AN0105
入门指南

AT32 MCU SDIO入门指南

前言

本应用入门指南主要介绍以下几部分内容：
1. 基于雅特力提供的 V2.x.x 的 BSP 板级支持包来进行 SD 存储卡、多媒体卡（MMC）的命令和数据通信的配置及操作，针对该存储卡的读、写进行讲解和案列解析。
2. 基于雅特力提供的 V2.x.x 的 BSP 板级支持包来进行 SD 存储卡、多媒体卡（MMC）加载 FATFS 文件系统，针对该存储卡的格式化、创建并读写文件进行讲解和案列解析。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	具备 SDIO 的型号

目录

表目录

图目录

1 简介

AT32 MCU 的主机模块（SDIO）在 AHB 外设总线和多媒体卡（MMC）、SD 存储卡、SDIO 卡间提供了操作接口。

SD 储存卡和 SDIO 卡的系统规格书可以通过 SD 卡协议网站(www.sdcard.org)。

多媒体卡系统规格书由 MMCA 技术委员会发布，可以在多媒体卡协会的网站上（www.mmca.org）获得。

1、SD Card：SDSC/SDHC/SDXC，支持的卡最大容量不同，支持 1bit 或 4bit 传输，0-25MHz 或 0-50MHz 的传输模式。

2、MMC Card：支持 1bit，4bit 或 8bit 传输，0-20MHz，0-26MHz 或 0-52MHz 的传输模式。

3、SDIO Card：一种使用 SD 接口协议，支持多功能的卡，比如 wifi 卡，GPS 卡，蓝牙卡等等。

1.1 SDIO 主要结构

SDIO 包含 4 个部分：

1、SDIO 适配器模块：由控制单元、命令单元和数据单元所组成， 实现所有 MMC/SD/SD I/O 卡的相关功能，如时钟的产生、命令和数据的传送

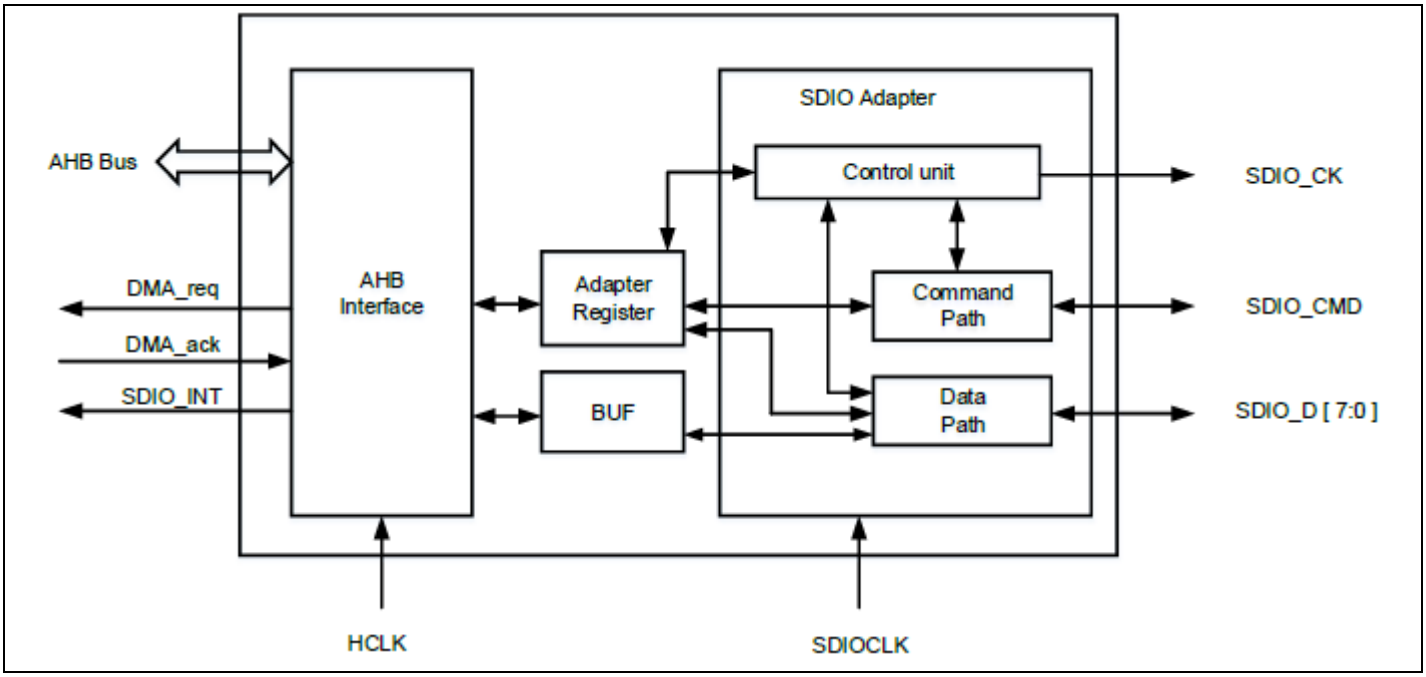
- 控制单元：管理并产生时钟信号
- 命令单元：管理命令的传输
- 数据单元：管理数据的传输

2、AHB 接口：产生中断和 DMA 请求信号

3、适配器寄存器：SDIO 寄存器

4、BUF：用于数据传输校准功能

图 1. SDIO 框图



- 所有数据线配置为复用推挽模式。**SDIO_CMD** 和 **SDIO_D[7：0]**可双向通信，应外接上拉电阻或内部上拉。

- SDIO 使用一个时钟信号：SDIO 适配器时钟（SDIOCLK = AHB 总线时钟（HCLK））。
- 复位后默认情况下 **SDIO_D0** 用于数据传输。初始化后主机可以改变数据总线的宽度。可选 1bit(**SDIO_D0**)、4bit(**SDIO_D[3：0]**)、8bit(**SDIO_D[7：0]**)三种数据总线的宽度。

表 1. SDIO 外部引脚说明

引脚	方向	说明
SDIO_CK	输出	多媒体卡/SD/SDIO 卡时钟。这是从主机至卡的时钟线。
SDIO_CMD	双向	多媒体卡/SD/SDIO 卡命令。这是双向的命令/响应信号线。
SDIO_D[7：0]	双向	多媒体卡/SD/SDIO 卡数据。这些是双向的数据总线。

