# MMC 子系统简介

周春华

July 19, 2011

# SD 卡相关的概念 ---卡的种类

## 从外形尺寸来分

1. 

MMC 卡，$32mm \times 24mm \times 1.4mm$，外形规格众多，贴图只是其中一种规格
2. 

SD 卡，$32mm \times 24mm \times 2.1mm$，要比 MMC 卡厚一点点，但是容量和速度大得却是很多倍
3. 

Micro SD 卡，外形尺寸比 SD 卡小

注意

- SD 卡和 MMC 是不同的，现在很多接插件可以同时支持这两种卡，但有时能使用 MMC 卡的地方未必能插进 SD 卡。
- SD 卡和 MMC 卡，两者在通讯协议上也是不完全相同的。

# SD 卡相关的概念 ---卡的种类

## 从性能来分

1. class 0 卡：此类卡表示没有性能指标，可以无视其性能
2. class 2，class 4，class 6 卡：这些卡都是有性能指标的，分别是大于等于 2MB/sec，4MB/sec，6MB/sec。这些卡的版面上会印有标记。
3. 本文不涉及 sd spec3.0，所以 class10 也不作介绍。



# SD 卡相关的概念 ---卡的种类

## 从容量来分

1. Normal 卡：$MAX_{frq} = 25MHz$，性能为 class0，容量最高达 2G
2. SDHC 卡：$MAX_{frq} = 50MHz$，性能有 class2，class4，class6 三种类型，容量为 2G-32G

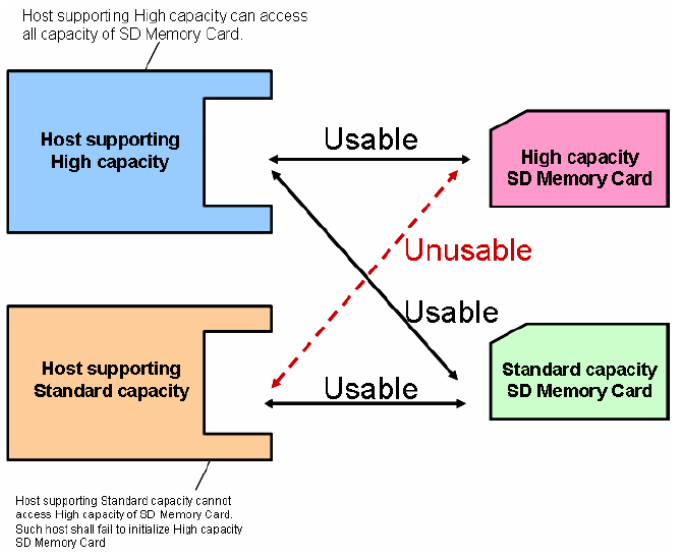
注意

- 这里 HC 的意思不是高速，而是 high capacity。

# SD 卡相关的概念 ---兼容性问题

## 主机兼容的卡

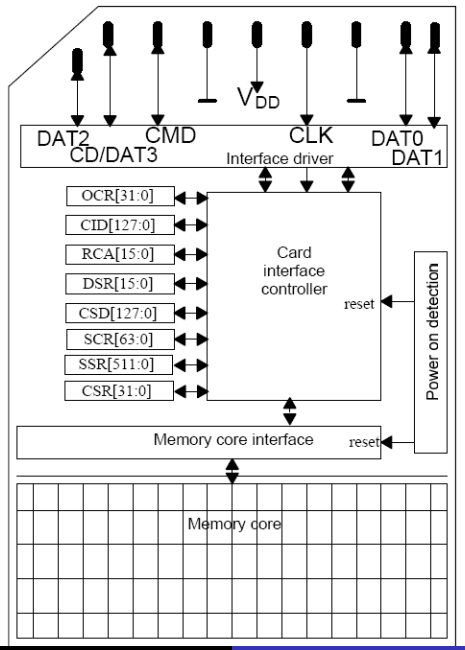
1. 支持 SDHC 的 SD Host Controller，可以读写 Normal 卡以及 SDHC 卡。
2. 不支持 SDHC 的 SD Host Controller，只能读写 Normal 卡。



# SD 卡相关的概念 ---卡的结构

## 卡的结构

1. SD 卡有 9 个引脚，microSD 有 8 个引脚，少了根地线
2. SD 卡内部是有寄存器的，如上图。主机可以通过相关的命令获取寄存器中的值，从而了解卡的一些信息



# 总线定义 ---信号线定义

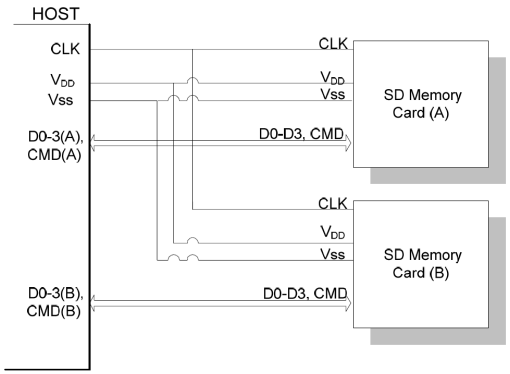
## SD Bus 包含了以下九根信号线

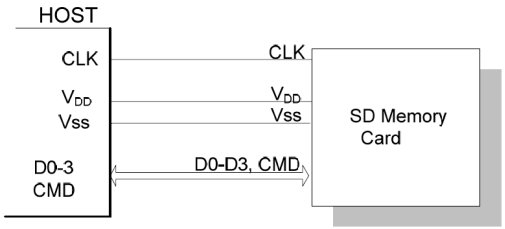
1. CLK：Host 到卡的时钟线
2. CMD：双向的命令和响应信号线
3. DAT0 - DAT3：双向的数据传输线
4. $V_{DD}$，$V_{SS1}$，$V_{SS2}$ 电源和信号地

注意

- 这里说的是 SD Bus 有 9 根线，不是说所有的 SD Card 都必须有 9 个引脚。

# 总线定义 ---总线拓扑




## 注意

- 虽然 SD Bus 可以支持一个 SD Host 可以连接多个卡，但是 Spec 2.0 强烈推荐一个 SD Host 只连接一个 SD 卡。所以接下来的也不会讲述连接多卡的相关内容了，默认的就是 host 和卡是一一对应的。

