据缺乏主机超时比特 (hto) 为 1，数据路径将发送一个数据缺乏 (data-starvation) 错误信号到 BIU，然后继续等待 FIFO 缓存变空。

时钟

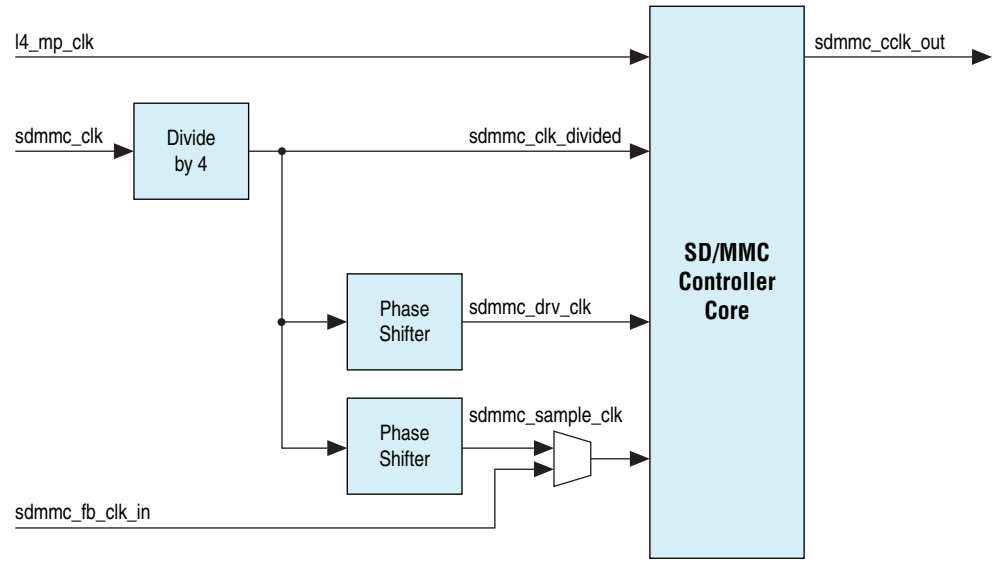
表 11-13 列出了 SD/MMC 控制器时钟。

表 11-13. SD/MMC 控制器时钟

时钟名	方向	说明
sdmmc_clk	进	用于 SD/MMC 控制器 CIU 的时钟
l4_mp_clk	进	用于 SD/MMC 控制器 BIU 的时钟
sdmmc_cclk_out	出	用于卡的生成输出时钟
sdmmc_sample_clk	内部	用于采样卡中的命令和数据的 sdmmc_clk 的相移时钟
sdmmc_drv_clk	内部	sdmmc_clk 的相移时钟，用于控制器驱动命令和数据到卡中，以满足保持时间要求
sdmmc_clk_divided	内部	sdmmc_clk 的四分频 (divide-by-four) 时钟

图 11 - 10 显示了多种时钟到控制器的连接。

图 11 - 10. SD/MMC 控制器时钟连接



来自时钟管理器的 `sdrmclk` 时钟在被传递到移相器和控制器前被四分频 (`sdrmclk_divided` 时钟)。移相器用于生成 `sdrmclk_drv_clk` 和 `sdrmclk_sample_clk` 时钟。这些移相器提供高达八个相移，包括 0、45、90、135、180、225、270 和 315 度。`sdrmclk_sample_clk` 时钟能被移相器的输出驱动。相移度数和 `sdrmclk_sample_clk` 时钟源的选择在系统管理器 (system manager) 中完成。关于设置相移和选择 `sdrmclk_sample_clk` 时钟源的详细信息，请参考第 11 - 34 页的“时钟设置 (Clock Setup)”。

控制器生成 `sdrmclk_cclk_out` 时钟，该时钟被驱动到卡中。关于 `sdrmclk_cclk_out` 时钟生成的详细信息，请参考第 11 - 26 页的“时钟控制模块”。

复位

SD/MMC 控制器有一个复位信号。复位管理器通过冷或热复位将此信号驱动到 SD/MMC 控制器。

 关于详细信息，请参考 *Cyclone V Device Handbook* 卷 3 中的 *Reset Manager* 章节。

接口信号

表 11 - 14 列出了 SD/MMC 控制器接口信号的 I/O 管脚使用。

表 11 - 14. SD/MMC 控制器接口 I/O 管脚

信号	宽度	方向	说明
<code>sdrmclk_cclk_out</code>	1	出	从控制器到卡的时钟
<code>sdrmclk_cmd</code>	1	进 / 出	卡命令
<code>sdrmclk_pwren</code>	1	出	外部器件电源使能
<code>sdrmclk_data</code>	8	进 / 出	卡数据

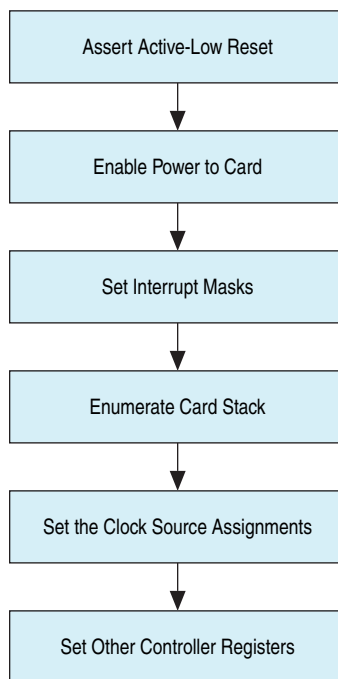
SD/MMC 控制器编程模型

初始化

本小节描述了如何初始化 SD/MMC 控制器。

图 11-11 显示了 SD/MMC 控制器的初始化流程。控制的电源和时钟稳定后，控制器 active-low 复位被置位。复位流程初始化控制器中的寄存器、FIFO 缓存指针、DMA 接口控制和状态机。

图 11-11. SD/MMC 控制器初始化流程



上电复位后，软件必须执行下面步骤：

1. 使能卡的电源前，要确认对电压适配器的电压设置正确。
2. 通过设置电源使能寄存器 (pwren) 中的电源使能比特 (power_enable) 为 1 来使能卡的电源。执行下一步之前，等待电源上升时间。
3. 通过重设置 intmask 寄存器中相应比特为 0 来设置中断屏蔽。
4. 设置 ctrl 寄存器中的 int_enable 比特为 1。



Altera 建议您写入 0xFFFFFFFF 到 rintsts 寄存器，以在 int_enable 比特设为 1 前清除所有未决中断。

5. 根据卡类型来识别卡堆栈，必须将时钟频率限制到 400 kHz，以符合 SD/MMC/CE-ATA 标准。关于更多信息，请参考第 11-31 页的“枚举卡堆栈 (Enumerated Card Stack)”。
6. 设置时钟源分配。使用控制器的 clkdiv 和 clksrc 寄存器设置卡频率。关于更多信息，请参考第 11-34 页的“时钟设置 (Clock Setup)”。

7. 初始化过程中可以设置下面的一般寄存器和字段：

- `tmout` 寄存器的响应超时字段 (`response_timeout`)。典型值为 0x64。
- `tmout` 寄存器的数据超时字段 (`data_timeout`)，最大值如下：
 - $10 * N_{AC}$
 - $= 10 * ((TAAC * F_{Op}) + (100 * NSAC))$
 其中：
 - N_{AC} = 卡器件的总访问时间
 - $TAAC$ = 数据访问时间的时间相关因子 (time-dependent factor)
 - F_{Op} = 用于卡操作的卡时钟频率
 - $NSAC$ = 数据访问时钟的最坏情况的时钟速率相关因子 (rate-dependent factor)
- 主机 FIFO 缓存延迟
 - 读操作：主机开始从满 FIFO 缓存读取数据前的消逝的时间
 - 写操作：主机开始写入空 FIFO 缓存前的消逝的时间
- 去抖计数寄存器 (`debncn`)。典型的去抖值是 25 ms。
- FIFO 阈值水印寄存器 (`fifoth`) 的 TX 水印字段 (`tx_wmark`)。通常情况下，阈值设为 512，是 FIFO 缓存深度的一半。
- `fifoth` 寄存器的 RX 水印字段 (`rx_wmark`)。通常情况下，阈值设为 511。

这些寄存器不需要随着每个 SD/MMC/CE-ATA 命令的改变而改变。可以根据 SD/MMC/CE-ATA 规范对这些寄存器设置一个典型值。

枚举卡堆栈 (Enumerated Card Stack)

卡堆栈执行下面任务：

- 发现连接卡
- 设置连接卡中相关的卡地址寄存器 (RCA)
- 读取卡指定信息
- 本地存储卡指定信息

连接到控制器的卡可以是一个 MMC，CE-ATA，SD 或者 SDIO (包括 IO ONLY，MEM ONLY 和 COMBO) 卡。要识别连接的卡类型，需要执行下面的发现流程 (discovery sequence)：

1. 将 `ctype` 寄存器中的卡宽 1 或 4 bit (`card_width2`) 和卡宽 8 bit (`card_width1`) 复位成 0。
2. 识别卡类型 SD, MMC, SDIO 或者 SDIO-COMBO:
 - a. 发送一个 0 自变量的 SD/SDIO `IO_SEND_OP_COND` (CMD5) 命令到卡中。
 - b. 在控制器上读取 `resp0`。 `IO_SEND_OP_COND` 命令的响应提供卡所支持的电压。
 - c. 发送 `IO_SEND_OP_COND` 命令和自变量中所需电压窗口。此命令设置电压窗口，并使卡退出初始化状态。

- d. 检查 resp0 中的 bit 27。
 - 如果 bit 27 是 0, 那么 SDIO 卡是 IO ONLY。这种情况下, 直接跳到步骤 5。
 - 如果 bit 27 是 1, 那么卡类型是 SDIO COMBO。继续下面的步骤。
3. 如果 SDIO 卡类型是 COMBO 或者从前一个 IO_SEND_OP_COND 命令没有接收到响应, 那么仅继续执行此步骤。否则, 跳到步骤 5。
 - a. 发送带有以下参数的 SD/SDIO SEND_IF_COND (CMD8) 命令:
 - Bit[31:12] = 0x0 (反转比特)
 - Bit[11:8] = 0x1 (电源电压值)
 - Bit[7:0] = 0xAA (SD 存储卡所需的检查码型, 兼容 *SDIO Simplified Specification Version 2.00* 及后续版本。)
 -  请参考第 11 - 67 页的“参考文献”中描述的 *SDIO Simplified Specification Version 2.00*。
 - b. 如果接收到前一个 SEND_IF_COND 命令的响应, 那么卡支持 SD High-Capacity, 兼容 *SD Specifications, Part 1, Physical Layer Simplified Specification Version 2.00*。
如果没有接收到响应, 那么跳到步骤 e。
 - c. 发送带有以下参数的 SD_SEND_OP_COND (ACMD41) 命令:
 - Bit[31] = 0x0 (反转比特)
 - Bit[30] = 0x1 (高性能状态)
 - Bit[29:25] = 0x0 (反转比特)
 - Bit[24] = 0x1 (S18R -- 支持 1.8V 电压切换)
 - Bit[23:0] = 支持的电压范围
 - d. 如果接收到前一个 SD_SEND_OP_COND 命令的响应, 那么卡的类型是 SDHC, 否则是 MMC 或者 CE-ATA。在任何一种情况下, 都直接跳到步骤 5。
 - e. 如果没有接收到初始 SEND_IF_COND 命令的响应, 那么卡不支持 High Capacity SD2.0。接下来发出 GO_IDLE_STATE 命令, 然后带有以下参数的 SD_SEND_OP_COND 命令:
 - Bit[31] = 0x0 (保留比特)
 - Bit[30] = 0x0 (高容量状态)
 - Bit[29:24] = 0x0 (保留比特)
 - Bit[23:0] = 支持的电压范围
 - f. 如果接收到前一个 SD_SEND_OP_COND 命令的响应, 那么卡是 SD 类型, 否则是 MMC 或者 CE-ATA。



您必须在第一个 SD_SEND_OP_COND 命令之前发出 SEND_IF_COND 命令，以初始化 High Capacity SD 存储卡。当下面任意条件为真时，此卡返回一个 busy 信号作为对 SD_SEND_OP_COND 命令的响应。

- 此卡执行它的内部初始化进程。
- 在 SD_SEND_OP_COND 命令前不发出 SEND_IF_COND 命令。
- ACMD41 命令被发出。在命令参数中，对于高性能 SD 卡，Host Capacity Support (HCS) 比特设为 0。

4. 通过执行下面步骤来确定卡是否是一个 CE-ATA 1.1, CE-ATA 1.0 或 MMC 器件：

- a. 通过尝试选择 ATA 模块来确定卡是否是一个 CE-ATA v1.1 卡器件。发送 SD/SDIO SEND_IF_COND 命令，查询外部卡中的 EXT_CSD 寄存器模块的 byte 504 (S_CMD_SET)。

- 如果 bit 4 设为 1，那么卡器件支持 ATA 模式。发送 SWITCH_FUNC (CMD6) 命令，将 EXT_CSD 寄存器切片 191 (CMD_SET) 的 ATA bit (bit 4) 设为 1。此命令选择 ATA 模式并启动 ATA 命令组。