1.2 SDIO 总线通信

总线上的通信是通过传送命令和数据实现。

1、在多媒体卡/SD/SDIO 总线上的基本操作是命令/响应结构。

2、在 SD/SDIO Card 上传送的数据是只能以数据块的形式传输；在 MMC Card 上传送的数据是以数据块或数据流的形式传输。

图 2. SDIO 命令“无响应”和“有响应”操作

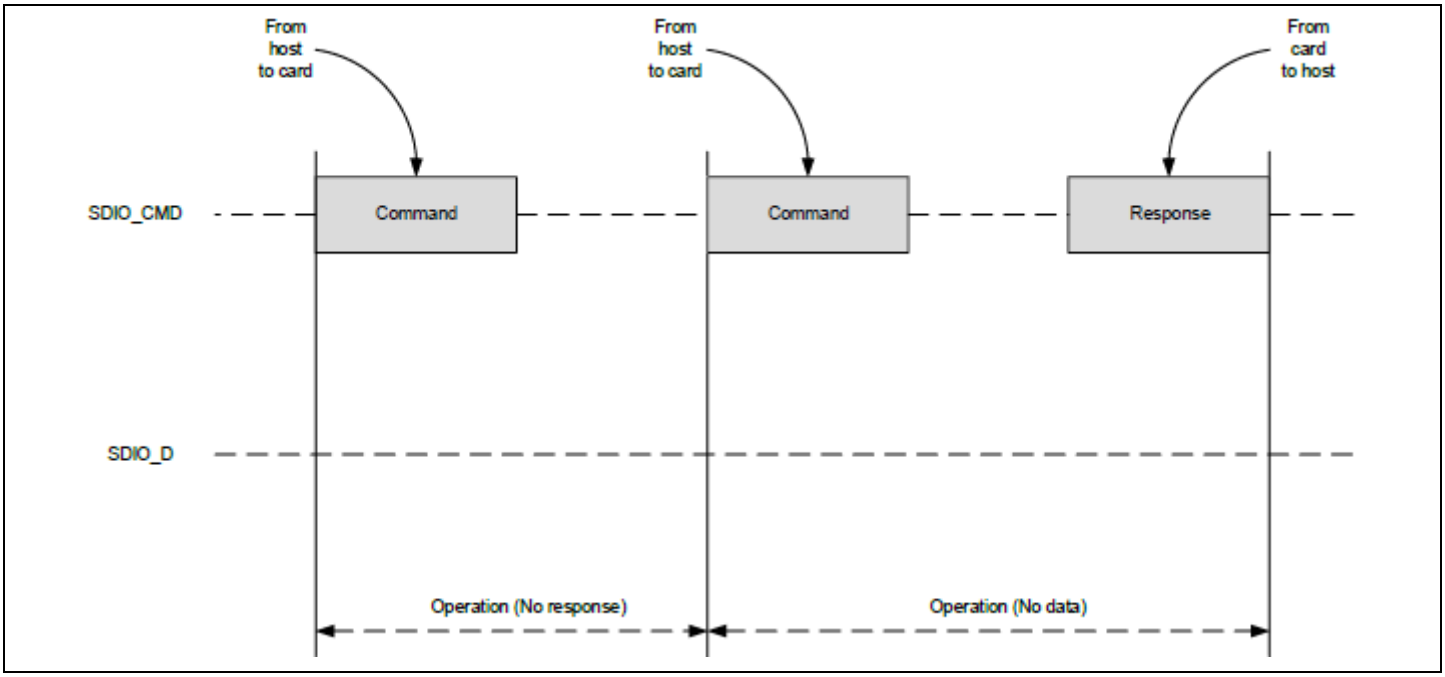


图 3. SDIO（ 多） 数据块读操作

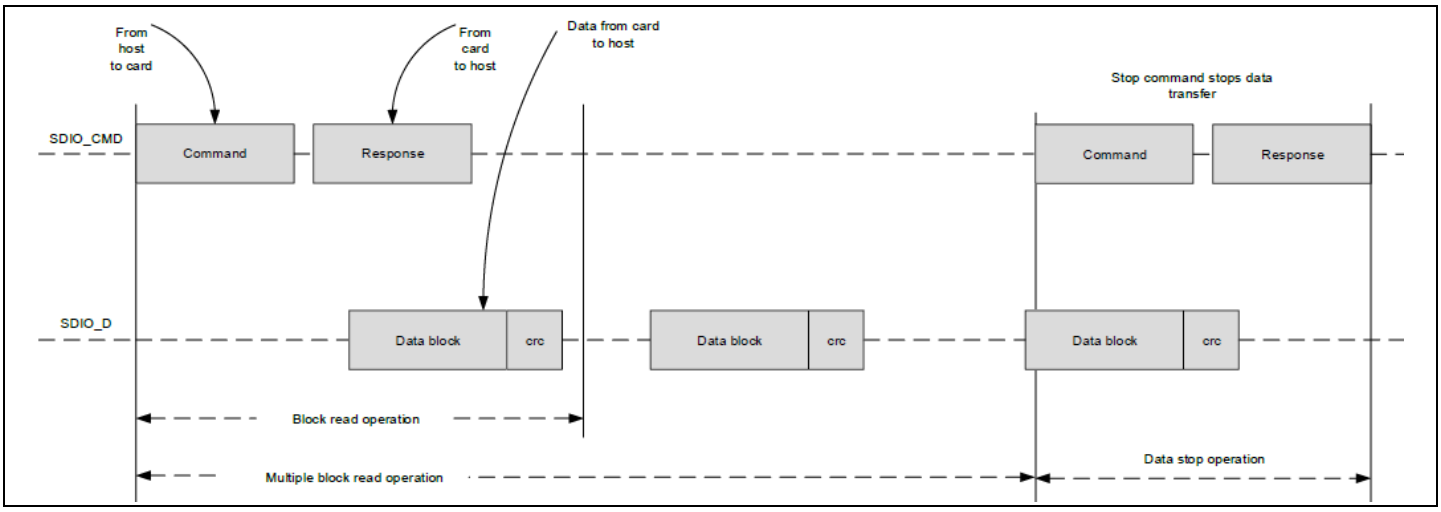
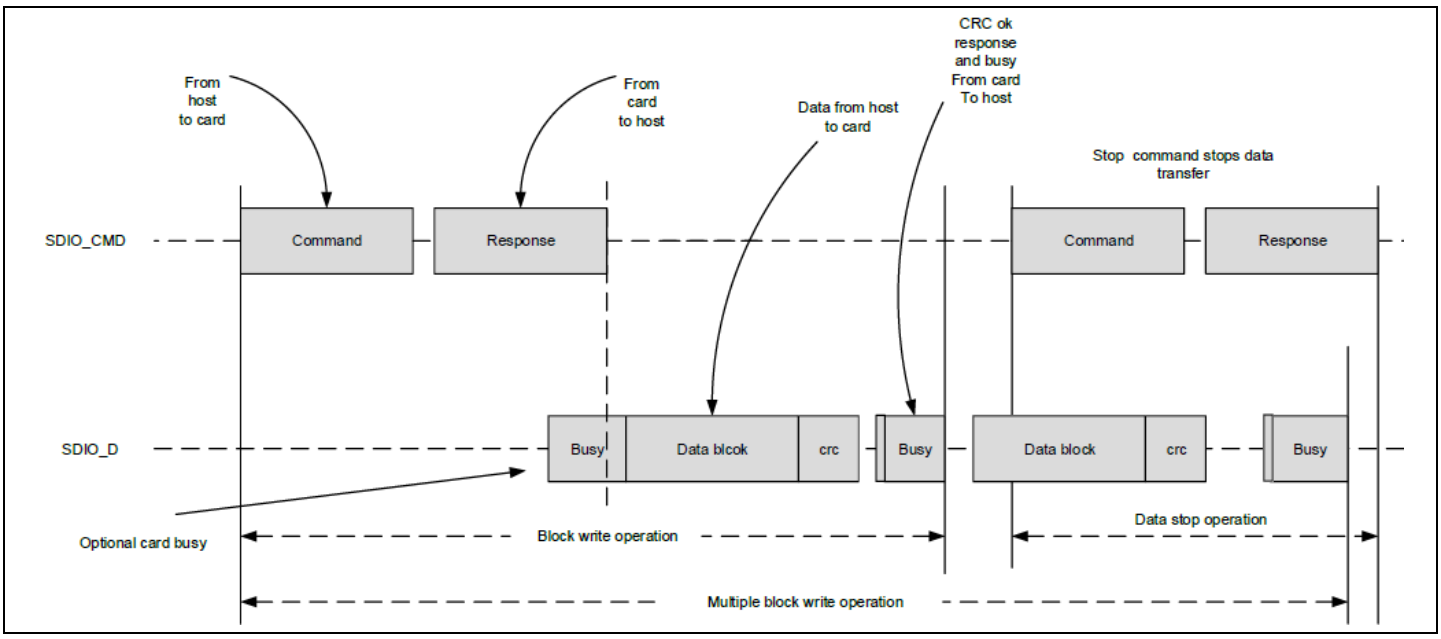