# 总线协议 ---协议术语

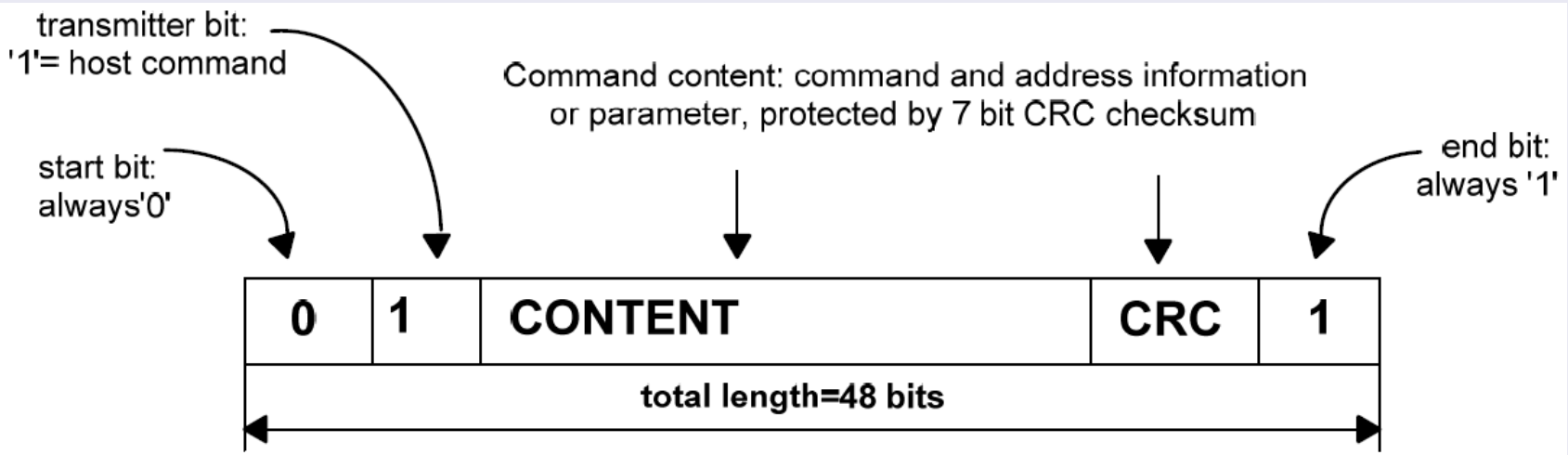
SD 总线上通讯是基于命令和数据流的。

## 命令，响应，数据

1. Command：每一个操作都是以命令开始的，命令是由主机发送给卡的，而且命令总是在 CMD 线上串行传输。
2. Response：响应是作为收到命令后应该返回主机的应答令牌。所有的响应都是串行在 CMD 上传输的。
3. Data：数据可以从卡传向主机，也可以从主机传向卡。所有的数据都是在 Data 线上进行传输的。

# 总线协议 ---命令格式

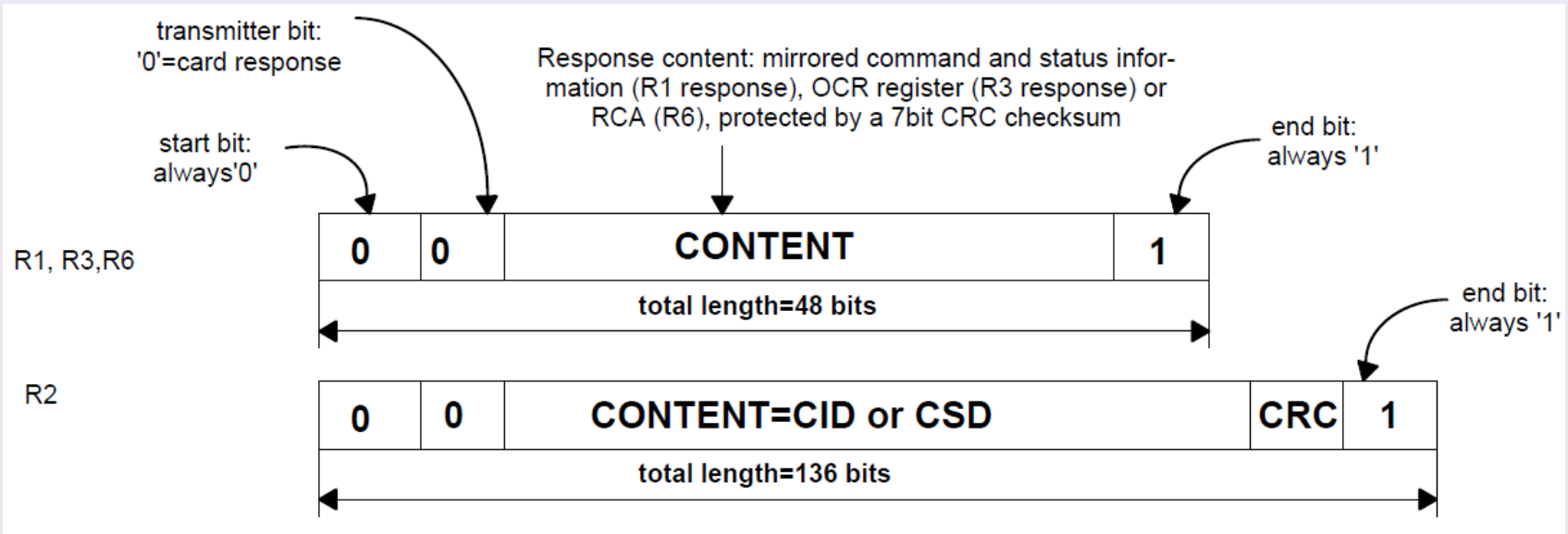
## 命令帧的格式




| 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' |
| Description | start bit | transmission bit | command index | argument | CRC7 | end bit |