通过从 EXT_CSD 寄存器的 byte 191 回读，能够验证当前选择的模式。

跳到步骤 5。

- 如果卡器件不支持 ATA 模式，那么它可能是一个 MMC 卡或者 CE-ATA v1.0 卡。执行步骤 b。

- b. 通过发送 RW_REG 命令来确定卡是否是一个 CE-ATA 1.0 卡器件或者 MMC 卡器件。如果接收到一个响应并且响应数据包含 CE-ATA 签名，那么此卡是一个 CE-ATA 1.0 卡器件，否则，是一个 MMC 卡器件。

5. 此时，软件已经确定卡的类型为 SD/SDHC, SDIO 或者 SDIO-COMBO。现在，它必须根据已识别的类型来枚举卡堆栈。

6. 将卡时钟源频率设置成识别时钟频率 400 KHz。使用下面其中一个发现命令 (discovery command) 序列：

- 对于 SD 卡或者 SDIO 存储器部分，发送下面的 SD/SDIO 命令序列：

- GO_IDLE_STATE
- SEND_IF_COND
- SD_SEND_OP_COND (ACMD41)
- ALL_SEND_CID (CMD2)
- SEND_RELATIVE_ADDR (CMD3)

- 对于 SDIO 卡，发送下面的命令序列：

- IO_SEND_OP_COND.
- 如果功能计数 (function count) 是有效的，那么发送 SEND_RELATIVE_ADDR 命令。

- 对于 MMC, 发送下面命令序列：
 - GO_IDLE_STATE
 - SEND_OP_COND (CMD1)
 - ALL_SEND_CID
 - SEND_RELATIVE_ADDR
- 7. 通过对 `clkdiv` 寄存器（分频 `sdmnc_clk` 时钟）写入一个值，您可以在发现后修改卡时钟频率。
下面列表显示了各种类型卡的典型时钟频率：
 - SD 存储卡， 25 MHz
 - MMC 卡器件， 12.5 MHz
 - 全速 SDIO， 25 MHz
 - 低速 SDIO， 400 kHz

时钟设置 (Clock Setup)

以下 SD/MMC 控制器的寄存器支持软件选择卡所需的时钟频率：

- `clksrc`
- `clkdiv`
- `clkena`

如本节中所描述，当控制器接收到一个更新时钟命令时会加载这些寄存器。执行下面步骤修改卡时钟频率：

1. 禁用时钟前，要确保卡没有忙于之前的数据命令。通过验证状态寄存器 (`status`) 的 `data_busy` 比特为 0 来确认这一点。
2. 将 `clkena` 寄存器的 `cclk_enable` 比特复位成 0 来禁止卡时钟的生成。
3. 将 `clksrc` 寄存器复位成 0。
4. 将 `cmd` 寄存器中的下面比特设为 1：
 - `update_clk_regs_only`— 指定更新时钟命令
 - `wait_prvdata_complete`— 确保直到完成全部现有的数据传输才修改时钟参数
 - `start_cmd`— 启动命令
5. 等待直到 `start_cmd` 和 `update_clk_regs_only` 比特修改成 0。时钟修改完成时没有中断。命令完成时，控制器不设置 `rintsts` 寄存器中的 `command_done` 比特。如果队列中已经有另一个命令，那么控制器可能发送一个硬件锁定错误 (`hardware lock error`) 信号。在此情况下，返回到步骤 4。
关于硬件锁定错误的信息，请参考第 11 - 55 页的“中断与错误处理”。
6. 将时钟管理器外设 PLL 组 (`perpllgrp`) 的 `enable` 寄存器中的 `sdmnc_clk_enable` 比特复位成 0。
7. 在系统管理器中的 SDMMC 控制器组 (`sdmncgrp`) 的控制寄存器 (`ctrl`) 中，设置驱动时钟相移选择 (`drvsel`) 和采集时钟相移选择 (`smplsel`) 比特来指定所需的相移值。
8. 将时钟管理器 `perpllgrp` 组的 `Enable` 寄存器中的 `sdmnc_clk_enable` 比特设为 1。
9. 将控制器的 `clkdiv` 寄存器设置成所需时钟频率的正确分频值。


10. 将 `clkena` 寄存器的 `cclk_enable` 比特设为 1, 以使能卡时钟生成。

您也可以使用 `clkena` 寄存器使能低功耗 (low-power) 模式, 当卡处于空闲超过 8 个时钟周期时自动停止 `sdmmc_cclk_out` 时钟。

控制器 /DMA/FIFO 缓存复位使用

下列显示了对 SD/MMC 控制器中不同部分的复位影响:

- 控制器复位—通过设置 `ctrl` 寄存器中的 `controller_reset` 比特为 1 来复位控制器。控制器复位复位 CIU 和状态机, 也复位 BIU-to-CIU 接口。由于该复位比特是自清零的 (self-clearing), 因此发出复位命令后要等待该比特变成 0。
- FIFO 缓存复位—通过设置 `ctrl` 寄存器中的 FIFO 复位比特 (`fifo_reset`) 为 1 来复位 FIFO 缓存。FIFO 缓存复位复位 FIFO 缓存中的 FIFO 缓存指针和计数器。由于该复位比特是自清零的 (self-clearing), 因此发出复位命令后要等待该比特变成 0。
- DMA 复位—通过设置 `ctrl` 控制器中的 DMA 复位比特 (`dma_reset`) 为 1 来复位内部 DMA 控制器, 这会即刻终止所有正在进行的 DMA 传递。由于该复位比特是自清零的 (self-clearing), 因此发出复位命令后要等待该比特变成 0。

 要确保在执行 DMA 复位前 DMA 是空闲的, 否则可能会使 L3 互联处于不确定状态。

Altera 建议首先将 `ctrl` 寄存器中的 `controller_reset`, `fifo_reset` 和 `dma_reset` 比特设成 1, 然后通过另一个写操作将 `rintsts` 寄存器复位成 0 以清除所有复位导致发生的中断 (resultant interrupt)。

使能 FIFO 缓存 ECC

要通过 ECC 保护 FIFO 缓存数据, 就必须在执行 SD/MMC 控制器操作前使能 ECC 功能。执行下面的步骤使能 FIFO 缓存 ECC 功能:

1. 验证没有对控制器执行任何命令。
2. 确保初始化 FIFO 缓存。通过对所有 1024 FIFO 缓存位置写入 0 来初始化 FIFO 缓存。对 0x200 到最大 FIFO 缓存容量的任何地址的 FIFO 缓存写操作都是有效的。
3. 将系统管理器的 `eccgrp` 组中的 `sdmmc` 寄存器中的 SDMMC RAM ECC 单和双, 可纠正错误中断状态比特 (`serrporta`, `derrporta`, `serrportb` 和 `derrportb`) 设为 1, 以清除所有之前检测到的 ECC 错误。
4. 通过将 `ctrl` 寄存器中的 `fifo_reset` 比特设为 1 来复位 FIFO 缓存, 此操作将复位 FIFO 缓存中的指针和计数器。由于该复位比特是自清零的 (self-clearing), 因此发出复位命令后要等待该比特变成 0。
5. 将系统管理器的 `eccgrp` 组中的 `sdmmc` 寄存器中的 `en` 比特设为 1 来对 SD/MMC 控制器中的 FIFO 缓存使能 ECC。

非数据传递命令 (Non-Data Transfer Commands)

要发送非数据传递命令, 软件需要写入相应的参数到 `cmd` 寄存器和 `cmdarg` 寄存器。通过使用这两个寄存器, 控制器生成命令并发送到 CMD 管脚。控制器通过 `rintsts` 寄存器中的错误列表报告命令响应中的错误。

当接收到一个响应 — 无论是错误的还是有效的 — 控制器都会将 `rintsts` 寄存器中的 `command_done` 比特设为 1。短响应被复制到 `resp0`，而长响应则被复制到全部四个寄存器 (`resp0`，`resp1`，`resp2` 和 `resp3`)。对于长响应，`resp3` 的 bit 31 代表 MSB，`resp0` 的 bit 0 代表 LSB。

对于基本的和非数据传递命令，执行下面步骤：

1. 写入 `cmdarg` 寄存器相应的命令参数。
2. 使用第 11 - 37 页的表 11 - 15 中的设置写入 `cmd` 寄存器。
3. 等待控制器接受命令。接受命令后，`start_cmd` 比特变为 0。

当命令被加载到控制器中时会出现下面情况：

- 如果之前的命令没有被处理，那么控制器接受执行的命令并将 `cmd` 寄存器中 `start_cmd` 比特复位成 0。如果早先的命令正在被执行，控制器将新的命令加载到命令缓存中。
 - 如果控制器不能加载新的命令 — 也就是，一个命令正在处理，第二个命令在缓存中，第三个命令尝试运行 — 控制器生成一个硬件锁错误 (`hardware lock error`)。
4. 检查是否存在硬件锁错误。
 5. 等待命令执行完成。接收到来自卡的响应或者响应超时后，控制器将 `rintsts` 寄存器中的 `command_done` 比特设为 1。软件轮询该比特或者响应生成的中断（如果使能）。
 6. 检查响应超时引导应答接收 (`bar`)，`rcrc` 或者 `re` 比特是否设为 1。软件能够响应这些错误导致的中断或者轮询 `rintsts` 寄存器的 `re`，`rcrc` 和 `bar` 比特。如果没有接收到响应错误，那么响应是有效的。如果需要，软件能够从响应寄存器复制响应。



执行命令期间，软件不能修改时钟参数。

表 11 - 15. 非数据传递命令的 cmd 寄存器设置

参数	值	说明
默认		
start_cmd	1	命令执行后，该比特自复位成 0。
use_hold_reg	1 或 0	根据使用的速度模式选择值。
update_clk_regs_only	0	表明不是一个时钟更新命令
data_expected	0	表明不是一个数据命令
card_number	1	一个卡
cmd_index	Command Index	此参数表示命令编号。例如，8 代表 SD/SDIO SEND_IF_COND (CMD8) 命令。
send_initialization	0 或 1	1— 卡复位命令，例如 SD/SDIO GO_IDLE_STATE 命令 0— 其它命令
stop_abort_cmd	0 或 1	1— 命令停止数据传递，例如 SD/SDIO STOP_TRANSMISSION 命令 0— 其它命令
response_length	0 或 1	1— R2（长）响应 0— 短响应
response_expect	0 或 1	0— 无响应的命令，例如：SD/SDIO GO_IDLE_STATE，SET_DSR (CMD4) 或者 GO_INACTIVE_STATE (CMD15)。 1— 其它命令
用户可选		
wait_prvdata_complete	1	在命令行上发送一个命令前，主机必须等待正在进行的数据命令完成。Altera 建议将该比特设成 1，除非当前命令要查询状态或者在数据传递过程中要停止数据传递。
check_response_crc	1 或 0	1— 如果响应包含有效 CRC，并且需要软件来交叉校验响应 CRC 比特。 0— 代表其它情况

数据传递命令

数据传递命令在存储卡与控制器之间传递数据。要发出一个数据命令，控制器需要命令参数，总数据大小和数据块大小。控制器 FIFO 缓冲器缓存传递到存储卡的数据，或者缓存来自存储卡的数据。

发出数据传递命令前，软件必须通过执行下面步骤来确认卡不忙并处于传递状态：

1. 发出一个 SD/SDIO SEND_STATUS (CMD13) 命令。控制器发送卡状态作为对命令的响应。
2. 检查卡的 busy 状态。
3. 等待直到卡不忙。
4. 检查卡的传递状态。如果卡处于 stand-by 状态，那么发出一个 SD/SDIO SELECT/DESELECT_CARD (CMD7) 命令将其置于传递状态。



在 CE-ATA RW_BLK 写传递过程中，MMC busy 信号可能在最后一个数据块之后被置位。如果 CE-ATA 卡器件中断被禁用（卡器件的 ATA 控制寄存器的 nIEN 比特被设为 1），rintsts 寄存器中的 dto 比特被设为 1，即使卡发送 MMC BUSY。主机不能在 CMD61 命令之后发出一个 CMD60 命令来检查 ATA busy 状态，而必须执行下面其中一个操作：

- 在发出一个新的 CMD60 命令前发出 SEND_STATUS 命令并检查 MMC busy 状态
- 在发出一个新的 CMD60 命令前发出 CMD39 命令并检查 ATA busy 状态

对于数据传递命令，软件必须将 ctype 寄存器设置成在卡中编程的总线宽度。

在数据传递过程中，控制器对不同的条件生成中断，由下面的 rintsts 寄存器比特反映：

1. dto—数据传递结束或被终止。如果存在响应超时错误，那么控制器将不尝试任何数据传递，Data Transfer Over 比特从不被设置。
2. 发送 FIFO 数据请求比特 (txdr)—达到发送数据的 FIFO 缓存阈值；需要软件（如果可用）将数据写入 FIFO 存储器中。
3. 接收 FIFO 数据请求比特 (rxdr)—达到接收数据的 FIFO 缓存阈值；需要软件从 FIFO 缓存中读取数据。
4. hto—FIFO 缓存在数据发送期间是空的，在数据接收期间是满的。除非软件通过写入数据来改变空的情况，或通过读数据来改变满的情况，控制器不能继续数据传递。卡时钟停止。
5. bds— 在超时期间，卡还没有发送数据。
6. dcrc—数据接收期间出现了一个 CRC 错误。
7. sbe—数据接收期间没有接收到起始比特。
8. ebe—数据接收期间或者对于写操作没有接收到结束比特，卡指示一个 CRC 错误。