图 4. SDIO（ 多） 数据块写操作



注意：当有Busy（繁忙） 信号时， SDIO（ SDIO_D0 被拉低） 将不会发送任何数据。

图 5. SDIO MMC 卡数据流读操作

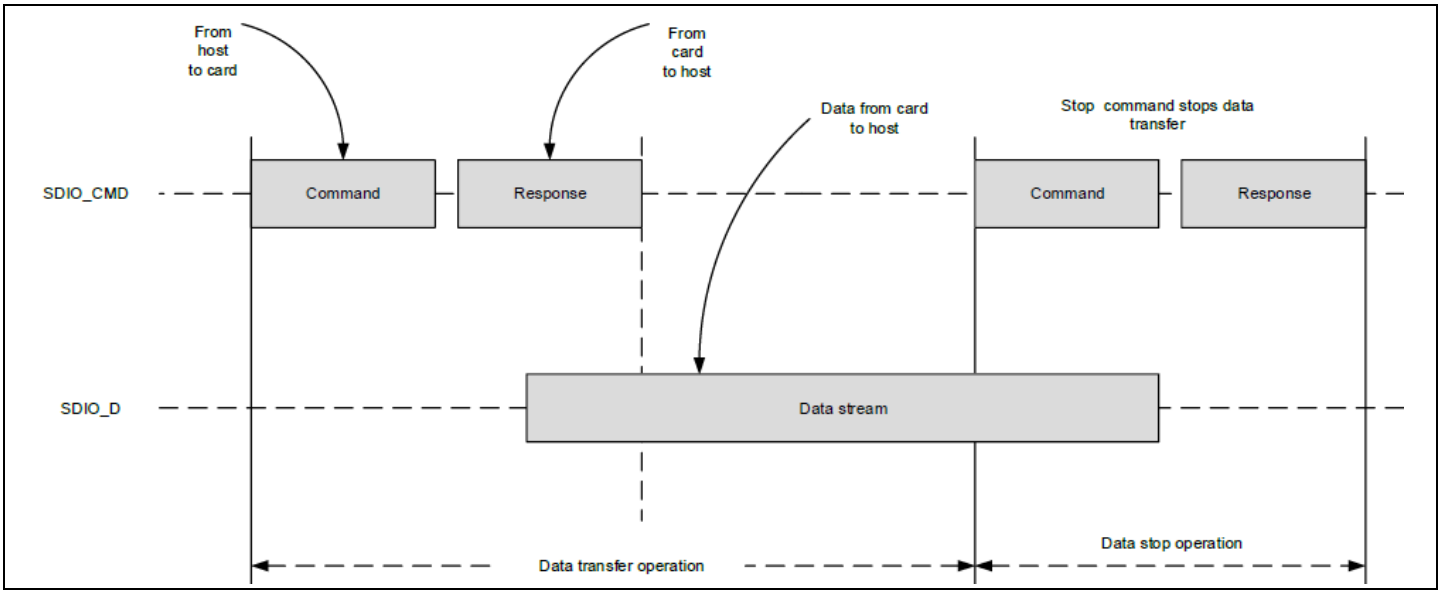
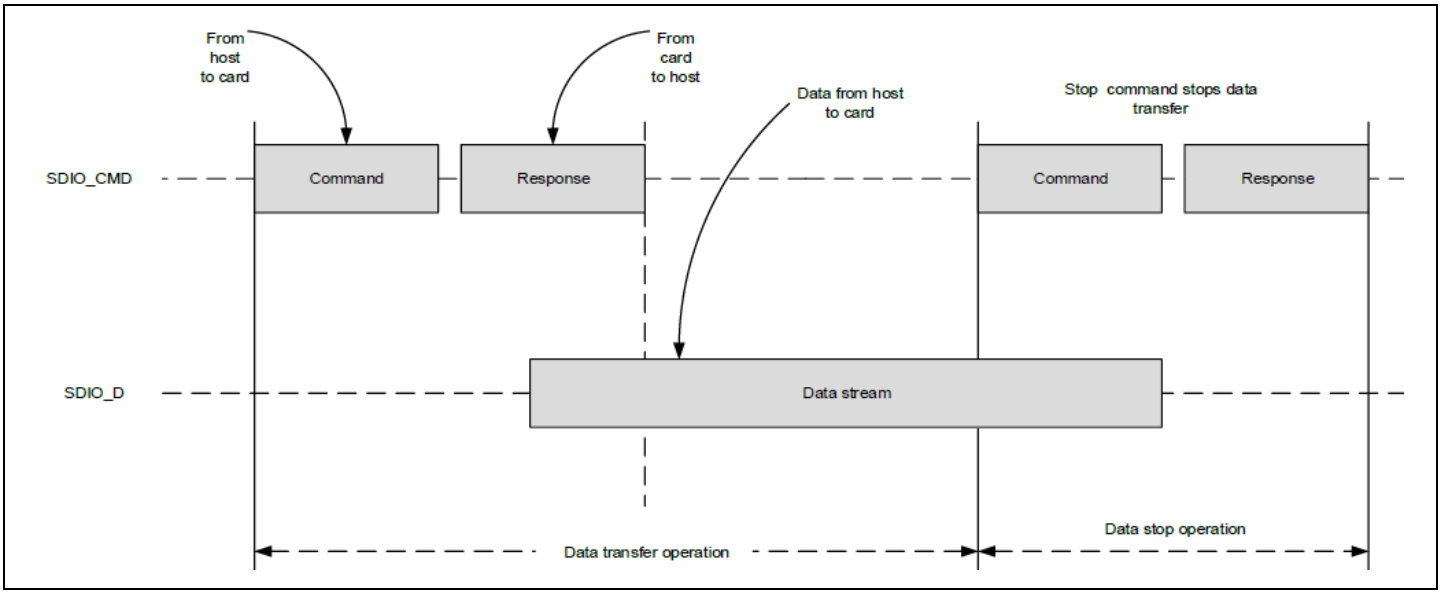