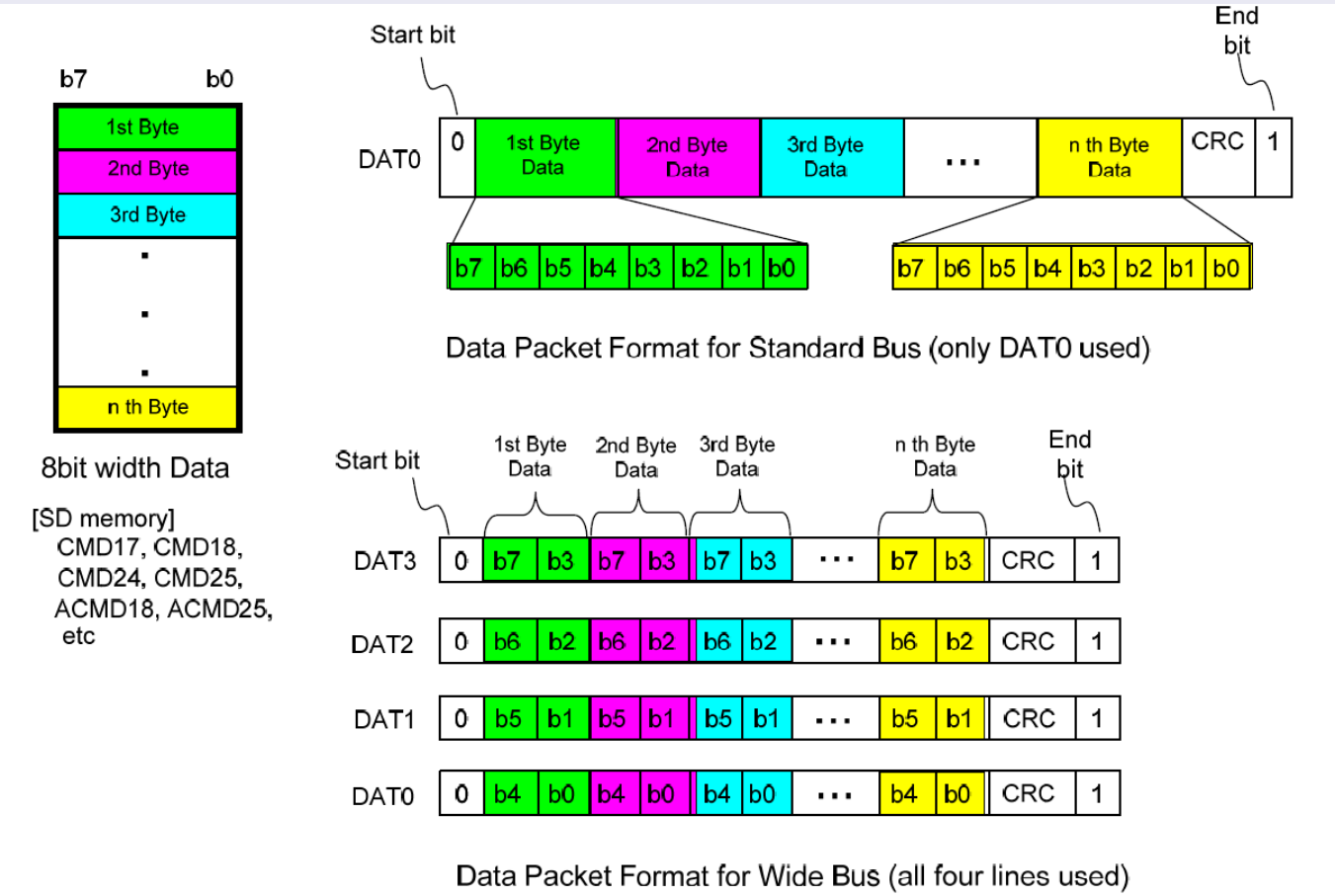
# 总线协议 ---响应格式

## 响应帧的格式



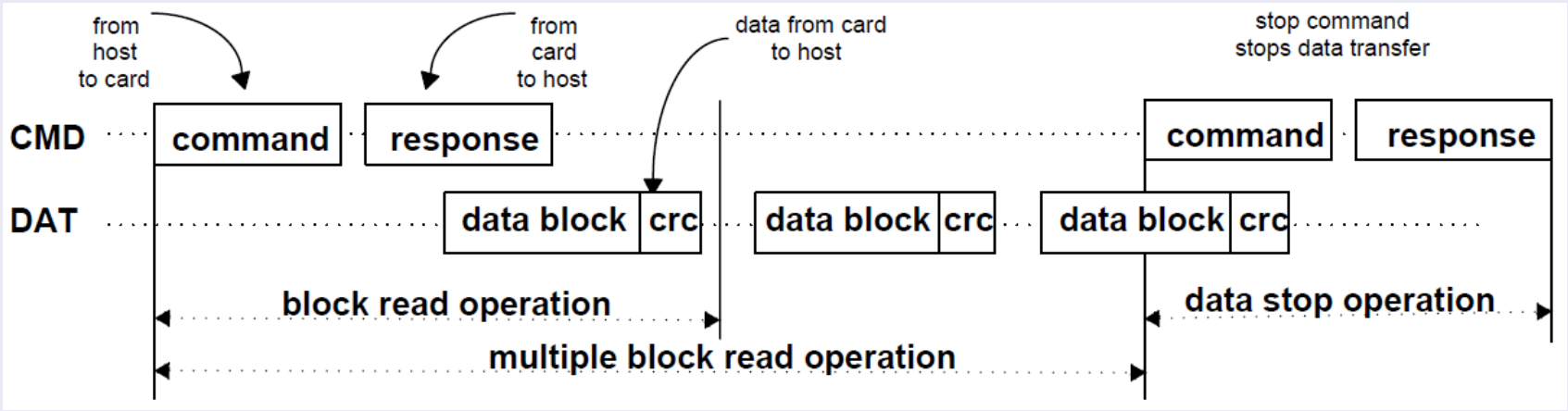
# 总线协议 ---数据格式

## 数据的传输形式



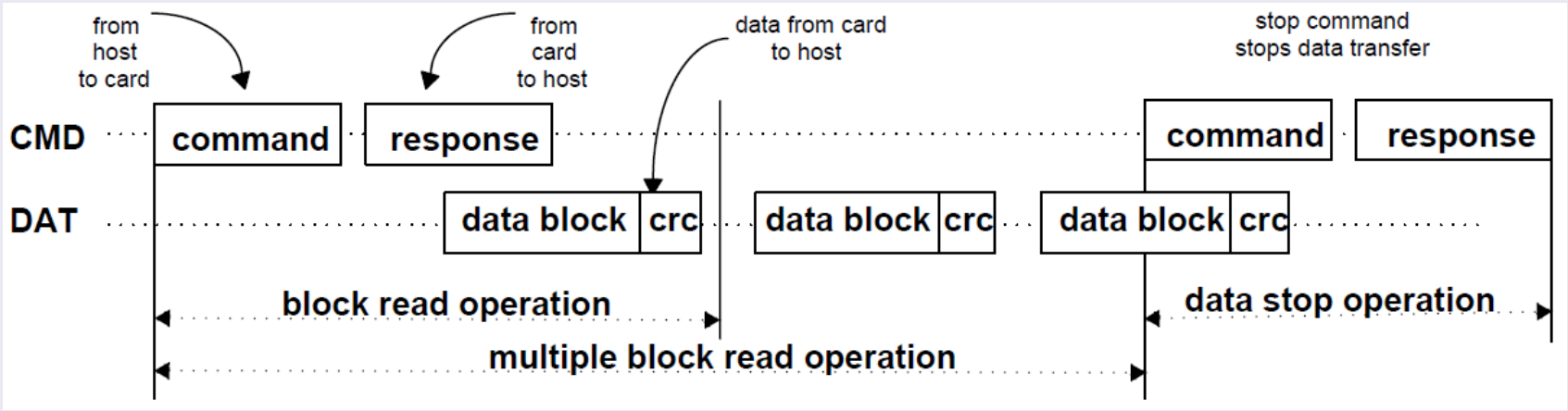
# 总线协议 ---数据传输过程

## 多块连读的过程示意图



# 总线协议 ---数据传输过程

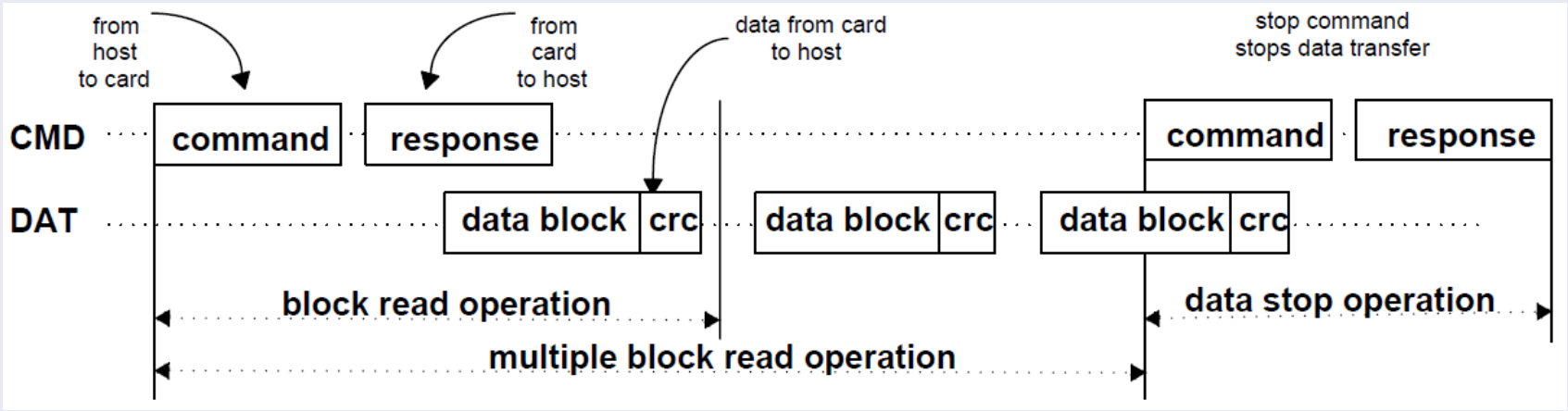
## 多块连读的过程示意图




1. step1： 发送多块连读命令，命令参数指定要读取的起始扇区号