条件 6, 7 和 8 指示接收的数据可能存在错误。如果存在响应超时，则不会有数据传递。

单数据块或多数据块读操作 (Single-Block or Multiple-Block Read)

要实现单数据块或者多数据块读操作，软件需要执行下面的步骤：

1. 将数据大小（以字节为单位）写入到 bytcnt 寄存器中。对于多数据块读操作，bytcnt 必须是数据块大小的倍数。
2. 将数据块大小（以字节为单位）写入到 blksize 寄存器中。控制器预期数据以 blksize 数据块大小从卡返回。
3. 如果读返回延迟（包括卡延迟）大于 sdmmc_clk_divided 的一半，那么写入卡阈值控制寄存器 (cardthrc1) 以确保从卡到主机的数据块传递过程中卡时钟不会停止。关于更多信息，请参考第 11-53 页的“卡读取阈值”。



如果卡读取阈值使能比特 (cardrdthren) 为 0，那么主系统必须确保读取 RX FIFO 缓存的速率要大于写入该缓存的速率，以保证 RX FIFO 缓存在读数据传递期间不会变满。否则可能会出现上溢。

4. 将数据读操作的起始数据地址写入到 cmdarg 寄存器。
5. 将表 11-16 中列出的参数写入 cmd 寄存器中。对于 SD 和 MMC 卡，将 SD/SDIO READ_SINGLE_BLOCK (CMD17) 命令用于单数据块读操作，将 READ_MULTIPLE_BLOCK (CMD18) 命令用于多数据块读操作。对于 SDIO 卡，将 IO_RW_EXTENDED (CMD53) 命令用于单数据块以及多数据块传递。写入 cmd 寄存器后，控制器开始执行命令。发送命令到总线后生成 Command Done 中断。
6. 软件必须检查 rintsts 寄存器中 dcrc, bds, sbe 和 ebe 比特中报告的数据错误中断。如果需要，软件能够发送一个 SD/SDIO STOP 命令来终止数据传递。

7. 软件必须检查 `rintsts` 寄存器中的主机超时条件：
- 接收 FIFO 缓存数据请求
 - 主机的数据缺乏 (data starvation) — 主机从 FIFO 缓存读取数据的速度不足以跟上卡中的数据。软件必须执行下面步骤来纠正此情况：
 - 读取 `status` 寄存器的 `fifo_count` 字段
 - 从 FIFO 缓存读取相应数量的数据
- 在这两种情况中，软件都必须从 FIFO 缓存中读取数据，以腾出空间接收更多的数据。
8. 接收到 DTO 中断时，软件必须从 FIFO 缓存中读取剩余的数据。

表 11 - 16. 单数据块和多数据块读操作的 `cmd` 寄存器设置

参数	值	说明
默认		
<code>start_cmd</code>	1	命令执行后，该比特自复位成 0。
<code>use_hold_reg</code>	1 或 0	根据使用的速度模式选择值。
<code>update_clk_regs_only</code>	0	不需要更新时钟参数
<code>data_expected</code>	1	数据命令
<code>card_number</code>	1	一个卡
<code>transfer_mode</code>	0	数据块传递
<code>send_initialization</code>	0	1— 卡复位命令，例如：SD/SDIO <code>GO_IDLE_STATE</code> 命令 0— 其它命令
<code>stop_abort_cmd</code>	0	1— 停止数据传递的命令，例如：SD/SDIO <code>STOP_TRANSMISSION</code> 命令 0— 其它命令
<code>send_auto_stop</code>	0 或 1	根据第 11 - 23 页的表 11 - 10 进行设置
<code>read_write</code>	0	从卡读取
<code>response_length</code>	0	1— R2（长）响应 0— 短响应
<code>response_expect</code>	1 或 0	0— 无响应的命令，例如：SD/SDIO <code>GO_IDLE_STATE</code> ， <code>SET_DSR</code> 和 <code>GO_INACTIVE_STATE</code> 。 1— 其它命令
用户可选		
<code>wait_prvdata_complete</code>	1 或 0	0— 即刻发送命令到 CIU 1— 前一个数据传递结束后发送命令
<code>check_response_crc</code>	1 或 0	0— 控制器不要检查响应 CRC 1— 控制器必须检查响应 CRC
<code>cmd_index</code>	命令索引	将此参数设成命令编号。例如，17, 18 分别代表 SD/SDIO <code>READ_SINGLE_BLOCK</code> (CMD17) 和 <code>READ_MULTIPLE_BLOCK</code> (CMD18)。

单数据块或多数据块写操作 (Single-Block or Multiple-Block Write)
单数据块或多数据块写操作包括下面几个步骤：

1. 将数据大小（以字节为单位）写入到 `bytcnt` 寄存器中。对于多数据块写操作，`bytcnt` 必须是数据块大小的倍数。
2. 将数据块大小（以字节为单位）写入到 `blksiz` 寄存器中。控制器发送 `blksiz` 数据块大小的数据。
3. 将数据必须写入的数据地址写入到 `cmdarg` 寄存器。
4. 将数据写入到 FIFO 缓存。要获得最佳性能，主软件应该连续写入数据，直到 FIFO 缓存变满。
5. 使用第 11 - 41 页的表 11 - 17 中列出的参数写入 `cmd` 寄存器。对于 SD 和 MMC 卡，对单模块写操作使用 SD/SDIO `WRITE_BLOCK` (CMD24) 命令，对多模块写操作使用 `WRITE_MULTIPLE_BLOCK` (CMD25) 命令。对于 SDIO 卡，对单模块以及多模块传递都使用 `IO_RW_EXTENDED` 命令。

写入 `cmd` 寄存器后，如果没有执行其它命令，那么控制器就开始执行命令。当命令发送到总线时会生成一个 Command Done 中断。

6. 软件必须检查数据错误中断；也就是 `rintsts` 寄存器的 `dcrc`，`bds` 和 `ebe` 比特。如果需要，软件能够通过发送 SD/SDIO `STOP` 命令来终止数据传递。
7. 软件必须检查 `rintsts` 寄存器中的主机超时情况 (host timeout condition):
 - 发送 FIFO 缓存数据请求
 - 数据缺乏 (data starvation) by the host— 控制器将数据写入卡中的速度要快于主机提供数据的速度。

在这两种情况中，软件必须将数据写入到 FIFO 缓存中。

有两种类型的传递：开端 (open-ended) 和固定长度 (fixed length)。

- 开端传递 — 在开端模块传递中，字节数是 0。在数据传递的最后，软件必须发送 `STOP_TRANSMISSION` 命令 (CMD12)。
 - 固定长度传递 — 字节数是非零值。您必须已经写入字节数到 `bytcnt` 寄存器。如果 `cmd` 寄存器的 `send_auto_stop` 比特设为 1，那么控制器将发出 `STOP` 命令。指定数量的字节传递完成后，控制器发送 `STOP` 命令。Auto Command Done 中断反映 `AUTO_STOP` 命令的完成。对 `AUTO_STOP` 命令的响应写入到 `respl` 寄存器中。
- 如果软件没有将 `cmd` 寄存器中的 `send_auto_stop` 比特设为 1，那么与开端情况中一样，软件必须发出 `STOP` 命令。

当 `rintsts` 寄存器的 `dto` 比特被设置，数据命令完成。

表 11 - 17. 单模块和多模块写的 cmd 寄存器设置

参数	值	说明
默认		
start_cmd	1	命令执行后（被 BIU 接收），此比特自复位到 0。
use_hold_reg	1 或 0	根据所使用的速度模式选择相应值。
update_clk_regs_only	0	不需要更新时钟参数
data_expected	1	数据命令
card_number	1	一个卡
transfer_mode	0	数据块传递
send_initialization	0	可为 1，但仅用于卡复位命令，例如：SD/SDIO GO_IDLE_STATE
stop_abort_cmd	0	可为 1，使命令停止数据传递，例如：SD/SDIO STOP_TRANSMISSION
send_auto_stop	0 或 1	根据第 11 - 23 页的表 11 - 10 进行设置
read_write	1	写入卡
response_length	0	对于 R2（长）响应，可为 1
response_expect	1	对于无响应命令，可为 0。例如：SD/SDIO GO_IDLE_STATE，SET_DSR，GO_INACTIVE_STATE 等等。
用户可选		
wait_prvdata_complete	1	0— 立即发送命令到 CIU 1— 前一个数据传递结束后发出命令
check_response_crc	1	0— 控制器一定不要检查响应 CRC 1— 控制器必须检查响应
cmd_index	Command Index	将此参数设成命令编号。例如，对 SD/SDIO WRITE_BLOCK（CMD24）设为 24，或者对 WRITE_MULTIPLE_BLOCK（CMD25）设为 25。

流读取与写入 (Stream Read and Write)

在流传递中，如果字节数等于 0，那么软件也必须发送 SD/SDIO STOP 命令。如果字节数不是 0，那么当指定数量的字节完成一个传递时，控制器会自动发送 STOP 命令。Auto_command_done 中断反映了此 AUTO_STOP 命令的完成。对 AUTO_STOP 命令的响应被写入 respl 寄存器。流传递仅用于 1-bit 数据总线的卡接口。

流读取的步骤与第 11 - 38 页的“单数据块或多数据块读操作 (Single-Block or Multiple-Block Read)”中描述的数据块读取步骤相同，除了 cmd 寄存器中的下面比特：

- transfer_mode = 0x1（流传递）
- cmd_index = 20（SD/SDIO CMD20）

流写入的步骤与第 11 - 39 页的“单数据块或多数据块写操作 (Single-Block or Multiple-Block Write)”中描述的数据块写入步骤相同，除了 cmd 寄存器中的下面比特：

- `transfer_mode = 0x1`（流传递）
- `cmd_index = 11`（SD/SDIO CMD11）

Packed 命令

为减少开销，读写命令能够打包成一组命令——全读或全写——在总线上在一此传输中传输所有命令的数据。使用 SD/SDIO SET_BLOCK_COUNT (CMD23) 命令提前陈述将要传递的数据块数量。然后发出一个 READ_MULTIPLE_BLOCK 或 WRITE_MULTIPLE_BLOCK 命令以读或写多个数据块。

- SET_BLOCK_COUNT—设置数据块数量（使用 READ_MULTIPLE_BLOCK 或 WRITE_MULTIPLE_BLOCK 命令传递的数据块数量）
- READ_MULTIPLE_BLOCK—多数据块读命令
- WRITE_MULTIPLE_BLOCK—多数据块写命令

应用软件以包的形式组织 packed 命令，packed 命令对控制器是透明的。



关于 packed 命令的更多信息，请参考第 11 - 67 页的“参考文献”中引用的 JEDEC Standard No. 84-A441。

传递 Stop 和 Abort 命令

此部分描述了 stop 和 abort 命令。SD/SDIO STOP_TRANSMISSION 命令能够终止存储器卡与控制器之间的数据传递。ABORT 命令能够对 SDIO 卡终止 I/O 数据传递。

STOP_TRANSMISSION (CMD12)

输出传递期间，主机能够随时在 CMD 管脚上发送 STOP_TRANSMISSION (CMD12) 命令。执行下面的步骤发送 STOP_TRANSMISSION 命令到 SD/SDIO 卡器件：

1. 将 cmd 寄存器的 `wait_prvdata_complete` 比特设为 0。
2. 将 cmd 寄存器中的 `stop_abort_cmd` 设为 1，以确保 CIU 停止。

STOP_TRANSMISSION 命令是一个非数据传递命令。关于更多信息，请参考第 11 - 35 页的“非数据传递命令 (Non-Data Transfer Commands)”。

ABORT

ABORT 命令仅用于 SDIO 卡。要终止正在传递数据的功能，需要使用 IO_RW_DIRECT (CMD52) 命令编程卡器件中卡通用控制寄存器 (CCCR) 的地址 0x06 上的 ASx[2:0] 中的 ABORT 功能编号。CCCR 位于卡空间 0x0 - 0xFF 的底部。

ABORT 命令是一个非数据传递命令。关于更多信息，请参考第 11 - 35 页的“非数据传递命令 (Non-Data Transfer Commands)”。

执行下面步骤发送 ABORT 命令到 SDIO 卡器件：

1. 设置 cmdarg 寄存器以包括表 11 - 18 中列出的相应命令自变量参数。
2. 通过设置 cmd 寄存器的下面域来发送 IO_RW_DIRECT 命令：
 - 将 command index 设成 0x52 (IO_RW_DIRECT)。
 - 将 cmd 寄存器的 `stop_abort_cmd` 比特设成 1 以通知控制器主机终止了数据传递。
 - 将 cmd 寄存器的 `wait_prvdata_complete` 比特设为 0。

3. 等待 `rintsts` 寄存器中的 `cmd` 比特修改成 1。
4. 对所有错误读取响应寄存器中 `IO_RW_DIRECT` 命令 (R5) 的响应。