图 6. SDIO MMC 卡数据流写操作



2 SDIO 功能

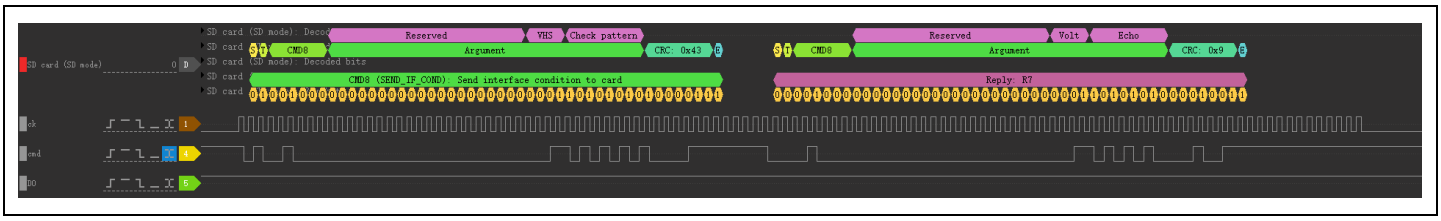
2.1 SDIO 时钟

1） SDIO_CK 是 MCU 端输出到卡的时钟：每个时钟周期在命令和数据线上传输 1bit 命令或数据。

2） SDIO_CK 信号的两下降沿之间为一个时钟周期，硬件在上升沿捕获数据。

3） 当启动了省电模式并且卡总线处于空闲状态（命令通道和数据通道子单元进入空闲阶段后的 8 个时钟周期）。

图 7. 开启省电模式的命令/响应波形图



4） 支持多达 10 位分频系数，也就是 1024 级分频，

此时可得 SDIO_CK 频率 = SDIOCLK / [CLKPSC[9：0] + 2]。

5） 可以使用 bypass 模式，输出一个不分频的时钟， SDIO_CK 频率 = SDIOCLK 。

6） 硬件流控模式，可在数据传输即将发生上溢或者下溢的时候，通过停止 SDIO_CK 防止溢出。

7） 可以配置 CLKEDG bit 来选择时钟的产生。SDIO_CK 可实现略微偏移（半个 SDIOCLK）。

8） 应用时，SD 卡初始化的时候 SDIO_CK 不能大于 400kHZ，之后的时钟频率由对应卡型号限制。

SDIO 时钟配置相关函数

配置 SDIO 时钟

```
void sdio_clock_config(sdio_type *sdio_x, uint16_t clk_div, sdio_edge_phase_type clk_edg);
```

使能省电模式

```
void sdio_power_saving_mode_enable(sdio_type *sdio_x, confirm_state new_state);
```

使能 SDIO 时钟 bypass 模式

```
void sdio_clock_bypass(sdio_type *sdio_x, confirm_state new_state);
```

使能硬件流控模式

```
void sdio_flow_control_enable(sdio_type *sdio_x, confirm_state new_state);
```

设置 SDIO 电源

```
void sdio_power_set(sdio_type *sdio_x, sdio_power_state_type power_state);
```

使能 SDIO 时钟输出

```
void sdio_clock_enable(sdio_type *sdio_x, confirm_state new_state);
```

SDIO 时钟初始化举例：

```
  /* config sdio clock divide and edge phase */
```

```
sdio_clock_config(SDIOx, clk_psc, SDIO_CLOCK_EDGE_FALLING);

/* disable flow control */
sdio_flow_control_enable(SDIOx, FALSE);

/* disable clock bypass */
sdio_clock_bypass(SDIOx, FALSE);

/* disable power saving mode */
sdio_power_saving_mode_enable(SDIOx, FALSE);

/* sdio power on */
sdio_power_set(SDIOx, SDIO_POWER_ON);

/* enable to output sdio_ck */
sdio_clock_enable(SDIOx, TRUE);
```

2.2 命令通道

1） 命令通道单元通过 SDIO_CMD 向卡发送命令并从卡接收响应。

2） 命令超时，即等待卡响应的时间，固定为 64 个 SDIO_CK 时钟周期。这个由通信协议决定，固定不可配置。

3） 可以置位 WAITPEND bit，命令只有在数据传输完之后才由硬件自动发出，而不是立刻发出。多用于流数据的传输模式，目的是保证中止命令可以精准地停止卡的数据传输。

表 2. 命令格式

位	47	46	[45 ： 40]	[39 ： 8]	[7 ： 1]	0
宽度	1	1	6	32	7	1
数值	0	1	-	-	-	1
说明	开始位	传输位	命令索引	参数	CRC7	结束位

4） 根据该命令的需要，可配置等待响应位 (Wait for response bits) 来指示 CPSM 是否需要等待响应。具体可配置为：1、无响应；2、短响应；3、长响应。