# 总线协议 ---数据传输过程

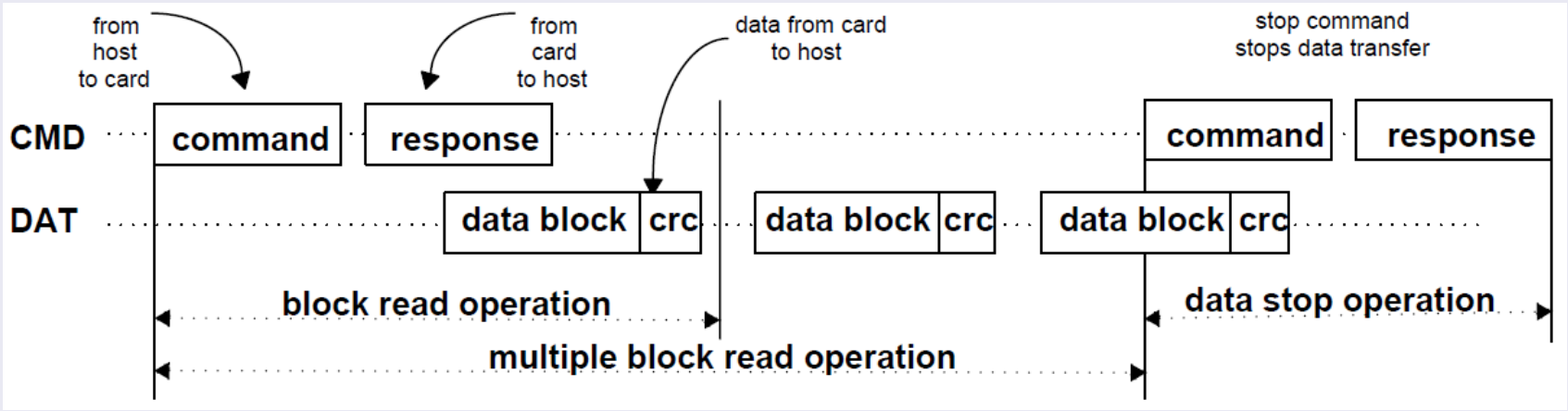
## 多块连读的过程示意图




1. step1： 发送多块连读命令，命令参数指定要读取的起始扇区号
2. step2： 卡收到命令后，开始将数据传输给主机，主机开始接收数据

# 总线协议 ---数据传输过程

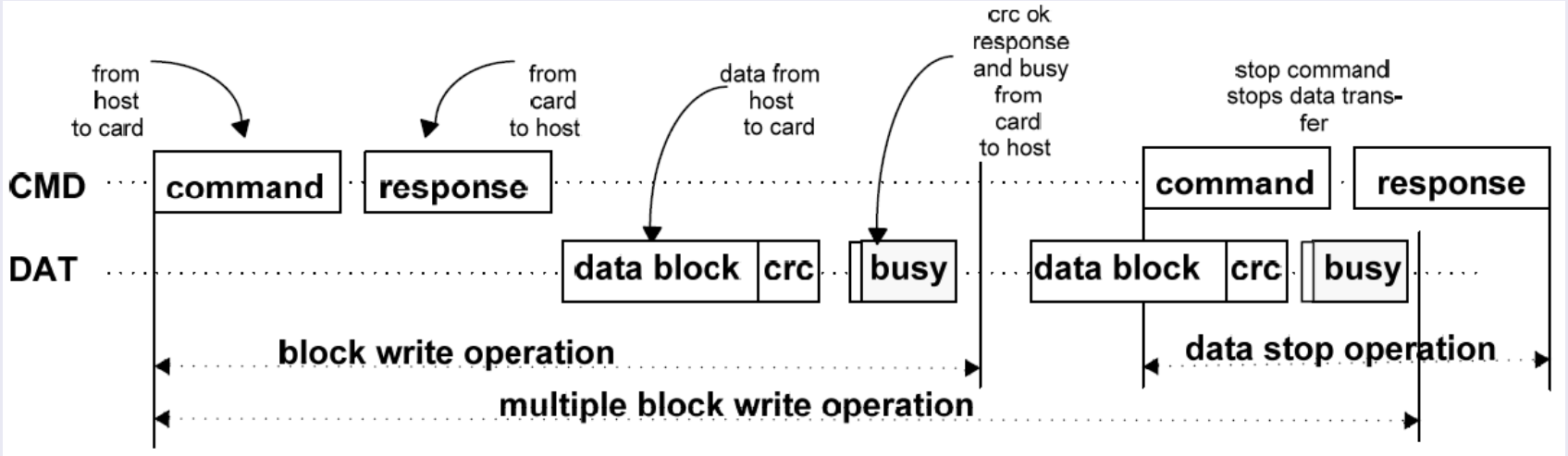
## 多块连读的过程示意图




1. step1： 发送多块连读命令，命令参数指定要读取的起始扇区号
2. step2： 卡收到命令后，开始将数据传输给主机，主机开始接收数据
3. step3： 主机接收到足够的数据后，发送 stop 命令，终止这次数据传输