 关于响应值的更多信息，请参考第 11 - 67 页的“参考文献”中描述的 *Physical Layer Simplified Specification, Version 3.01*。

表 11 - 18. SD/SDIO ABORT 命令的 `cmdarg` 寄存器设置

比特	内容	值
31	R/W 标志位	1
30:28	功能编号	0, 访问卡器件中的 CCCR
27	RAW 标志位	1, 如果需要在写入之后读取
26	Don't care	-
25:9	寄存器地址	0x06
8	Don't care	-
7:0	写数据	要终止的功能代码

内部 DMA 控制器操作

要获得更佳的性能，您可以使用内部 DMA 控制器传递主机与控制器之间的数据。本部分描述了内部 DMA 控制器的初始化过程，发送序列和接收序列。

内部 DMA 控制器初始化

执行下面步骤初始化内部 DMA 控制器：

1. 设置所需的 `bmod` 寄存器比特：
 - 如果在 DMA 传递中间 `bmod` 寄存器的内部 DMA 控制器使能比特 (`de`) 设为 0，那么这种改变无效。禁用仅对新的数据传递命令有效。
 - 即刻发出一个软件复位将终止传递。发出软件复位之前，Altera 建议主机通过将 `ctrl` 寄存器的 `dma_reset` 比特设为 1 来复位 DMA 接口。
 - `bmod` 寄存器中的 `pbl` 域是只读的，是 `fifoth` 寄存器中 DMA 多传输大小 (`dw_dma_multiple_transaction_size`) 内容的直接反映。
 - 对于系统性能，必须相应地设置 `bmod` 寄存器的 `fb` 比特。
2. 根据下面的指南写入 `idinten` 寄存器以屏蔽不必要的中断：
 - 当置位一个 Descriptor Unavailable 中断时，软件需要形成描述符，相应地设置它自己的比特，然后写入轮询命令寄存器 (`pldmnd`)，使内部 DMA 控制器重新提取描述符。
 - 软件使能异常中断是必要的，因为与传递有关的任何错误都会报告给软件。
3. 在存储器中填充一个发送或者接收描述符列表。然后将表中的第一个描述符的基地址写入到内部 DMA 控制器的描述符列表基地址寄存器 (`dbaddr`) 中。DMA 控制器然后开始加载存储器中的描述符。

接下来的两个章节“内部 DMA 控制器发送序列”和第 11 - 44 页的“内部 DMA 控制器接收序列”详细介绍了该步骤。

内部 DMA 控制器发送序列

执行下面步骤来使用内部 DMA 控制器发送数据：

1. 主机设置 Descriptor 域 (DES0—DES3) 用于发送，并将 OWN 比特 (DES0[31]) 设为 1。主机也将写入到 SD 卡的数据加载到系统存储器中的数据缓存中。
2. 主机将相应的写数据命令 (SD/SDIO WRITE_BLOCK 或 WRITE_MULTIPLE_BLOCK) 写入到 cmd 寄存器中。内部 DMA 控制器决定需要执行写数据传递。
3. 主机在 fifoth 寄存器中的 tx_wmark 域设置所需的发送阈值级 (transmit threshold level)。
4. 内部 DMA 控制器引擎提取描述符并检查 OWN 比特。如果 OWN 比特设为 0，那么主机具有描述符。在此情况下，内部 DMA 控制器进入暂挂 (suspend) 状态并置位 Descriptor Unable 中断。主机然后需要将描述符 OWN 比特设为 1，并通过对 pldmnd 寄存器写入任意值来释放 DMA 控制器。
5. 主机必须将描述符基地址写入 dbaddr 寄存器中。
6. 内部 DMA 控制器等待 rintsts 寄存器中的 Command Done (CD) 比特设为 1，无 BIU 错误。此情况表明传递能够完成。
7. 内部 DMA 控制器引擎等待来自 BIU 的 DMA 接口请求。BIU 将每个传递分成较小的块 (chunk)。每个块是一个 DMA 的内部请求。根据发送阈值生成此请求。
8. 内部 DMA 控制器从系统存储器中的数据缓存中提取发送数据，然后将这些数据传递到 FIFO 缓存，以准备发送到卡中。
9. 当数据跨越多个描述符时，内部 DMA 控制器提取下一个描述符，并继续对其进行操作。描述符 DES0 域中的 Last Descriptor 比特表明数据是否跨越多个描述符。
10. 当数据发送完成，通过将 ti 比特设为 1，idsts 寄存器中的状态信息被更新。此外，DMA 控制器通过更新描述符的 DES0 域将 OWN 比特设为 0。

内部 DMA 控制器接收序列

要使用内部 DMA 控制器接收数据，需执行下面步骤：

1. 主机设置 Descriptor 域 (DES0—DES3) 用于接收，并将 OWN (DES0 [31]) 设成 1。
2. 主机将相应的读数据命令写入到 BIU 中的 cmd 寄存器中。内部 DMA 控制器决定需要执行读数据传递。
3. 主机在 fifoth 寄存器中的 tx_wmark 域设置所需的接收阈值级 (receive threshold level)。
4. 内部 DMA 控制器引擎提取描述符并检查 OWN 比特。如果 OWN 比特设为 0，那么主机具有描述符。在此情况下，内部 DMA 控制器进入暂挂 (suspend) 状态并置位 Descriptor Unable 中断。主机然后需要将描述符 OWN 比特设为 1，并通过向 pldmnd 寄存器写入任意值来释放 DMA 控制器。
5. 主机必须将描述符基地址写入 dbaddr 寄存器中。
6. 内部 DMA 控制器等待 rintsts 寄存器中的 Command Done (CD) 比特设为 1，BIU 无错误。此情况表明传递能够完成。
7. 内部 DMA 控制器引擎等待来自 BIU 的 DMA 接口请求。BIU 将每个传递分成较小的块 (chunk)。每个块是一个 DMA 的内部请求。根据接收阈值生成此请求。
8. 内部 DMA 控制器从 FIFO 缓存中提取数据，然后将这些数据传递到系统存储器中。

9. 当数据跨越多个描述符时, 内部 DMA 控制器提取下一个描述符, 并继续对其进行操作。描述符 DES0 域中的 Last Descriptor 比特表明数据是否跨越多个描述符。
10. 当数据发送完成, 通过将 `ti` 比特设为 1, `idsts` 寄存器中的状态信息被更新。此外, DMA 控制器通过更新描述符的 DES0 域将 `OWN` 比特设为 0。

SDIO 卡器件的命令

这一部分介绍了用于暂停控制器与 SDIO 器件之间的传递的命令。

Suspend 和 Resume 序列

对于 SDIO 卡, 使用 SUSPEND 命令可以暂停 I/O 功能与控制器之间的数据传递。通过另一个功能执行高优先权的数据传递时可能需要这一性能。如果需要, 可以使用 RESUME 命令继续暂停的数据传递。

通过写入 SDIO 卡的 CCCR (Function 0) 中的相应比特来实现 SUSPEND 和 RESUME 操作。使用控制器的 `IO_RW_DIRECT` 命令对 CCCR 进行读写操作。

Suspend

执行下面步骤暂停 (suspend) 数据传递:

1. 检查 SDIO 卡是否支持 SUSPEND/RESUME 协议, 这可以通过卡的偏移 0x08 上的 CCCR 的 SBS 比特完成。
2. 检查所需功能代码的数据传递是否正在进行。当前激活 (active) 的功能代码在卡的偏移 0x0D 上的 bit 3:0, CCCR 的功能选择比特 (FSx) 中反映。



如果总线状态比特 (BS), 地址 0xC 上的 bit 0 设为 1, 那么只有 FSx 比特提供的功能代码是有效的。

3. 要暂停传递, 需要将总线释放比特 (BR), 地址 0xC 上的 bit 2 设为 1。
4. 轮询卡的偏移 0x0C 上的 CCCR 的 BR 和 BS 比特, 直到它们被设置为 0。当前所选的功能正在使用数据总线时, BS 比特是 1。BR bit 保持为 1, 直到总线释放完成。当 BR 和 BS 比特是 0, 所选功能的数据传递暂停。
5. 读数据传递期间, 控制器能够等待来自卡的数据。如果数据传递是一个对卡的读操作, 那么 SUSPEND 命令成功完成后必须通知控制器。控制器然后复位数据状态机并从等待状态中释放出来。要实现这一点, 需要将 `ctrl` 寄存器中的终止读数据比特 (`abort_read_data`) 设为 1。
6. 通过轮询直到 `rintsts` 寄存器中的 `dto` 比特设为 1, 等待数据完成。

确定要传递的暂挂 (pending) 字节的数量, 需要读取控制器的传递 CIU 卡字节数 (`tcbcnt`) 寄存器。从总传递容量中减去此值, 使用得到的结果继续传递。

Resume

执行下面步骤继续数据传递:

1. 检查卡没有处于传递状态, 这表明总线可用于数据传递。
2. 如果卡处于断开状态, 那么使用 `SD/SDIO SELECT/DESELECT_CARD` 命令选择它。卡状态能够作为对 `IO_RW_DIRECT` 或 `IO_RW_EXTENDED` 命令的响应而被检索。
3. 检查将继续的功能已准备用于数据传递。通过读取卡的偏移 0x0F 上的 CCR 中的相应 RF <n> 标记位来确定这一状态。如果 `RF <n> ≥ 1`, 此功能已经准备好数据传输。

 关于 RF<n>标记位的详细信息，请参考第 11 - 67 页的“参考文献”中描述的 *SDIO Simplified Specification Version 2.00*。

- 4. 要继续传递，使用 IO_RW_DIRECT 命令在卡的偏移 0x0D，CCCR 中的 FSx 比特上写入功能代码。形成 IO_RW_DIRECT 命令的自变量，并写入到 cmdarg 寄存器中。表 11 - 19 列出了比特值。

表 11 - 19. RESUME 命令的 cmdarg 比特值

比特	内容	值
31	R/W 标志位	1
30:28	功能编号	0, CCCR 访问
27	RAW 标志位	1, 写入后读取
26	Don't care	-
25:9	寄存器地址	0x0D
8	Don't care	-
7:0	写数据	要继续的功能代码

- 5. 将数据块大小写入到 blksiz 寄存器中，数据以该数据块尺寸为单位进行传递。
- 6. 将字节数写入到 bytcnt 寄存器。指定要传递的剩余字节的总数量。软件负责处理数据。

要确定传递的暂挂字节的数量，需读取传递 CIU 卡字节数寄存器 (tbcnt)。从总的传递容量减去此值得到要传递的剩余字节。
- 7. 与数据块传递操作类似，写入 cmd 寄存器。当写入 cmd 寄存器时，命令被发送，功能继续数据传递。关于更多信息，请参考第 11 - 38 页的“单数据块或多数据块读操作 (Single-Block or Multiple-Block Read)”和第 11 - 39 页的“单数据块或多数据块写操作 (Single-Block or Multiple-Block Write)”。
- 8. 读取 SDIO 卡器件的继续数据标志位 (resume data flag (DF))。按如下解读 DF 标志位：
 - DF=1— 功能函数有要传递的数据，一旦继续执行功能或存储器，功能就开始数据传递。
 - DF=0— 功能函数没有要传递的数据。如果数据传递是一个读操作，那么控制器等待数据。数据超时过后，它发出一个数据超时错误。

读等待序列 (Read-Wait Sequence)

Read_wait 仅用于 SDIO 卡，暂停功能函数或者存储器的数据传递，允许主机发送命令到 SDIO 卡器件中的任意函数功能。主机能够停止该传递直到所需时长。控制器具有发送停止传递信号到卡中的组件。要发送停止信号，需要执行下面的步骤：

- 1. 通过读取 SDIO 的 SRW 比特，CCCR 中的 offset 0x8 上的 bit 2，检查卡是否支持 read_wait 组件。
- 2. 如果此比特是 1，那么卡中的所有功能都支持 read_wait 组件。使用 SD/SDIO IO_RW_DIRECT 命令读取该比特。
- 3. 如果卡支持 read_wait 信号，那么通过将 ctrl 寄存器中的读等待比特 (read_wait) 设为 1 来将其置位。
- 4. 将 ctrl 寄存器中的 read_wait 比特置位成 0。

CE-ATA 数据传递命令

这一部分介绍了 CE-ATA 数据传递命令。关于基本设置和不同条件下生成的中断的详细信息，请参考第 11 - 37 页的“数据传递命令”。

置位和卡器件发现概述

开始任何的 CE-ATA 操作前，主机必须执行 MMC 置位和初始化过程。在卡进入 MMC TRAN 状态前，主机和卡器件必须协商 MMC 传递 (MMC TRAN) 状态。



关于 MMC TRAN 状态，MMC 置位和初始化的详细信息，请参考第 11 - 67 页的“参考文献”的 *JEDEC Standard No. 84-A441*。

主机必须遵循现有的 MMC 发现过程来协商 MMC TRAN 状态。完成正常的 MMC 置位和初始化过程后，主机必须使用 RW_REG 或者 CMD39 命令来查询初始的 ATA 任务。