表 3. 短响应格式

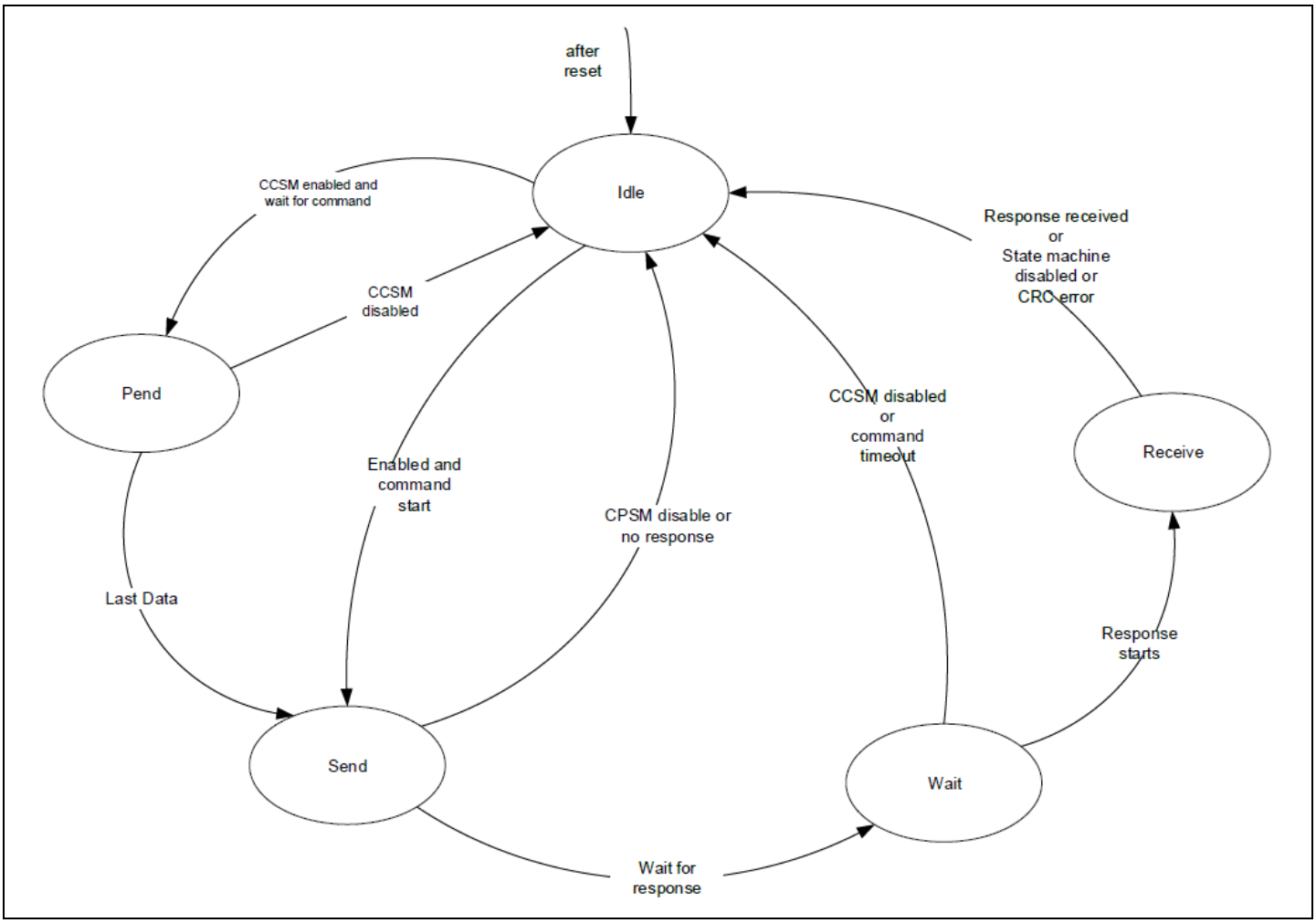
位	47	46	[45 ： 40]	[39 ： 8]	[7 ： 1]	0
宽度	1	1	6	32	7	1
数值	0	0	-	-	-	1
说明	开始位	传输位	命令索引	参数	CRC7(或 1111111)	结束位

表 4. 长响应格式

位	135	134	[133：128]	[127 ： 1]	0
宽度	1	1	6	127	1
数值	0	0	111111	-	1
说明	开始位	传输位	保留	CID 或 CSD(包含内部 CRC7)	结束位

5） 当设置 SDIO_CMDCTRL 寄存器的 CCSMEN 位，控制器开始发送命令。命令发送完成时，命令通道状态机（CCSM）设置命令通道状态标志并在不需要响应时进入空闲状态。当收到响应后，接收到的 CRC 码将会与内部产生的 CRC 码比较，然后设置相应的状态标志。

图 8. 命令通道状态机（CCSM）



6） 当 CMDRSPCMPL、CMDFAIL、CMDCMPL、CMDTIMEOUT 置位后，命令通道状态机（CCSM）都会回到 Idle 状态。而 DOCMD 置位时，CPSM 处于除 Idle 以外的任何状态。

表 5. 命令通道标志

标志	说明
CMDRSPCMPL	已接受到响应(CRC 检测成功)
CMDFAIL	已收到命令响应(CRC 检测失败)

CMDCMPL	命令（不需要响应的命令）已发送
CMDTIMEOUT	命令响应超时(64 个 SDIO_CK 时钟周期)
DOCMD	正在发送命令

命令通道配置相关函数

配置命令通道状态机

```
void sdio_command_config(sdio_type *sdio_x, sdio_command_struct_type *command_struct);
```

使能命令通道状态机

```
void sdio_command_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
```