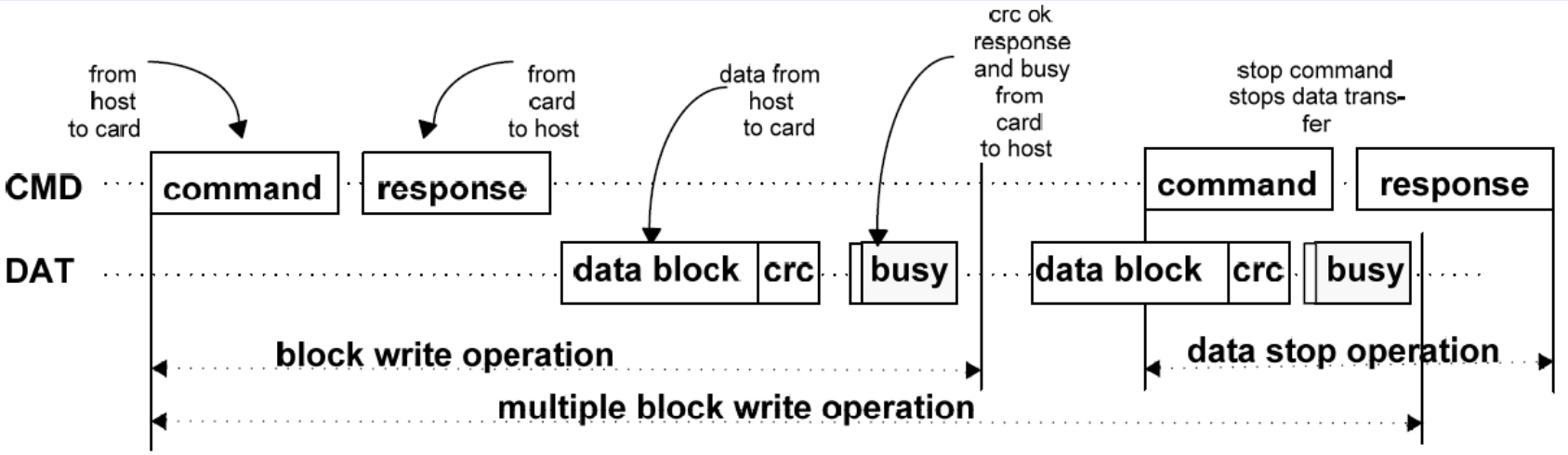
# 总线协议 ---数据传输过程

## 多块连写的过程示意图



# 总线协议 ---数据传输过程

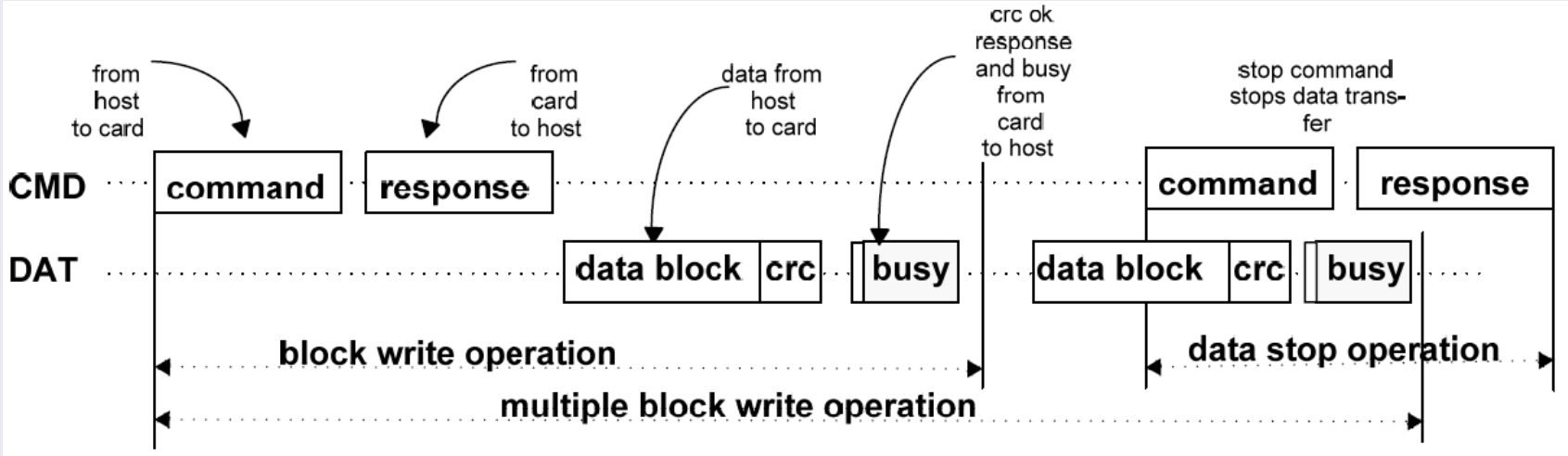
## 多块连写的过程示意图




1. step1： 发送多块连写命令，命令参数指定要写入的起始扇区号

# 总线协议 ---数据传输过程

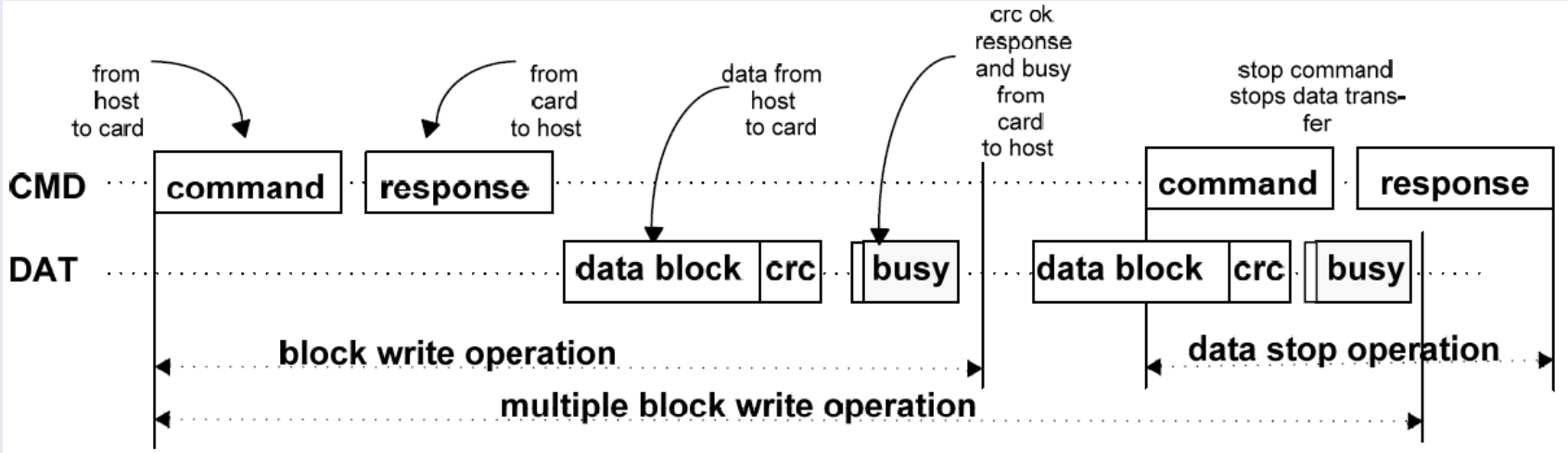
## 多块连写的过程示意图




1. step1： 发送多块连写命令，命令参数指定要写入的起始扇区号
2. step2： 卡收到命令后，准备接收从主机发过来的数据，主机开始接收数据

# 总线协议 ---数据传输过程

## 多块连写的过程示意图




1. step1： 发送多块连写命令，命令参数指定要写入的起始扇区号
2. step2： 卡收到命令后，准备接收从主机发过来的数据，主机开始接收数据
3. step3： 主机发送完数据后，发送 stop 命令，终止这次数据传输

# 总线协议 ---SD 与 MMC 区别

SD 协议规范和 MMC 协议规范绝大部分是相同的，这里只列举几点比较明显的差别：

# 总线协议 ---SD 与 MMC 区别

SD 协议规范和 MMC 协议规范绝大部分是相同的，这里只列举几点比较明显的差别：

1. 初始化卡时，要求卡发回操作条件命令：SD 协议使用的是 CMD8，ACMD41，而 MMC 协议则使用 CMD1。

# 总线协议 ---SD 与 MMC 区别

SD 协议规范和 MMC 协议规范绝大部分是相同的，这里只列举几点比较明显的差别：

1. 初始化卡时，要求卡发回操作条件命令：SD 协议使用的是 CMD8，ACMD41，而 MMC 协议则使用 CMD1。
2. 对于 CMD3，要求卡发回卡相对地址：SD 协议使用的是 R6 作为回应，而 MMC 协议则使用 R1 作为回应。

# 总线协议 ---SD 与 MMC 区别

SD 协议规范和 MMC 协议规范绝大部分是相同的，这里只列举几点比较明显的差别：

1. 初始化卡时，要求卡发回操作条件命令：SD 协议使用的是 CMD8，ACMD41，而 MMC 协议则使用 CMD1。
2. 对于 CMD3，要求卡发回卡相对地址：SD 协议使用的是 R6 作为回应，而 MMC 协议则使用 R1 作为回应。
3. 擦除命令：SD 卡使用 CMD32 设置起始块，使用 CMD33 设置结束块，而 MMC 使用的 CMD35 和 CMD36。