默认情况下，MMC 模块大小是 512 字节 — 由 CE-ATA 卡器件中的 srcControl 寄存器的比特 1:0 指示。主机能够协商使用 1 KB 还是 4 KB MMC 模块大小。通过 MMC 中的 srcCapabilities 寄存器，卡指示它能支持的 MMC 模块大小；主机读取此寄存器来协商 MMC 模块大小。当主控制器将 MMC 模块大小写入到卡的 srcControl 寄存器 bits 1:0 时，协商完成。

ATA 任务文件传递概述

ATA 任务文件寄存器被映射到 MMC 寄存器空间中的地址 0x00h 到 0x10h。RW_REG 命令用于发出 ATA 命令，ATA 任务文件以单一 RW_REG MMC 命令序列发送。

在控制器中设置 cmdarg 和 cmd 寄存器之前，主软件堆栈必须将任务文件映像写入到 FIFO 缓存中。主处理器然后在设置 cmd 寄存器比特之前将地址和字节数写入到 cmdarg 寄存器中。

对于 RW_REG 命令，没有来自 CE-ATA 卡器件的 CCS。

使用 RW_MULTIPLE_REGISTER (RW_REG) 命令的 ATA 任务文件传递

此命令包括 CE-ATA 卡器件与控制器之间的数据传递。要发送一个数据命令，控制器需要命令变量，总数据大小和数据块大小。软件通过 FIFO 缓存接收或者发送数据。

通过执行下面的步骤来实现 ATA 任务文件传递（读或写）：

1. 将数据大小（以字节形式）写入到 bytcnt 寄存器。bytcnt 必须等于数据块大小，因为控制器需要一个单数据块传递。
2. 将数据块大小（以字节形式）写入到 blksiz 寄存器。
3. 写入起始寄存器地址到 cmdarg 寄存器。

您必须根据表 11 - 20 到表 11 - 23 设置 cmdarg，cmd，blksiz 和 bytcnt 寄存器。

表 11 - 20. ATA 任务文件传递的 cmdarg 寄存器设置 (1/2)

比特	值	说明
31	1 或 0	读操作设成 0，写操作设成 1
30:24	0	保留（被主处理器设成 0 的比特）
23:18	0	读或写操作的起始寄存器地址 (DWORD 对齐)
17:16	0	寄存器地址 (DWORD 对齐)

表 11 - 20. ATA 任务文件传递的 cmdarg 寄存器设置 (2/2)

比特	值	说明
15:8	0	保留（被主处理器设成 0 的比特）
7:2	16	读或写的字节数（整数 DWORD）
1:0	0	字节数（整数 DWORD）

表 11 - 21. ATA 任务文件传递的 cmd 寄存器设置

比特	值	说明
start_cmd	1	-
ccs_expected	0	不需要 CCS
read_ceata_device	0 或 1	如果 RW_BLK 或 RW_REG 读，则设成 1
update_clk_regs_only	0	没有时钟参数更新命令
card_num	0	-
send_initialization	0	没有初始化序列
stop_abort_cmd	0	-
send_auto_stop	0	-
transfer_mode	0	数据块传递模式。数据块大小和字节数必须匹配读或写的字节数。
read_write	1 或 0	写设成 1，读设成 0
data_expected	1	需要数据
response_length	0	-
response_expect	1	-
cmd_index	Command index	此参数设为命令编号。例如，对 SD/SDIO WRITE_BLOCK (CMD24) 设成 24，对 WRITE_MULTIPLE_BLOCK (CMD25) 设成 25。
wait_prvdata_complete	1	<ul style="list-style-type: none"> ■ 0，立即发送命令 ■ 1，在前一个 DTO 中断后发送命令
check_response_crc	1	<ul style="list-style-type: none"> ■ 0，不检查响应 CRC ■ 1，检查响应 CRC

表 11 - 22. ATA 任务文件传递的 blksiz 寄存器设置

比特	值	说明
31:16	0	设为 0 的保留比特
15:0 (block_size)	16	用于访问整个任务文件 (16, 8-bit 寄存器)。16 字节的数据块大小

表 11 - 23. ATA 任务文件传递的 bytcnt 寄存器设置

比特	值	说明
31:0	16	用于访问整个任务文件 (16, 8-bit 寄存器)。值为 16 的字节数用于设置为 16 的数据块大小。

使用 RW_MULTIPLE_BLOCK (RW_BLK) 命令的 ATA 净荷传递

此命令包括 CE-ATA 卡器件与控制器之间的数据传递。要发送一个数据命令，控制器需要命令变量，总数据大小和数据块大小。软件通过 FIFO 缓存接收或者发送数据。

通过执行下面的步骤来实现 ATA 净荷传递（读或写）：

- 1. 将数据大小（以字节形式）写入到 bytcnt 寄存器。
- 2. 将数据块大小（以字节形式）写入到 blksiz 寄存器。控制器需要一个单 / 多数据块传递。
- 3. 写入 cmdarg 寄存器来表明数据单元数。

您必须根据表 11 - 24 到表 11 - 27 设置 cmdarg，cmd，blksiz 和 bytcnt 寄存器。

表 11 - 24. ATA 净荷传递的 cmdarg 寄存器设置

比特	值	说明
31	1 或 0	读操作设成 0，写操作设成 1
30:24	0	保留（被主处理器设成 0 的比特）
23:16	0	保留（被主处理器设成 0 的比特）
15:8	Data count	数据数单元 [15:8]
7:0	Data count	数据数单元 [7:0]

表 11 - 25. ATA 净荷传递的 cmd 寄存器设置

比特	值	说明
start_cmd	1	-
ccs_expected	1	需要 CCS。如果 CE-ATA 卡器件中的中断被使能，对 RW_BLK 命令设成 1 (ATA 控制寄存器中的 nIEN bit 设为 0)。
read_ceata_device	0 或 1	对 RW_BLK 或 RW_REG 读命令设为 1
update_clk_regs_only	0	没有时钟参数更新命令
card_num	0	-
send_initialization	0	没有初始化序列
stop_abort_cmd	0	-
send_auto_stop	0	-
transfer_mode	0	数据块传递模式。字节数必须是 4kB 的整数倍。数据块大小可以是 512, 1k 或者 4k 字节。
read_write	1 或 0	写为 1, 读为 0
data_expected	1	需要数据
response_length	0	-
response_expect	1	-
cmd_index	Command index	此参数设为命令编号。例如，对 SD/SDIO WRITE_BLOCK (CMD24) 设成 24, 对 WRITE_MULTIPLE_BLOCK (CMD25) 设成 25。
wait_prvdata_complete	1	<ul style="list-style-type: none"> ■ 0, 立即发送命令 ■ 1, 在前一个 DTO 中断后发送命令
check_response_crc	1	<ul style="list-style-type: none"> ■ 0, 不检查响应 CRC ■ 1, 检查响应 CRC

表 11 - 26. ATA 净荷传递的 blksiz 寄存器

比特	值	说明
31:16	0	设为 0 的保留比特
15:0 (block_size)	512, 1024 或 4096	MMC 数据块大小可以通过主机协商的 512, 1024 或 4096 字节。

表 11 - 27. ATA 净荷传递的 bytcnt 寄存器设置

比特	值	说明
31:0	$\langle n \rangle * \text{block_size}$	字节数必须是数据块大小的整数倍。对于 ATA 介质访问命令，字节数必须是 4 KB 的倍数。 $(\langle n \rangle * \text{block_size} = \langle x \rangle * 4 \text{ KB}, \text{ 其中的 } \langle n \rangle \text{ 和 } \langle x \rangle \text{ 是整数})$

CE-ATA CCS

这一部分介绍了禁用 CCS, CCS 超时后的恢复以及 I/O 读发送延迟 (N_{ACTO}) 超时后的恢复。

禁用 CCS

当等待未执行的 RW_BLK 命令的 CCS 时，主机能够通过发送 CCSD 命令来禁用 CCS：

- 发送 CCSD 命令 — 如果控制器的 `ctrl` 寄存器中的 `send_ccsd` 比特设为 1，那么控制器发送 CCSD 命令到 CE-ATA 卡器件。该比特只能在接收到 RW_BLK 命令的响应后设置。
- 发送内部 STOP 命令 — 在发送 CCSD 码型之后发送一个内部生成的 SD/SDIO STOP_TRANSMISSION (CMD12) 命令。当控制器被设置为发送 CCSD 码型时，如果 `ctrl` 寄存器的 `send_auto_stop_ccsd` 比特也设为 1，那么控制器发送内部生成的 STOP 命令到 CMD 管脚。发送 STOP 命令后，控制器将 `rintsts` 寄存器中的 `acd` 比特设为 1。

CCS 超时后的恢复

如果等待 CCS 器件出现超时，那么主机需要发送 CCSD 命令，跟着发送 STOP 命令来终止暂挂的 ATA 命令。主机能够设置控制器以在发送 CCSD 码型后发送内部生成的 STOP 命令：

- 发送 CCSD 命令 — 将 `ctrl` 寄存器中的 `send_ccsd` 比特设为 1。
- 发送外部 STOP 命令 — 终止 CE-ATA 卡器件与控制器之间的数据传递。关于发送 STOP 命令的详细信息，请参考第 11 - 42 页的“传递 Stop 和 Abort 命令”。
- 发送内部 STOP 命令 — 将 `ctrl` 寄存器中的 `send_auto_stop_ccsd` 比特设为 1，指示控制器发送内部生成的 STOP 命令。发送 STOP 命令后，控制器将 `rintsts` 寄存器中的 `acd` 比特设为 1。与设置 `send_ccsd` 比特一起，`send_auto_stop_ccsd` 比特必须设为 1。

I/O 读发送延迟 (N_{ACIO}) 超时后的恢复

如果 CE-ATA 卡器件出现 I/O 读发送延迟 (N_{ACIO}) 超时，那么从超时中恢复可执行下面其中的一个步骤：

- 如果 CE-ATA 卡器件需要 CCS（也就是，`cmd` 寄存器中的 `ccs_expected` 比特设为 1），那么执行第 11 - 51 页的“CCS 超时后的恢复”中的步骤。
- 如果 CE-ATA 卡器件不需要 CCS，那么执行下面的步骤：
 - a. 发送一个外部 STOP 命令。
 - b. 终止控制器与 CE-ATA 卡器件之间的数据传递。

简化的 ATA 命令集

CE-ATA 卡器件对简化的 ATA 命令集的支持是必要的。这一部分介绍了简化的命令集。

IDENTIFY DEVICE 命令

IDENTIFY DEVICE 命令返回一个 512-byte 数据结构到主机，该数据结构描述了指定器件的信息和性能。只有 MMC 数据块大小设为 512 字节时，主机才发出 IDENTIFY DEVICE 命令。其它任何 MMC 数据块大小都有不确定的结果。

主机对 ATA 命令发出一个 RW_REG 命令，并通过 RW_BLK 命令检索数据。

当对 IDENTIFY DEVICE ATA 命令并同时发送一个 RW_REG 命令时，主控制器使用下面的设置。以下列表显示了主要的比特设置：

- cmd 寄存器设置：data_expected 比特设为 0
- cmdarg 寄存器设置：
 - Bit [31] 设为 0
 - Bits [7:2] 设为 128
 - 所有其它比特设为 0
- 任务文件设置：
 - ATA 任务文件的 command 域设为 0xEC
 - 任务文件的 reserved 域设为 0
- bytcnt 寄存器和 blksiz 寄存器的 block_size 域：设为 16

主控制器对数据检索 (RW_BLK 命令) 使用下面的设置：

- cmd 寄存器设置：
 - ccs_expected 设为 1
 - data_expected 设为 1
- cmdarg 寄存器设置：
 - Bit [31] 设为 0 (读操作)
 - Bits [15:0] 设为 1 (数据单元数 = 1)
 - 所有其它比特设为 0
- bytcnt 寄存器和 blksiz 寄存器的 block_size 域：设为 512

READ DMA EXT 命令

READ DMA EXT 命令使用 Data-In 数据传递协议从卡器件读取一组逻辑数据块。主机使用 RW_REG 命令发出用于数据传递的 ATA 命令和 RW_BLK 命令。

WRITE DMA EXT 命令

WRITE DMA EXT 命令使用 Data-Out 数据传递协议将一组逻辑数据块写入卡器件。主机使用 RW_REG 命令发出用于数据传递的 ATA 命令和 RW_BLK 命令。

STANDBY IMMEDIATE 命令

此 ATA 命令使卡器件立即进入最积极的功耗管理模式，此模式仍然包含内部器件描述表。此命令不需要数据传递 (RW_BLK)。

对于那些不提供功耗节省模式的卡器件而言，STANDBY IMMEDIATE 命令返回一个成功的状态指示。主机对 ATA 命令发出一个 RW_REG 命令，通过 SD/SDIO CMD39 或者 RW_REG 命令检索状态。只有 ATA 任务文件的状态域包含成功状态；没有错误状态。

当对 STANDBY IMMEDIATE ATA 命令发送 RW_REG 命令时，主控制器使用下面的设置：

- cmd 寄存器设置：data_expected 比特设为 0
- cmdarg 寄存器设置：
 - 比特 [31] 设为 1
 - 比特 [7:2] 设为 4
 - 其它所有比特设为 0

- 任务文件设置：
 - ATA 任务文件的 Command 域设为 0xE0
 - 任务文件的 Reserved 域设为 0
- bytcnt 寄存器和 blksiz 寄存器的 block_size 域：设为 16