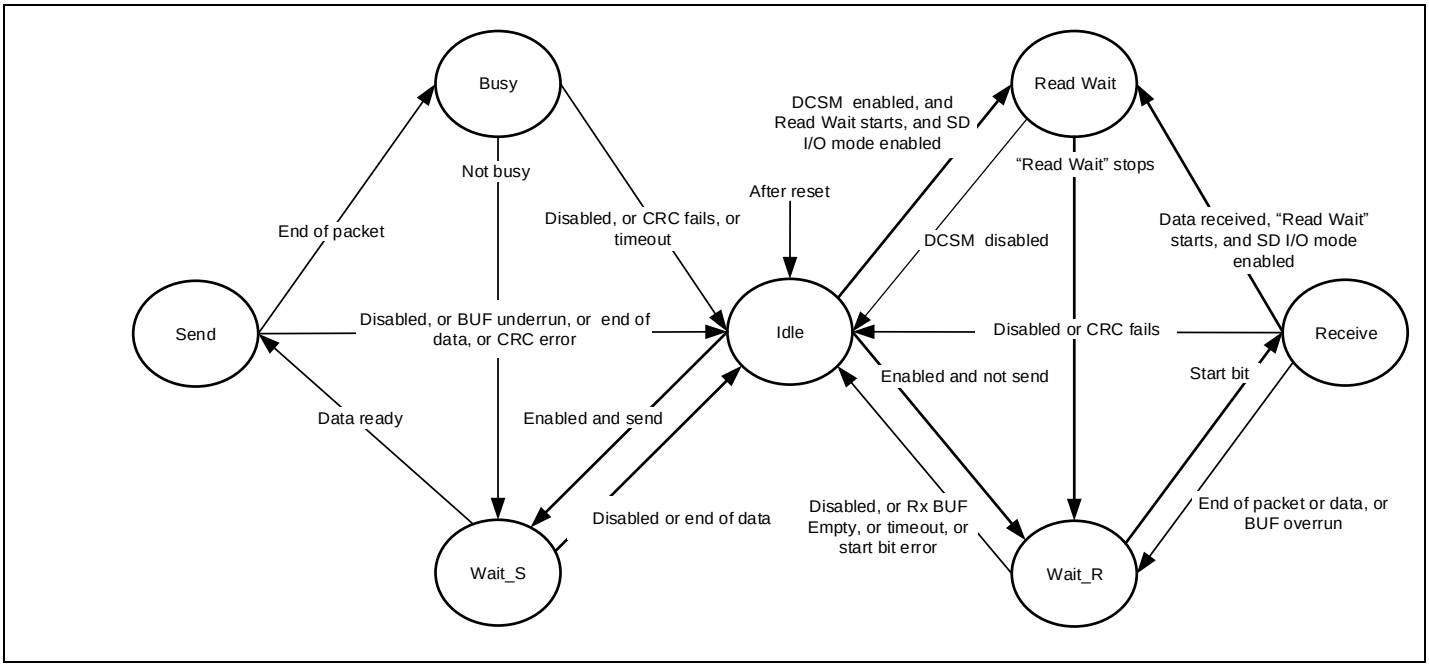
命令通道初始化举例：

```
/* 声明命令通道状态机参数结构体 */
sdio_data_struct_type sdio_data_init_struct;

/* send cmd3, get rca */
sdio_command_init_struct.argument = 0x00;
sdio_command_init_struct.cmd_index = SD_CMD_SET_REL_ADDR;
sdio_command_init_struct.rsp_type = SDIO_RESPONSE_SHORT;
sdio_command_init_struct.wait_type = SDIO_WAIT_FOR_NO;

/* 配置命令通道状态机参数 */
sdio_command_config(SDIOx, &sdio_command_init_struct);

/* 使能命令通道状态机 */
sdio_command_state_machine_enable(SDIOx, TRUE);
```

2.3 数据通道

1）数据通道单元通过 SDIO_D[7：0]在主机与卡之间传输数据。

图 9. 数据通道状态机（DCSM）



2）数据 BUF（先进先出）子单元是一个具有发送和接收单元的数据缓冲区。

3）BUF 包含一个每字 32 位宽、共 32 个字的数据缓冲区，共 128Byte。

表 6. 数据令牌格式

说明	开始位	数据	CRC16	结束位
块数据	0	-	有	1
流数据	0	-	无	1

4）DOTX 和 DORX 由数据通道单元设置而且是互斥的：

- 当 DOTX 标志有效时，BUF 代表发送数据缓冲区。（DPSM 处于 Wait_R 或者 Receive 状态）
- 当 DORX 标志有效时，BUF 代表接收数据缓冲区。（ DPSM 处于 Wait_S 或者 Send 状态）

表 7. 发送 BUF 状态标志

标志	说明
TXBUFF	当所有 32 个发送 BUF 字都有有效的数据时，该标志为高。
TXBUFE	当所有 32 个发送 BUF 字都没有有效的数据时，该标志为高。
TXBUFH	当 8 个或更多发送 BUF 字为空时，该标志为高。该标志可以作为 DMA 请求。
TXBUF	当发送 BUF 包含有效数据时，该标志为高。该标志的意思刚好与 TXBUF_E 相反。
TXERRU	当发生下溢错误时，该标志为高。写入 SDIO 清除寄存器时清除该标志。（ DCSM 回到 Idle）

表 8. 接收 BUF 状态标志

标志	说明
RXBUFF	当所有 32 个接收 BUF 字都有有效的数据时，该标志为高。
RXBUFE	当所有 32 个接收 BUF 字都没有有效的数据时，该标志为高。

RXBUFH	当 8 个或更多接收 BUF 字有有效的数据时，该标志为高。该标志可以作为 DMA 请求。
RXBUF	当接收 BUF 包含有效数据时，该标志为高。该标志的意思刚好与 RXBUF_E 相反。
RXERRO	当发生上溢错误时，该标志为高。写入 SDIO 清除寄存器时清除该标志。（ DCSM 回到 Wait_R）

SDIO_D[7：0]时序

1）在块模式下，发送完数据后，卡会返回一个确认 CRC 的序列，数据通道状态机（DCSM）在等待“这个序列和 Busy 结束”时有超时控制，超时时间由 SDIO 数据定时器寄存器（SDIO_DTTMR）设置。

2）卡端返回的 CRC status（5bit）只会发送在 SDIO_D0 上，卡端收到正确的数据（CRC 正确）后发出的序列为“00101”，若是错误的数据（CRC 错误）的序列为“01011”。

3）数据通道状态机（DCSM）在 WAIT_R 等待接收数据时也有超时控制，超时时间也由 SDIO 数据定时器寄存器（SDIO_DTTMR）设置。

4）在块模式下的数据传输总个数一定要是 block size 的整数倍。当一个块发送完后需要收到“CRC 序列和 Busy 结束”时，硬件才会发送下一个数据块。在流模式下不需设定 block size 的大小。