# 总线协议 ---SD 与 MMC 区别

SD 协议规范和 MMC 协议规范绝大部分是相同的，这里只列举几点比较明显的差别：

1. 初始化卡时，要求卡发回操作条件命令：SD 协议使用的是 CMD8，ACMD41，而 MMC 协议则使用 CMD1。
2. 对于 CMD3，要求卡发回卡相对地址：SD 协议使用的是 R6 作为回应，而 MMC 协议则使用 R1 作为回应。
3. 擦除命令：SD 卡使用 CMD32 设置起始块，使用 CMD33 设置结束块，而 MMC 使用的 CMD35 和 CMD36。
4. 总线宽度设置命令：早期版本的 MMC 协议没有设置总线命令，因为默认就是一根数据线。而 SD 卡有 1 位总线模式和 4 位总线模式。

# 子系统中的对象

在 Linux 中，是由 MMC 子系统来对 SD 卡提供支持的。MMC 子系统为 MMC/SD/SDIO host 提供了统一的驱动接口，我们只需要实现其接口，就可以完成一个新 host 驱动的添加了。

在前面简单地描述中，涉及到了几个对象，比如 host，card，command，response，data 等。在子系统中，MMC 子系统的设计者们分别将他们进行抽象一系列的结构体。接下来，就从这些结构体中，体会设计者是如何架构整个子系统的。

# 子系统中的对象 ---mmc_host

MMC/SD/SDIO host 都被抽象成 mmc_host，其主要成员如下：

```
1  struct mmc_host {
2      ... ...
3      const struct mmc_host_ops   *ops;        /*跟具体控制器相关，需要 host 驱动实现*/
4      unsigned int                 f_min;      /*控制器最低工作频率，需要 host 驱动指定*/
5      unsigned int                 f_max;      /*控制器最高工作频率，需要 host 驱动指定*/
6      ... ...
7      unsigned long                caps;       /* Host capabilities 需要 host 驱动指定*/
8  #define MMC_CAP_4_BIT_DATA       (1 << 0)    /* Can the host do 4 bit transfers */
9  #define MMC_CAP_MMC_HIGHSPEED    (1 << 1)    /* Can do MMC high-speed timing */
10 #define MMC_CAP_SD_HIGHSPEED     (1 << 2)    /* Can do SD high-speed timing */
11     ... ...
12 #define MMC_CAP_SPI              (1 << 4)    /* Talks only SPI protocols */
13     ... ...
14 #define MMC_CAP_8_BIT_DATA       (1 << 6)    /* Can the host do 8 bit transfers */
15     ... ...
16     struct mmc_ios               ios;        /* current io bus settings */
17     ... ...
18     struct mmc_card             *card;        /* device attached to this host */
19     ... ...
20     struct delayed_work          detect;
21     const struct mmc_bus_ops    *bus_ops;    /* current bus driver */
22     ... ...
23 }
```

# 子系统中的对象 ---mmc_card

前面 mmc_host 中有一个成员就是指向其所对应的 card/SDIO 设备：mmc_card

```
1  struct mmc_card {
2      struct mmc_host             *host;      /* the host this device belongs to */
3      struct device                dev;       /* the device */
4      unsigned int                 rca;       /* relative card address of device */
5      unsigned int                 type;      /* card type */
6  #define MMC_TYPE_MMC             0          /* MMC card */
7  #define MMC_TYPE_SD              1          /* SD card */
8  #define MMC_TYPE_SDIO            2          /* SDIO card */
9      unsigned int                 state;     /* (our) card state */
10 #define MMC_STATE_PRESENT        (1<<0)     /* present in sysfs */
11 #define MMC_STATE_READONLY       (1<<1)     /* card is read-only */
12 #define MMC_STATE_HIGHSPEED      (1<<2)     /* card is in high speed mode */
13 #define MMC_STATE_BLOCKADDR      (1<<3)     /* card uses block-addressing */
14     ... ...
15     struct mmc_cid               cid;       /* card identification */
16     struct mmc_csd               csd;       /* card specific */
17     ... ...
18 };
```

mmc_card 也有一个成员 mmc_host，指向与其一一对应的 host

# 子系统中的对象 ---mmc_request

MMC 子系统中，需要发送命令，或者读写数据都是通过发一个请求包给 host 驱动，这个请求包就是 mmc_request 了。如果只是发送命令，只需要将 data 以及 stop 成员置为空就可以了。

```
1  struct mmc_request {
2      struct mmc_command          *cmd;
3      struct mmc_data             *data;
4      struct mmc_command          *stop;
5
6      void                        *done_data;     /* completion data */
7      void                       (*done)(struct mmc_request *);/*completion function*/
8  };
```

# 子系统中的对象 ---mmc_command

mmc_command 包含了命令的操作码，命令的参数，host 驱动发送完毕后，还需要将响应值填入了 resp 成员。如果发生错误，则填入 error 成员。

```
1  struct mmc_command {
2      u32                     opcode;
3      u32                     arg;
4      u32                     resp[4];
5      ... ...
6      unsigned int            retries;    /* max number of retries */
7      unsigned int            error;      /* command error */
8
9      struct mmc_data         *data;      /* data segment associated with cmd */
10     struct mmc_request      *mrq;       /* associated request */
11 };
```

# 子系统中的对象 ---mmc_data

mmc_data 结构有数据超时时间，数据块的大小，数据传输的方向，host 驱动完成后，需要填入 error 码，已经传输了 bytes_xfered 字节。

```
1  struct mmc_data {
2      unsigned int                timeout_ns;     /* data timeout (in ns, max 80ms) */
3      unsigned int                timeout_clks;   /* data timeout (in clocks) */
4      unsigned int                blksz;          /* data block size */
5      unsigned int                blocks;         /* number of blocks */
6      unsigned int                error;          /* data error */
7      unsigned int                flags;
8  #define MMC_DATA_WRITE          (1 << 8)
9  #define MMC_DATA_READ           (1 << 9)
10 #define MMC_DATA_STREAM         (1 << 10)
11     unsigned int                bytes_xfered;
12     struct mmc_command          *stop;          /* stop command */
13     struct mmc_request          *mrq;           /* associated request */
14     unsigned int                sg_len;         /* size of scatter list */
15     struct scatterlist          *sg;            /* I/O scatter list */
16 };
```

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

- host/：host 驱动目录，如果添加一个新的 host 驱动，就需要在此目录中增加。host 驱动主要作用有两个：

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

- host/：host 驱动目录，如果添加一个新的 host 驱动，就需要在此目录中增加。host 驱动主要作用有两个：
  1. 具体实现 mmc core 层发下来个各种请求，不仅仅是发送命令，还包括修改时钟，开启关闭电源等。

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

- host/：host 驱动目录，如果添加一个新的 host 驱动，就需要在此目录中增加。host 驱动主要作用有两个：
  1. 具体实现 mmc core 层发下来个各种请求，不仅仅是发送命令，还包括修改时钟，开启关闭电源等。
  2. 探测到卡热插拔消息，通知 mmc core 层

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

- host/：host 驱动目录，如果添加一个新的 host 驱动，就需要在此目录中增加。host 驱动主要作用有两个：
  1. 具体实现 mmc core 层发下来个各种请求，不仅仅是发送命令，还包括修改时钟，开启关闭电源等。
  2. 探测到卡热插拔消息，通知 mmc core 层
- core/：mmc 核心层代码，构建 mmc_bus 虚拟总线，初始化 SD 卡，提供发送读写请求包以及各种命令的接口等。