FLUSH CACHE EXT 命令

对于那些缓存写入数据的卡器件，FLUSH CACHE EXT 命令确保缓存的数据写入到卡介质。对于不缓存写入数据的卡器件，FLUSH CACHE EXT 命令返回一个成功状态。ATA 命令不需要数据传递 (RW_BLK)。

主机对 ATA 命令发出一个 RW_REG 命令，通过 SD/SDIO CMD39 或者 RW_REG 命令检索状态。此 ATA 命令能够有错误状态，在此情况下，ATA 任务文件的所有域（除了状态域）都是有效的。

当对 STANDBY IMMEDIATE ATA 命令发送 RW_REG 命令时，主控制器使用下面的设置：

- cmd 寄存器设置：data_expected 比特设为 0
- cmdarg 寄存器设置：
 - 比特 [31] 设为 1
 - 比特 [7:2] 设为 4
 - 所有其它比特设为 0
- 任务文件设置：
 - ATA 任务文件的 Command 域设为 0xEA
 - 任务文件的 Reserved 域设为 0
- bytcnt 寄存器和 blksiz 寄存器的 block_size 域：设为 16

卡读取阈值


当应用需要执行一个单 / 多数据块读命令时，此应用必须使用卡读取阈值域 (cardrdthreshold) 中的相应卡读取阈值来设置 cardthrctl 寄存器，并将 cardrdthren 比特设为 1。在控制器中指定的该额外信息确保了控制器仅在存在等同于 RX FIFO 缓存中的卡读取阈值的空间时才发送一个读命令。相反，这也确保了数据从卡发送过程中卡时钟不会停止。将卡读取阈值设为传递的数据块大小以保证在控制器使能卡时钟前，RX FIFO 缓存中有一个最小数据块大小的空间。

当往返 (round trip) 延迟大于 sdmmc_clk_divided 的一半时，需要卡读取阈值。

卡读取阈值的建议使用指南

1. 在设置 cmd 寄存器用于数据读命令之前必须设置 cardthrctl 寄存器。
2. 数据传递命令执行过程中一定不要设置 cardthrctl 寄存器。
3. cardthrctl 寄存器的 cardrdthreshold 域必须设成至少一个单 / 多数据块传递的数据块大小。cardrdthreshold 域大于或等于读传递的数据块大小确保了数据块传递过程中卡时钟不会停止。
4. 如果往返延迟大于卡时钟周期的一半，那么卡读取阈值必须使能，并且必须根据指南 3 设置卡阈值以保证数据块传递过程中卡时钟不会停止。

5. 如果 `cardrdthreshold` 域设成少于传递的数据块大小，那么主机必须确保在读传递过程中接收 FIFO 缓存从不上溢。上溢能导致控制器的卡时钟停止。控制器不能保证卡时钟在读传递期间不停止。

 如果 `cardthrctl` 寄存器的 `cardrdthreshold` 域和 `fifoth` 寄存器的 `rx_wmark` 和 `dw_dma_multiple_transaction_size` 域都被错误地设置，那么在没有控制器生成的中断情况下卡时钟可能无限期停止。

卡读取阈值编程序列

大多数卡（例如：SDHC 或者 SDXC）都支持卡中指定的数据块大小，或者支持固定为 512 字节的数据块大小。对于支持部分数据块读取（卡器件中的 CSD 寄存器中的 `READ_BL_PARTIAL` 设为 1）的 SDIO，MMC 和标准性能 SD 卡，数据块大小是可变的，可由应用选择。

要有效地使用卡读取阈值功能，并保证卡时钟不会因为从卡中读取数据块过程中 FIFO Full 情况而停止，需要遵循下面的步骤：

1. 选择四字节倍数的数据块大小。
2. 使能卡读取阈值功能。仅在指定传递的数据块大小小于或等于 FIFO 缓存的总深度时才能够使能卡读取阈值功能：
 $(\text{数据块大小} / 4) \leq 1024$
3. 选择卡读取阈值：
 - 若 $(\text{数据块大小} / 4) \geq 512$ ，则选择 `cardrdthreshold`，从而 $\text{cardrdthreshold} \leq (\text{数据块大小} / 4)$ （以字节为单位）
 - 若 $(\text{数据块大小} / 4) < 512$ ，则选择 `cardrdthreshold`，从而 $\text{cardrdthreshold} = (\text{数据块大小} / 4)$ （以字节为单位）
4. 将 `fifoth` 寄存器中的 `dw_dma_multiple_transaction_size` 域设为形成一个 DMA 传输的传递数量。例如，`size = 1` 表示移动 4 个字节。`size` 的可能值是 1, 4, 8, 16, 32, 64, 128 和 256 个传递。选择 `size` 以便值 $(\text{数据块大小} / 4)$ 被 `size` 均匀整除。
5. 将 `fifoth` 寄存器中的 `rx_wmark` 域设为 `size - 1`。

例如，如果 Block Size 是 512 字节，那么 `dw_dma_multiple_transaction_size` 和 `rx_wmark` 的合法值列在表 11-28 中。

表 11-28. `dw_dma_multiple_transaction_size` 和 `rx_wmark` 的合法值, Block Size = 512

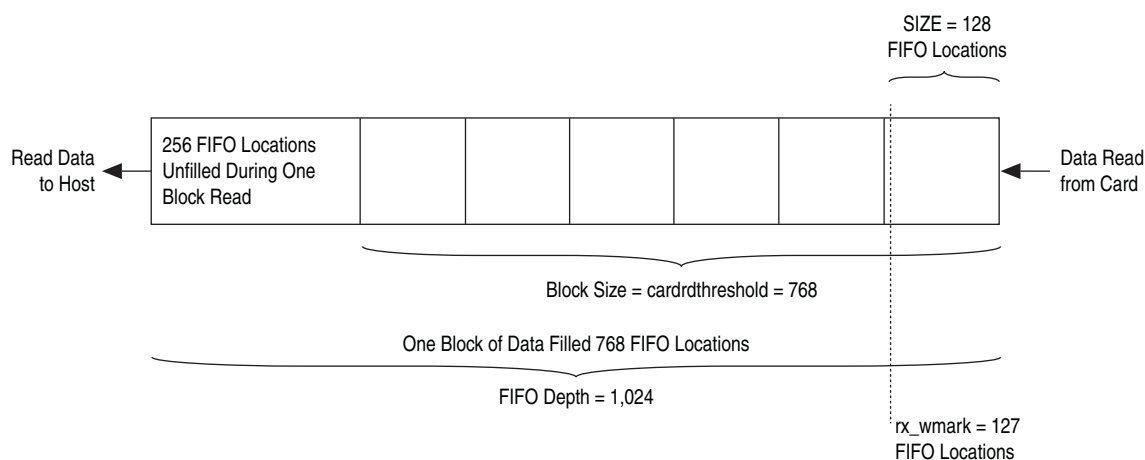
Block Size	<code>dw_dma_multiple_transaction_size</code>	<code>rx_wmark</code>
512	1	0
512	4	3
512	8	7
512	16	15
512	32	31
512	64	63
512	128	127

卡读取阈值编程实例

这一部分显示了如何编程卡读取阈值的实例。

- 选择一个 4 的倍数的数据块大小 (block size) (每个 FIFO 位置的字节数)，并少于 4096 (1024 FIFO 位置)。例如，3072 字节的数据块大小是合法的，因为 $3072 / 4 = 768$ FIFO 位置。
- 对于 DMA 模式，选择 size，使 block size 是 size 的倍数。例如：size = 128，其中 $\text{block size} \% \text{size} = 0$ 。
- 设置 rx_wmark 域 = size - 1。例如：rx_wmark 域 = 128 - 1 = 127。
- 因为数据块大小 $> \frac{1}{2}$ FifoDepth，所以将 cardrdthreshold 域设为数据块大小。例如：cardrdthreshold 域 = 3072 字节。

图 11 - 12. Card Read Threshold 设为 768 时的 FIFO 缓存内容



中断与错误处理


这一部分介绍了如何使用中断来处理错误。上电或复位时，中断被禁用 (ctrl 寄存器中的 int_enable 比特设为 0)，并且所有的中断被屏蔽 (intmask 寄存器默认为 0)。控制器错误处理包括下面类型的错误：

- 响应和数据超时错误 — 对于响应超时，主软件能够重试命令。对于数据超时，由于控制器还没有接收到卡的数据起始比特，因此软件能够再次重试整个数据传递，或者从指定数据块开始重试。通过稍后读取 tcbcnt 寄存器的内容，软件能够决定需要复制（读取）的剩余字节数。
- 响应错误 — 响应接收期间接收到一个错误时，设为 1。如果接收到的响应是无效的，那么软件能够重试命令。
- 数据错误 — 当数据接收错误发生时设为 1。数据接收错误的实例：
 - 数据 CRC
 - 没有找到起始比特
 - 没有找到结束比特

这些错误能够出现在任何数据块上。接收一个错误时，软件能够发出一个 SD/SDIO STOP 或者 SEND_IF_COND 命令，并对整个数据或者部分数据重试命令。

- 硬件锁定错误—当控制器不能加载一个由软件发出的命令时设为 1。当软件将 cmd 寄存器中的 start_cmd 比特设为 1 时，控制器尝试加载命令。如果命令缓存已经包含一个命令，那么会出现此错误，并且丢弃新的命令，从而需要软件重新加载命令。
 - FIFO 缓存 underrun/overrun 错误—如果 FIFO 缓存满了，软件尝试写入数据到 FIFO 缓存，那么会出现 overrun 错误。相反，如果 FIFO 缓存是空的，软件尝试从 FIFO 缓存中读取数据，那么出现 underrun 错误。对 FIFO 缓存读取或写入数据前，软件必须读取 status 寄存器中的 FIFO 缓存空比特 (fifo_empty) 或者 FIFO 缓存满比特 (fifo_full)。
 - 主机超时导致的数据欠缺 (data starvation)—当软件服务 FIFO 缓存不足够快而无法赶上控制器时会出现这一情况。此情况下进行读传递时，软件必须从 FIFO 缓存中读取数据，这样对更多的数据接收创造空间。当进行发送操作时，软件必须写入数据以填充 FIFO 缓存，以便控制器能够将数据写入卡中。
 - CE-ATA 错误
 - 命令上的 CRC 错误—如果在一个命令上检测到 CRC 错误，那么 CE-ATA 卡器件不发送响应，并且控制器会存在响应超时。不通知 ATA 层出现了 MMC 传输层错误。
 - 写操作—卡器件已知的任何 MMC 传输层错误都会导致未执行的 ATA 命令被终止。ATA 状态寄存器中的 ERR 比特被设置，相应的错误代码发送到 ATA 卡器件上的 Error Register (Error)。

如果 CE-ATA 卡的器件中断比特 (ATA 控制寄存器中的 nIEN 比特) 设为 0，那么 CCS 被发送到主机。

如果器件中断比特设为 1，那么卡器件完成整个数据单元数量（如果主控制器没有终止正在进行的传递）。
-  多数据块传递期间，如果从卡器件接收到一个负 CRC 状态，那么通过将 rintsts 寄存器中的 dcrc 比特设为 1，数据通路发送一个数据 CRC 错误信号到 BIU。然后，它继续进行数据发送直到发送全部字节。
- 读操作—如果主控制器检测到 MMC 传输层错误，那么主机会完成具有错误状态 ATA 命令。主机控制器能够发出 CCSD 命令，后面跟着 STOP_TRANSMISSION (CMD12) 命令以终止读传递。主机也能够传递整个数据单元数字节，而不终止数据传递。