图 10. 1bit 数据传输方式

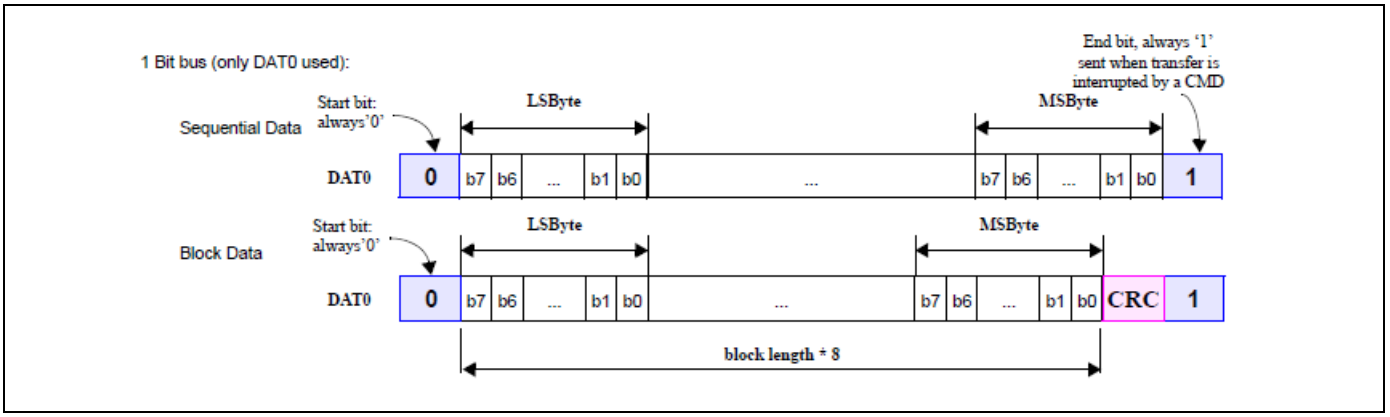


图 11. 4bit 数据传输方式

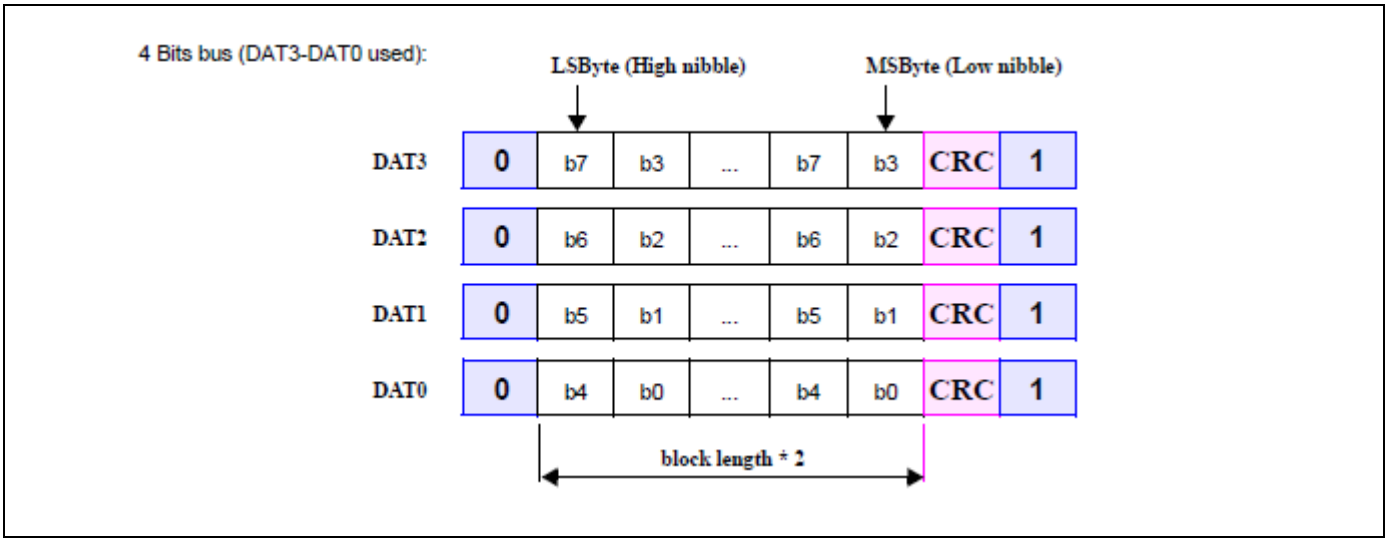
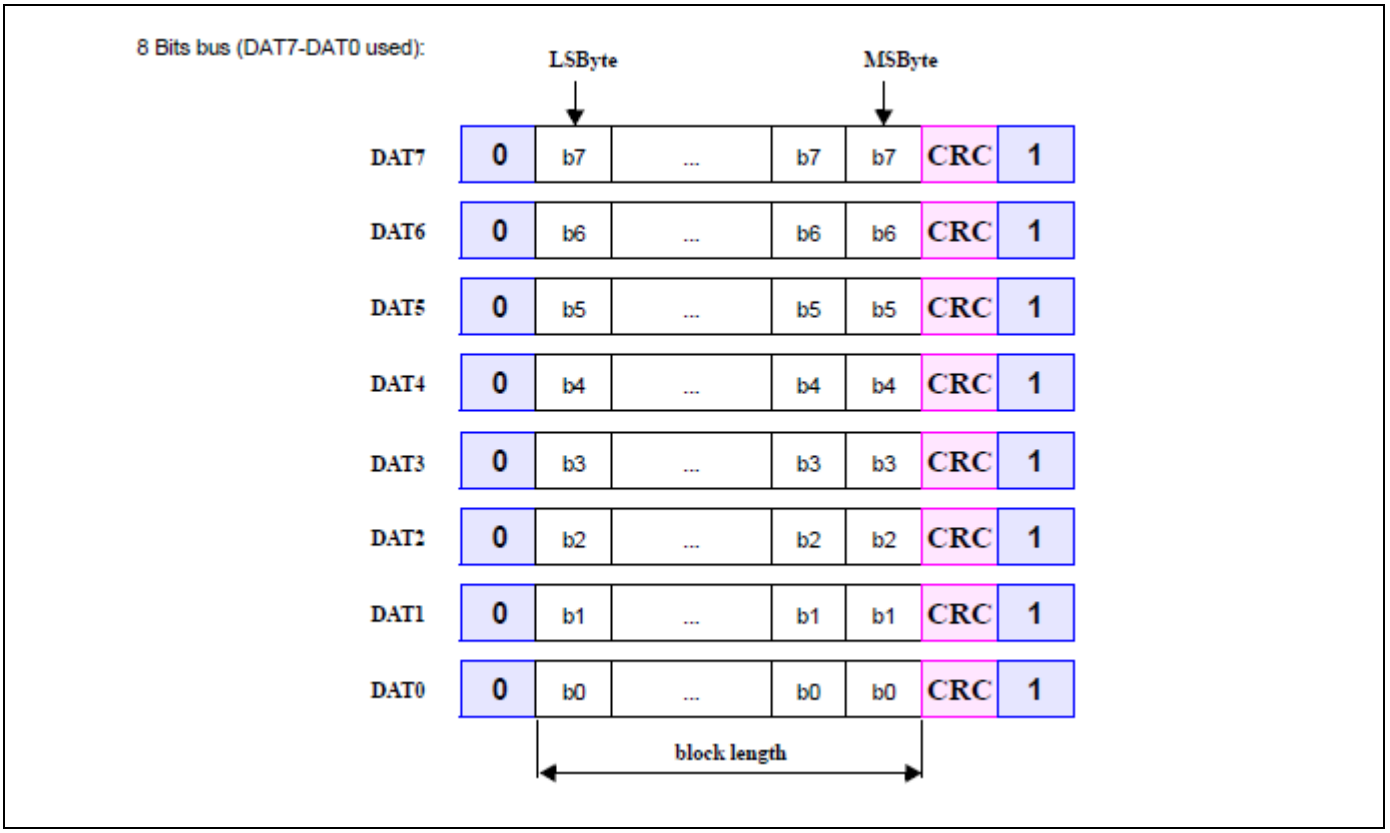