# 整体架构 ---目录结构

在 driver/mmc 目录下有三个目录：

- host/：host 驱动目录，如果添加一个新的 host 驱动，就需要在此目录中增加。host 驱动主要作用有两个：
  1. 具体实现 mmc core 层发下来个各种请求，不仅仅是发送命令，还包括修改时钟，开启关闭电源等。
  2. 探测到卡热插拔消息，通知 mmc core 层
- core/：mmc 核心层代码，构建 mmc_bus 虚拟总线，初始化 SD 卡，提供发送读写请求包以及各种命令的接口等。
- card/：为 mmc_card 注册或者注销其块设备，初步处理 filesystem 发下来的读写 requst。

# 整体架构 ---mmc bus.c

mmc_bus 虚拟总线维护着挂载在其上面的设备和驱动。所有插进来的 mmc_card 都会被注册到这个 bus 上充当设备；而总线上的驱动就是 card/block.c 文件中注册的 mmc_driver。

```
1  static struct bus_type mmc_bus_type = {
2          .name           = "mmc",
3          .dev_attrs      = mmc_dev_attrs,
4          .match          = mmc_bus_match,
5          .uevent         = mmc_bus_uevent,
6          .probe          = mmc_bus_probe,
7          .remove         = mmc_bus_remove,
8          .suspend        = mmc_bus_suspend,
9          .resume         = mmc_bus_resume,
10         .pm             = MMC_PM_OPS_PTR,
11 };
12
13 static int mmc_bus_match(struct device *dev, struct device_driver *drv)
14 {
15         return 1;
16 }
```

# 整体架构 ---block device

## card 目录

# 整体架构 ---block device

## card 目录

- block.c：在 mmc_bus 上注册 mmc_driver，这个驱动是用来匹配 mmc_bus 上的 mmc_card。匹配条件是，匹配任何的 mmc_card。匹配成功后，mmc_driver.probe 将 mmc_card 注册到 block layer，建立块设备队列。最终用户层会出现设备节点，用户层就可以读写该设备了。

# 整体架构 ---block device

## card 目录

- block.c：在 mmc_bus 上注册 mmc_driver，这个驱动是用来匹配 mmc_bus 上的 mmc_card。匹配条件是，匹配任何的 mmc_card。匹配成功后，mmc_driver.probe 将 mmc_card 注册到 block layer，建立块设备队列。最终用户层会出现设备节点，用户层就可以读写该设备了。
- queue.c：当上层有读写请求进入块设备队列，会调用队列处理函数 mmc_request。mmc_request 会唤醒 mmc_queue_thread 内核线程进行读写请求处理，然后回调 block.c 中的 mmc_blk_issue_rq 进行每个 request 的处理。

# 整体架构 ---mmc core

## core 目录

# 整体架构 ---mmc core

## core 目录

- host.c：给 host controller 驱动提供的接口，通过该文件提供的函数，可以将一个 host 添加到 mmc_core 中来。

# 整体架构 ---mmc core

## core 目录

- host.c：给 host controller 驱动提供的接口，通过该文件提供的函数，可以将一个 host 添加到 mmc_core 中来。
- sd.c, sd_ops.c：提供 SD Card 初始化的代码

# 整体架构 ---mmc core

## core 目录

- host.c：给 host controller 驱动提供的接口，通过该文件提供的函数，可以将一个 host 添加到 mmc_core 中来。
- sd.c, sd_ops.c：提供 SD Card 初始化的代码
- mmc.c, mmc_ops.c：提供 MMC Card 初始化的代码

# 整体架构 ---mmc core

## core 目录

- host.c：给 host controller 驱动提供的接口，通过该文件提供的函数，可以将一个 host 添加到 mmc_core 中来。
- sd.c, sd_ops.c：提供 SD Card 初始化的代码
- mmc.c, mmc_ops.c：提供 MMC Card 初始化的代码
- sdio.c, sdio_bus.c, sdio_cis.c, sdio_ops：提供支持 SDIO 的代码

# 整体架构 ---mmc core

## core 目录

- host.c：给 host controller 驱动提供的接口，通过该文件提供的函数，可以将一个 host 添加到 mmc_core 中来。
- sd.c, sd_ops.c：提供 SD Card 初始化的代码
- mmc.c, mmc_ops.c：提供 MMC Card 初始化的代码
- sdio.c, sdio_bus.c, sdio_cis.c, sdio_ops：提供支持 SDIO 的代码
- core.c：提供 card 热插拔处理，suspend/resume，修改 host 的 clock，power，读写，发送命令等接口。

# 整体架构 ---host driver 通用结构 <1>

下面介绍 host driver 源码文件一般性的结构：

```
1   static struct platform_driver xhost_mmc_driver = {
2           .driver = {
3                   .name   = "xhost-name",
4                   .owner = THIS_MODULE,
5           },
6           .probe    = xhost_mmc_probe,
7           .remove   = xhost_mmc_remove,
8           .suspend = xhost_mmc_suspend,
9           .resume   = xhost_mmc_resume,
10  };
11
12  static int  __init xhost_mmc_init(void)
13  {
14          return platform_driver_register(&xhost_mmc_driver);
15  }
16
17  static void __exit xhost_mmc_exit(void)
18  {
19          platform_driver_unregister(&xhost_mmc_driver);
20  }
21
```