eMMC 和 MMC 的引导操作

这一部分介绍了如何对 eMMC 和 MMC 引导操作设置控制器。

通过下拉 CMD 线的引导操作

控制器能够通过下拉 CMD 线从 MMC4.3, MMC4.4 和 MMC4.41 卡引导。

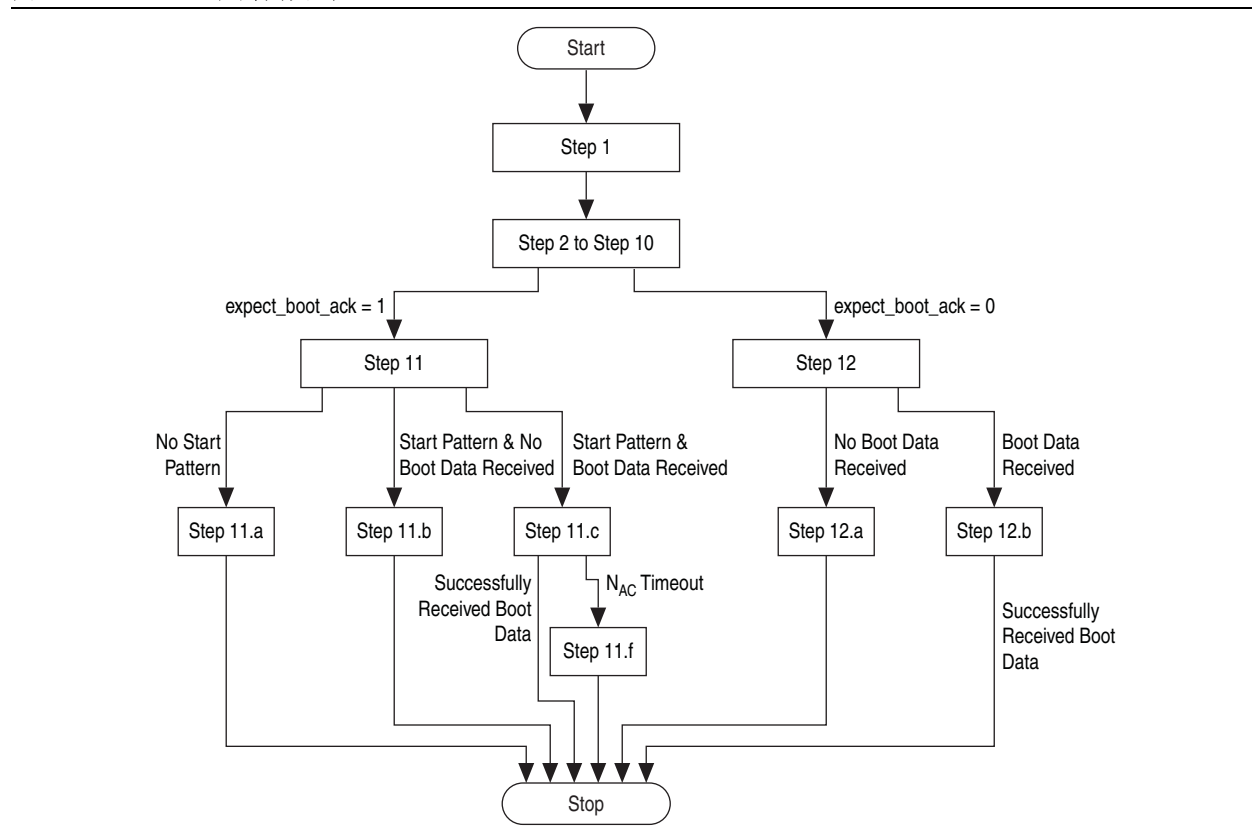
 关于此引导方法的详细信息，请参考第 11 - 67 页的“参考文献”中引用的以下规范：

- JEDEC Standard No. 84-A441
- JEDEC Standard No. 84-A44
- JEDEC Standard No. JESD84-A43

eMMC 卡器件的引导操作

图 11-13 显示了 eMMC 卡器件的详细引导步骤。

图 11-13. eMMC 引导操作流程




1. 软件驱动程序执行下面的检查：

- eMMC 卡器件是否支持引导操作（eMMC 卡的 EXT_CSD 寄存器中的 BOOT_PARTITION_ENABLE 比特设为 1）。
- 在引导过程中使用的 EXT_CSD 寄存器中的 BOOT_SIZE_MULT 和 BOOT_BUS_WIDTH 值。

2. 软件设置以下比特：

- 通过将 intmask 寄存器中的相应比特设为 0 来对中断设置屏蔽。
- 将 ctrl 寄存器中的全局 int_enable 比特设为 1。ctrl 寄存器中的其它比特必须设为 0。

 Altera 建议在设置 int_enable 比特之前将 0xFFFFFFFF 写入到 rintsts 和 idsts 寄存器以清除所有暂挂中断（pending interrupt）。对于内部 DMA 控制器模式，软件驱动程序需要对 idinten 寄存器中的所有相关域去屏蔽。

3. 如果软件驱动程序需要使用内部 DMA 控制器来传递接收到的引导数据，那么它必须执行下面的步骤：
 - 设置第 11 - 44 页的“内部 DMA 控制器发送序列”和第 11 - 44 页的“内部 DMA 控制器接收序列”中所述的描述符。
 - 将 ctrl 寄存器中的 use_internal_dmac 比特设为 1。
4. 使用 clkdiv 寄存器将卡器件频率设为 400 kHz。关于更多信息，请参考第 11 - 34 页的“时钟设置 (Clock Setup)”。
5. 将 tmout 寄存器的 data_timeout 域设成卡器件的总访问时间 N_{AC} 。
6. 将 blksiz 寄存器设成 0x200 (512 字节)。
7. 将 bytcnt 寄存器设成 128 KB 的倍数，由卡器件中的 BOOT_SIZE_MULT 值表明。
8. 设置 fifoth 寄存器中的 rx_wmark 域。通常情况下，阈值可设成 512，这是 FIFO 缓存深度的一半。
9. 设置 cmd 寄存器中的下面域：
 - 通过设置 start_cmd = 1 来启动命令
 - 使能引导 (enable_boot) = 1
 - 期望引导接收确认 (expect_boot_ack):
 - 如果期望来自卡器件的 start-acknowledge 方式，那么将 expect_boot_ack 设为 1。
 - 如果不期望来自卡器件的 start-acknowledge 方式，那么将 expect_boot_ack 设为 0。
 - 卡编号 (card_number) = 0
 - data_expected = 1
 - 将 cmd 寄存器的剩余比特设为 0
10. 如果不期望来自卡器件的 start-acknowledge 方式 (expect_boot_ack 设为 0)，那么跳到步骤 12。

11. 该步骤处理需要 start-acknowledge 方式的情况（在步骤 9 中 expect_boot_ack 设为 1）。

- a. 如果在启动命令的 50 ms 之内没有从控制器接收到 Boot ACK Received 中断（步骤 9），那么软件驱动程序必须设置下面的 cmd 寄存器域：

- start_cmd = 1
- 禁止引导 (disable_boot) = 1
- card_number = 0
- 所有其它域 = 0

控制器置低卡接口的 CMD 管脚后生成一个 Command Done 中断。

如果内部 DMA 控制器模式用于引导过程，那么 Boot ACK Received 超时后控制器将执行下面的步骤：

- DMA 描述符关闭。
- idsts 寄存器中的 ces 比特被设置，表明 Boot ACK Received 超时。
- idsts 寄存器中的 ri 比特没被设置。

- b. 如果接收到 Boot ACK Received 中断，那么软件必须通过写入 1 到 idsts 寄存器中的 ces 比特来清除此中断。

在 Boot ACK Received 中断的 0.95 秒之内，必须从控制器接收到 Boot Data Start 中断，否则软件驱动程序必须写入下面的 cmd 寄存器域：

- start_cmd = 1
- disable_boot = 1
- card_number = 0
- All other fields = 0

控制器置低卡接口的 CMD 管脚后生成一个 Command Done 中断。

如果内部 DMA 控制器模式用于引导过程，那么 Boot ACK Received 超时后控制器将执行下面的步骤：

- DMA 描述符关闭。
- idsts 寄存器中的 ces 比特被设置，表明 Boot ACK Received 超时。
- idsts 寄存器中的 ri 比特没被设置。

- c. 如果接收到 Boot Data Start 中断，那么它表明正在接收来自卡器件的引导数据。当 DMA 引擎没在内部 DMA 控制器模式时，软件驱动程序可以根据 rintsts 寄存器的 rxdr 中断比特来启动对控制器的数据读操作。

在内部 DMA 控制器模式中，一旦达到在 fifoth 寄存器的 rx_wmark 域中设置的级别，DMA 引擎就开始从 FIFO 缓存到系统存储器传递数据。

在成功引导数据传递的结尾生成下面中断：

- rintsts 寄存器中的 cmd 比特和 dto 比特
- idsts 寄存器中的 ri 比特，仅在内部 DMA 控制器模式中

- d. 如果一个错误出现在引导 ACK 码型 (0b010) 中，或者出现 EBE：

- 通过上拉 CMD 线，控制器自动终止引导过程

- 控制器生成一个 Command Done 中断
 - 控制器不生成 Boot ACK Received 中断
 - 应用程序终止引导传递
- e. 在内部 DMA 控制器模式中：
- 如果软件驱动程序创建的描述符多于接收引导数据所需的，那么多余的描述符不由控制器关闭。在描述符关闭前，软件不能重用它们。
 - 如果软件驱动程序创建的描述符少于接收引导数据所需的，那么控制器会生成一个 Descriptor Unavailable 中断，并且不继续传递任何数据到系统存储器。
- f. 如果数据块传递之间没有违反 N_{AC} ，那么 DRT0 中断会被置位。此外，如果存在关于起始比特或者结束比特的错误，那么也会生成 SBE 或者 EBE 中断。

eMMC 卡器件的引导操作完成。不要执行剩下的步骤 (12)。

12. 该步骤处理需要 start-acknowledge 方式的情况（在步骤 9 中 expect_boot_ack 设为 1）。

- a. 如果在启动命令的 1 秒钟之内没有从控制器接收到 Boot Data Start 中断（步骤 9），那么软件驱动程序必须写入下面的 cmd 寄存器域：

- start_cmd = 1
- disable_boot = 1
- card_number = 0
- 所有其它域 = 0

置低卡的 CMD 线后，控制器生成一个 Command Done 中断。在内部 DMA 控制器模式中，描述符关闭，idsts 寄存器中的 ces 比特设为 1，表明 Boot Data Start 超时。

- b. 如果接收到 Boot Data Start 中断，那么它表明正在接收来自卡器件的引导数据。当 DMA 引擎没在内部 DMA 控制器模式时，软件驱动程序可以根据 rintsts 寄存器的 rxdr 中断比特来启动对控制器的数据读操作。

在内部 DMA 控制器模式中，一旦达到在 fifoth 寄存器的 rx_wmark 域中设置的级别，DMA 引擎就开始从 FIFO 缓存到系统存储器传递数据。

在成功引导数据传递的结尾生成下面中断：

- rintsts 寄存器中的 cmd 比特和 dto 比特
- idsts 寄存器中的 ri 比特，仅在内部 DMA 控制器模式中

- c. 在内部 DMA 控制器模式中：

- 如果软件驱动程序创建的描述符多于接收引导数据所需的，那么多余的描述符不由控制器关闭。
- 如果软件驱动程序创建的描述符少于接收引导数据所需的，那么控制器会生成一个 Descriptor Unavailable 中断，并且不继续传递任何数据到系统存储器。

eMMC 卡器件的引导操作完成。

可移除 MMC4.3, MMC4.4 and MMC4.41 卡的引导操作

可移除 MMC4.3, MMC4.4 和 MMC4.41 卡不同于 eMMC— 控制器不知道这些卡插电后是否支持操作的引导模式。因此，控制器必须：


1. 发现时，第一次可能发现 MMC4.0/4.1/4.2 卡
2. 了解卡特征
3. 决定是否执行引导操作

对于这些卡，软件驱动程序必须执行下面的步骤：

1. 发现卡，如第 11 - 31 页的“枚举卡堆栈 (Enumerated Card Stack)”所描述。
2. 读取卡的 EXT_CSD 寄存器，并检验下面的域：

- BOOT_PARTITION_ENABLE
- BOOT_SIZE_MULT
- BOOT_INFO

3. 如果必要，软件能够修改卡中的引导信息。

 关于更多信息，请参考第 11 - 67 页的“参考文献”中引用的以下规范中的“Access to Boot Partition”：

- JEDEC Standard No. 84-A441
 - JEDEC Standard No. 84-A44
 - JEDEC Standard No. JESD84-A43
4. 如果主处理器需要在下一个上电周期执行引导操作，那么它能够使用 SWITCH_FUNC 命令来修改 EXT_CSD 寄存器内容。
 5. 此步骤完成后，软件驱动程序必须通过写入 pwren 寄存器对卡断电。
 6. 从这里开始，使用第 11 - 62 页的“eMMC 卡器件的可选引导操作”中的相同步骤。

可选引导操作

可选引导操作不同于之前的引导操作 — 软件使用 SD/SDIO GO_IDLE_STATE 命令来引导卡，而不是拉低卡的 CMD 线。只有在 BOOT_INFO 寄存器中的 bit 0 设为 1 时才能执行可选引导操作。BOOT_INFO 位于 EXT_CSD 寄存器中的偏移 228。