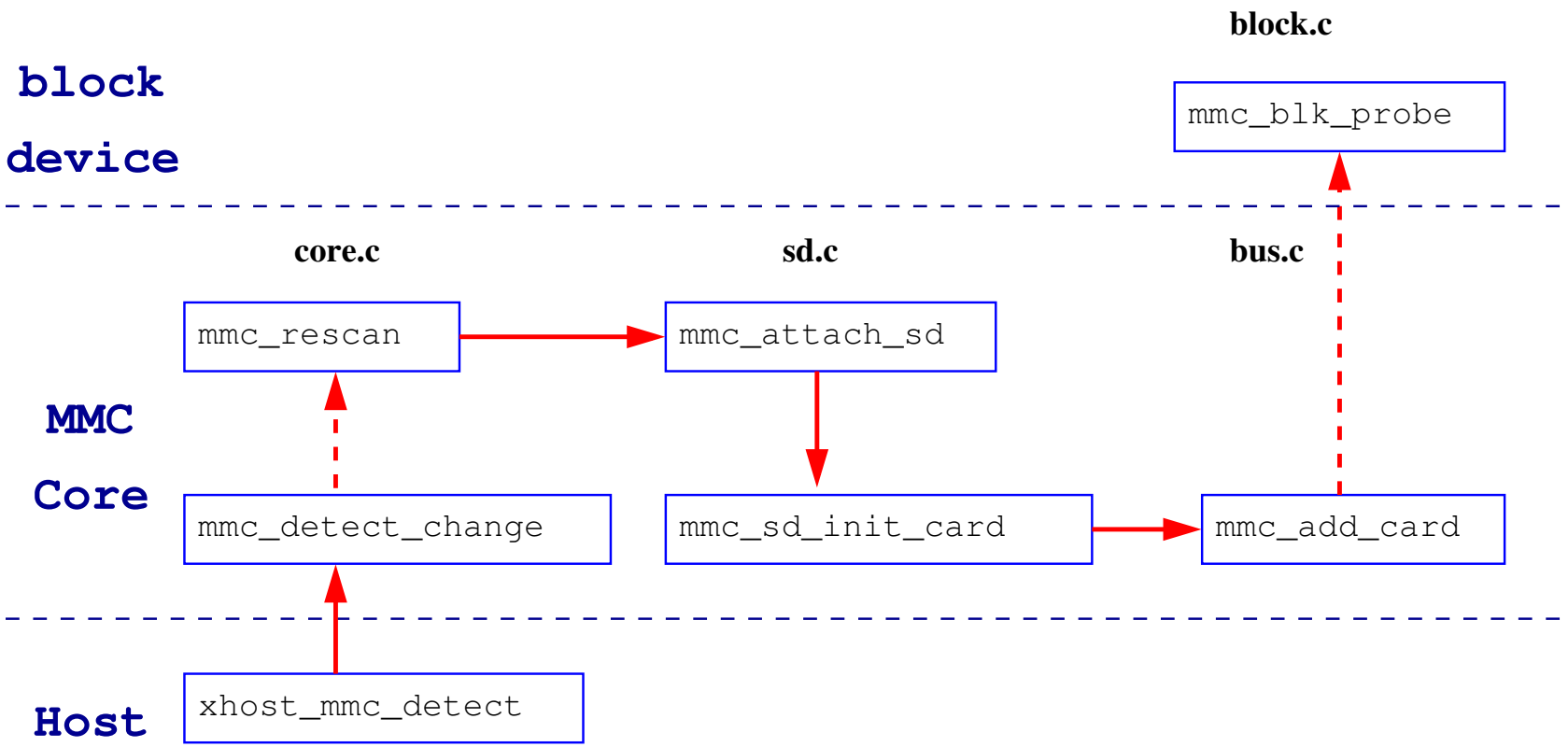
# 整体架构 ---host driver 通用结构 <2>

```
1   static const struct mmc_host_ops xhost_mmc_ops = {
2           .request        = xhost_mmc_request,
3           .set_ios        = xhost_mmc_set_ios,
4           .get_ro         = xhost_mmc_get_ro,
5           .get_cd         = xhost_mmc_get_cd,
6           .enable_sdio_irq = xhost_mmc_enable_sdio_irq,
7   };
8
9   static int xhost_mmc_probe()
10  {
11          ...
12          mmc = mmc_alloc_host(sizeof(struct xhost_mmc_host), ... );
13
14          mmc->ops            = &xhost_mmc_ops;
15          mmc->caps           = MMC_CAP_4_BIT_DATA | ...;
16          ...
17          mmc->max_req_size   = mmc->max_blk_size * mmc->max_blk_count;
18          mmc->max_seg_size   = mmc->max_req_size;
19          mmc->ocr_avail      = ... ;
20
21          ret = request_irq(irq_no, xhost_mmc_irq, IRQF_XX, "name", host);
22          ret = request_irq(irq_no, xhost_mmc_detect, IRQF_XX, "name", host);
23
24          mmc_add_host(mmc);
25          ...
26  }
27
```

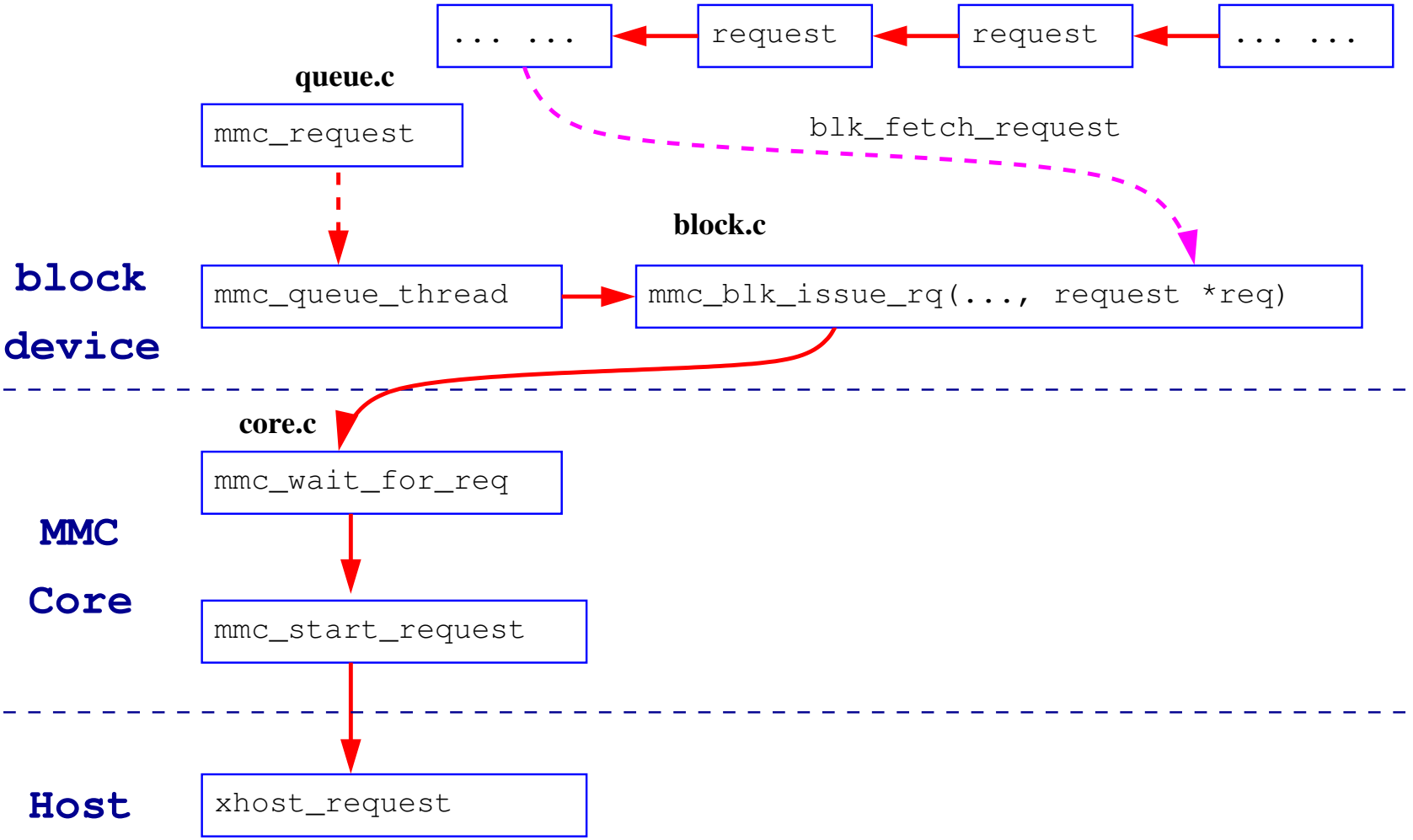
# 整体架构 ---host driver 通用结构 <3>

```
1
2   static irqreturn_t xhost_mmc_detect(int irq, void *devid)
3   {
4           ...
5           mmc_detect_change(host->mmc, msecs_to_jiffies(100));
6   }
7
```

# 整体架构 ---卡探测流程图


block.c
block
device
mmc_blk_probe
core.c
sd.c
bus.c
mmc_rescan
mmc_attach_sd
MMC
Core
mmc_detect_change
mmc_sd_init_card
mmc_add_card
Host
xhost_mmc_detect

# 整体架构 ---读写请求流程图



# 代码导读

- 卡探测 ==> 卡初始化 ==> 注册块设备
- 从队列中取出读写 request ==> 组建命令包 ==> 发送

# The end

Thank you.