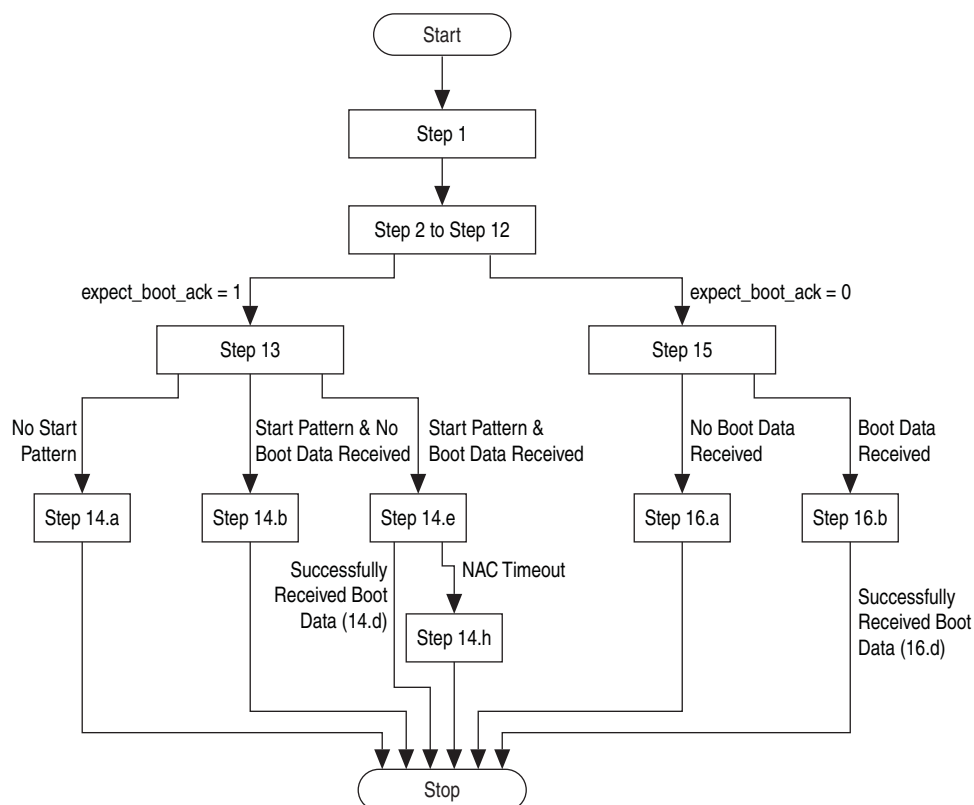
 关于可选引导操作的详细信息，请参考第 11 - 67 页的“参考文献”中引用的以下规范：

- JEDEC Standard No. 84-A441
- JEDEC Standard No. 84-A44
- JEDEC Standard No. JESD84-A43

eMMC 卡器件的可选引导操作

图 11-14 显示了执行 eMMC 卡器件的可选引导操作的步骤。详细的步骤在下面的流程图中有详细描述。

图 11-14. eMMC 可选引导操作的流程



1. 软件驱动程序检查：

- eMMC 卡器件是否支持可选引导操作（eMMC 卡中的 BOOT_INFO 比特设为 1）。
- 引导期间使用的卡器件中的 BOOT_SIZE_MULT 和 BOOT_BUS_WIDTH 值。

2. 软件设置下面的比特：

- 通过置位 intmask 寄存器相应的比特为 0 来对中断设置掩码。
- 将 ctrl 寄存器的 int_enable 比特设为 1。ctrl 寄存器中的其它比特必须设为 0。



Altera 建议在设置 int_enable 比特之前将 0xFFFFFFFF 写入到 rintsts 寄存器和 idsts 寄存器中来清除任何暂挂的中断。对于内部 DMA 控制器模式，软件驱动程序需要对 idinten 寄存器中的所有相关域去屏蔽。

3. 如果软件驱动程序需要使用内部 DMA 控制器来传递接收到的引导数据，那么它必须执行下面的步骤：
 - 设置第 11 - 44 页的“内部 DMA 控制器发送序列”和第 11 - 44 页的“内部 DMA 控制器接收序列”中所述的描述符。
 - 将 ctrl 寄存器中的使用内部 DMCA 比特 (use_internal_dmac) 设为 1。
4. 使用 clkdiv 寄存器将卡器件频率设为 400 kHz。关于更多信息，请参考第 11 - 34 页的“时钟设置 (Clock Setup)”。要确保卡时钟在运行。
5. 等待一段时间，这样可以确保至少 74 个卡时钟周期已经出现在卡接口上。
6. 将 tmout 寄存器的 data_timeout 域设成卡器件的总访问时间 N_{AC} 。
7. 将 blksiz 寄存器设成 0x200 (512 字节)。
8. 将 bytcnt 寄存器设成 128 KB 的倍数，由卡器件中的 BOOT_SIZE_MULT 值表明。
9. 设置 fifoth 寄存器中的 rx_wmark 域。通常情况下，阈值可设成 512，这是 FIFO 缓存深度的一半。
10. 将 cmdarg 寄存器设成 0xFFFFFFFFFA。
11. 启动命令，通过设置 cmd 寄存器的下面域：
 - start_cmd = 1
 - enable_boot = 1
 - expect_boot_ack:
 - 如果期望来自卡器件的 start-acknowledge 方式，那么将 expect_boot_ack 设为 1。
 - 如果不期望来自卡器件的 start-acknowledge 方式，那么将 expect_boot_ack 设为 0。
 - card_number = 0
 - data_expected = 1
 - cmd_index = 0
 - 将 cmd 寄存器的剩余比特设为 0。
12. 如果不期望来自卡器件的 start-acknowledge 方式 (expect_boot_ack 设为 0)，那么跳到步骤 15。
13. 等待 Command Done 中断。

14. 该步骤处理需要 start-acknowledge 码型的情况（在步骤 11 中 expect_boot_ack 设为 1）。
- a. 如果在启动命令的 50 ms 之内没有从控制器接收到 Boot ACK Received 中断（步骤 11），那么没有接收到起始码型。软件驱动程序必须终止引导进程，并开始正常发现。
- 如果内部 DMA 控制器模式用于引导过程，那么 Boot ACK Received 超时后控制器将执行下面的步骤：
- DMA 描述符关闭。
 - idsts 寄存器中的 ces 比特设为 1，表明 Boot ACK Received 超时。
 - idsts 寄存器中的 ri 比特没被设置。
- b. 如果接收到 Boot ACK Received 中断，那么软件必须通过写入 1 到 idsts 寄存器中的 ces 比特来清除此中断。
- 在 Boot ACK Received 中断的 0.95 秒之内，必须从控制器接收到 Boot Data Start 中断，否则软件驱动程序必须终止引导进程，并开始正常发现。
- 如果内部 DMA 控制器模式用于引导过程，那么 Boot ACK Received 超时后控制器将执行下面的步骤：
- DMA 描述符关闭。
 - idsts 寄存器中的 ces 比特被设为 1，表明 Boot Data Start 超时。
 - idsts 寄存器中的 ri 比特没被设置。
- c. 如果接收到 Boot Data Start 中断，那么它表明正在接收来自卡器件的引导数据。当 DMA 引擎没在内部 DMA 控制器模式时，软件驱动程序可以根据 rintsts 寄存器的 rxdr 中断比特来启动对控制器的数据读操作。
- 在内部 DMA 控制器模式中，一旦达到在 fifoth 寄存器的 rx_wmark 域中设置的级别，DMA 引擎就开始从 FIFO 缓存到系统存储器传递数据。
- d. 软件驱动程序必须通过指导控制器发送 SD/SDIO GO_IDLE_STATE 命令来终止引导进程：
- 置位 cmdarg 寄存器为 0。
 - 将 cmd 寄存器的 start_cmd 比特设为 1，并将其它所有比特设为 0。
- e. 在成功引导数据传递的结尾生成下面中断：
- rintsts 寄存器中的 cmd 比特和 dto 比特
 - idsts 寄存器中的 ri 比特，仅在内部 DMA 控制器模式中

- f. 如果一个错误出现在引导 ACK 码型 (0b010) 中，或者出现 EBE：
 - 控制器不生成 Boot ACK Received 中断。
 - 控制器检测到 Boot Data Start，并生成一个 Boot Data Start 中断。
 - 控制器继续接收引导数据。
 - 在接收到 Boot Data Start 中断后，应用程序必须终止引导进程。
 - g. 在内部 DMA 控制器模式中：
 - 如果软件驱动程序创建的描述符多于接收引导数据所需的，那么多于的描述符不由控制器关闭。
 - 如果软件驱动程序创建的描述符少于接收引导数据所需的，那么控制器会生成一个 Descriptor Unavailable 中断，并且不继续传递任何数据到系统存储器。
 - h. 如果数据块传递之间没有违反 N_{AC} ，那么 DRT0 中断会被置位。此外，如果存在关于起始比特或者结束比特的错误，那么也会生成 SBE 或者 EBE 中断。
- eMMC 卡器件的可选引导操作完成。不要执行剩下的步骤 (15 和 16)。

15. 等待 Command Done 中断。

16. 该步骤处理不需要 start-acknowledge 码型的情况 (在步骤 11 中 expect_boot_ack 设为 0)。

- a. 如果在启动命令的 1 秒钟之内没有从控制器接收到 Boot Data Start 中断 (步骤 11)，那么软件驱动程序必须终止引导进程，并开始正常发现。

在内部 DMA 控制器模式中：

- DMA 描述符关闭。
 - idsts 寄存器中的 ces 比特设为 1，表明 Boot Data Start 超时。
 - idsts 寄存器的 ri 比特没有设置。
- b. 如果接收到 Boot Data Start 中断，那么它表明正在接收来自卡器件的引导数据。当 DMA 引擎没在内部 DMA 控制器模式时，软件驱动程序可以根据 rintsts 寄存器的 rxdr 中断比特来启动对控制器的数据读操作。

在内部 DMA 控制器模式中，一旦达到在 fifoth 寄存器的 rx_wmark 域中设置的级别，DMA 引擎就开始从 FIFO 缓存到系统存储器传递数据。

- c. 软件驱动程序必须通过指导控制器发送 SD/SDIO GO_IDLE_STATE (CMD0) 命令来终止引导进程：
 - 置位 cmdarg 寄存器为 0。
 - 将 cmd 寄存器中的 start_cmd 比特设为 1，所有其它比特设为 0。
- d. 在成功引导数据传递的结尾生成下面中断：
 - 在 rintsts 寄存器中的 cmd 比特和 dto 比特
 - idsts 寄存器中的 ri 比特，仅在内部 DMA 控制器模式中


- e. 在内部 DMA 控制器模式中：
- 如果软件驱动程序创建的描述符多于接收引导数据所需的，那么多于的描述符不由控制器关闭。
 - 如果软件驱动程序创建的描述符少于接收引导数据所需的，那么控制器会生成一个 Descriptor Unavailable 中断，并且不继续传递任何数据到系统存储器。
- eMMC 卡器件的可选引导操作完成。


MMC4.3 卡的可选引导操作

可移除 MMC4.3 卡不同于 eMMC— 控制器不知道这些卡是否支持操作的引导模式。因此，控制器必须：

1. 发现这些卡，因为可能第一次发现 MMC4.0/4.1/4.2 卡
2. 了解卡特征
3. 决定是否执行引导操作

对于这些卡，软件驱动程序必须执行下面的步骤：


1. 发现卡，如第 11 - 31 页的“枚举卡堆栈 (Enumerated Card Stack)”所描述。
 2. 读取 MMC 卡的 EXT_CSD 寄存器，并检验下面的域：
 - BOOT_PARTITION_ENABLE
 - BOOT_SIZE_MULT
 - BOOT_INFO
-  关于更多信息，请参考第 11 - 67 页的“参考文献”中引用的 JEDEC Standard No. JESD84-A43 中的“Access to Boot Partition”。
3. 如果主处理器需要在下一个上电周期执行引导操作，那么它能够使用 SWITCH_FUNC 命令来修改 MMC 卡器件中的 EXT_CSD 寄存器内容。
 4. 此步骤完成后，软件驱动程序必须通过写入 pwren 寄存器对卡断电。
 5. 从这里开始，使用第 11 - 62 页的“eMMC 卡器件的可选引导操作”中的相同步骤。

 如果 EBE 在终止情况下生成，那么可以将其忽略。

如果出现引导应答错误，那么接收的引导应答中断超时。

在内部 DMA 控制器模式中，应用程序需要依赖描述符关闭中断 (descriptor close interrupt)，而不是数据完成中断 (data done interrupt)。

SD/MMC 控制器地址映射与寄存器定义

 地址映射和寄存器定义位于本卷中的 [hps.html](#) 文件中。点击链接打开此文件。

点击下面链接查看模块描述和基地址：

■ sdmmc

然后点击寄存器名称来查看寄存器和域描述。寄存器地址是相对于每个模块实例基地址的偏移。

 所有模块的基地址也列在 *Cyclone V Device Handbook* 卷 3 中的 *Introduction to the Hard Processor System* 章节中。

参考文献

 下面的业界规范提供了关于 SD/MMC 控制器实现的标准和协议的详细信息。

以下规范可以从 JEDEC 网站 (www.jedec.org) 获得：

- JEDEC Standard No. 84-A441—*Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, Sleep Modes, Dual Data Rate, Multiple Partitions Supports, Security Enhancement, Background Operation and High Priority Interrupt (MMCA, 4.41)*
- JEDEC Standard No. 84-A44—*Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, Sleep Modes, Dual Data Rate, Multiple Partitions Supports and Security Enhancement (MMCA, 4.4)*
- JEDEC Standard No. JESD84-A43—*Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, and Sleep Modes (MMCA, 4.3)*

以下规范可以从 SD Association 网站 (www.sdcard.org) 获得：

- *Physical Layer Simplified Specification, Version 3.01—SD Specifications Part 1 Physical Layer Simplified Specification Version 3.01*
- *SDIO Simplified Specification Version 2.00—SD Specifications Part E1 SDIO Simplified Specification Version 2.00*

文档修订历史

表 11 - 29 显示了本文档的修订历史。

表 11 - 29. 文档修订历史

日期	版本	修订内容
2012 年 11 月	1.1	<ul style="list-style-type: none">■ 添加了编程模型部分。■ 重组了编程信息。■ 添加了关于 ECC 的信息。■ 添加了管脚列。■ 更新了时钟部分。
2012 年 1 月	1.0	首次发布