图 12. 8bit 数据传输方式



数据通道配置相关函数

配置数据通道状态机

```
void sdio_data_config(sdio_type *sdio_x, sdio_data_struct_type *data_struct);
```

使能数据通道状态机

```
void sdio_data_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
```

设置数据总线宽度

```
void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
```

数据通道初始化举例：

```
/* 声明数据通道状态机参数结构体 */
sdio_data_struct_type sdio_data_init_struct;

/* config sdio bus width */
sdio_bus_width_config(SDIOx, SDIO_BUS_WIDTH_D1);

sdio_data_init_struct.block_size = SDIO_DATA_BLOCK_SIZE_8B;
sdio_data_init_struct.data_length = 8;
sdio_data_init_struct.timeout = SD_DATATIMEOUT;
sdio_data_init_struct.transfer_direction = SDIO_DATA_TRANSFER_TO_CONTROLLER;
sdio_data_init_struct.transfer_mode = SDIO_DATA_BLOCK_TRANSFER;
```

```
/* 配置数据通道状态机参数 */
sdio_data_config(SDIOx, &sdio_data_init_struct);

/* 使能数据通道状态机 */
sdio_data_state_machine_enable(SDIOx, TRUE);
```

2.4 SDIO AHB 接口

1）AHB 接口产生中断和 DMA 请求，并访问 SDIO 接口寄存器和数据 BUF。它包含一个数据通道、寄存器译码器和中断/DMA 控制逻辑。

2）SDIO 中断：

当至少有一个选中的状态标志为高时，中断控制逻辑产生中断请求。中断屏蔽寄存器用于选择可以产生中断的条件，如果设置了相应的屏蔽标志位，则对应的状态标志可以产生中断。

表 9. SDIO 中断屏蔽寄存器

域	简称	功能
位 22	IOIFIEN	SD I/O 模式接收中断使能（SD I/O mode received interrupt enable）
位 21	RXBUFIEN	接收 BUF 中的数据有效产生中断（Data available in RxBUF interrupt enable）
位 20	TXBUFIEN	发送 BUF 中的数据有效产生中断（Data available in TxBUF interrupt enable）
位 19	RXBUFEIEN	接收 BUF 空产生中断（RxBUF empty interrupt enable）
位 18	TXBUFEIEN	发送 BUF 空产生中断（TxBUF empty interrupt enable）
位 17	RXBUFFIEN	接收 BUF 满产生中断（RxBUF full interrupt enable）
位 16	TXBUFFIEN	发送 BUF 满产生中断（TxBUF full interrupt enable）
位 15	RXBUFHIEN	接收 BUF 半满产生中断（RxBUF half full interrupt enable）
位 14	TXBUFHIEN	发送 BUF 半空产生中断（TxBUF half empty interrupt enable）
位 13	DORXIEN	正在接收数据产生中断（Data receive acting interrupt enable）
位 12	DOTXIEN	正在发送数据产生中断（Data transmit acting interrupt enable）
位 11	DOCMDIEN	正在传输命令产生中断（Command acting interrupt enable）
位 10	DTBLKCMPLIEN	数据块传输结束产生中断（Data block end interrupt enable）
位 9	SBITERRIEN	起始位错误产生中断（Start bit error interrupt enable）
位 8	DTCMPLIEN	数据传输结束产生中断（Data end interrupt enable）
位 7	CMDCMPLIEN	命令已发送产生中断（Command sent interrupt enable）
位 6	CMDRSPCMPLIEN	接收到响应产生中断（Command response received interrupt enable）
位 5	RXERROIEN	接收 BUF 上溢错误产生中断（RxBUF overrun error interrupt enable）
位 4	TXERRUIEN	发送 BUF 下溢错误产生中断（TxBUF underrun error interrupt enable）
位 3	DTTIMEOUTIEN	数据超时产生中断（Data timeout interrupt enable）
位 2	CMDTIMEOUTIEN	命令超时产生中断（Command timeout interrupt enable）
位 1	DTFAILIEN	数据块 CRC 检测失败产生中断（Data CRC fail interrupt enable）
位 0	CMDFAILIEN	命令 CRC 检测失败产生中断（Command CRC fail interrupt enable）

3）DMA 接口：DMA 主要用于在 SDIO BUF 和 Memory 之间传输数据。

- BUF 在接收数据时，RXBUFH 标志作为 DMA 接收请求。
- BUF 在发送数据时，TXBUFH 标志作为 DMA 发送请求。

SDIO 中断/DMA 配置相关函数

SDIO 中断配置函数

```
void sdio_interrupt_enable(sdio_type *sdio_x, uint32_t int_opt, confirm_state new_state);
```

SDIO DMA 使能函数

```
void sdio_dma_enable(sdio_type *sdio_x, confirm_state new_state);
```

使能 SDIO DMA 接收数据举例：

```
dma_init_type dma_init_struct;
dma_default_para_init(&dma_init_struct);

crm_periph_clock_enable(CRM_DMA2_PERIPH_CLOCK, TRUE);

dma_reset(DMA2_CHANNEL4);
dma_channel_enable(DMA2_CHANNEL4, FALSE);

dma_init_struct.peripheral_base_addr = (uint32_t)&SDIOx->buf;
dma_init_struct.memory_base_addr = (uint32_t)mbuf;
dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
dma_init_struct.buffer_size = buf_size / 4;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_WORD;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_WORD;
dma_init_struct.loop_mode_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_HIGH;
dma_init(DMA2_CHANNEL4, &dma_init_struct);

dmamux_init(DMA2MUX_CHANNEL4, DMAMUX_SDIOx);
dmamux_enable(DMA2, TRUE);
dma_channel_enable(DMA2_CHANNEL4, TRUE);

sdio_dma_enable(SDIOx, TRUE);
```

3 数据传输模式

当 SD/MMC 卡需要读/写数据时，可以配置、使能 SDIO 主机的数据通道后，通过 PIO 模式或 DMA 模式来读/写 SDIO 的数据 BUF 的方式实现。

3.1 PIO 模式

SDIO 的数据 BUF 共 128 字节，读/写数据共用。根据 SDIO 主机数据通道所配置的传送方向来判断读或写数据 BUF。
在读取数据 BUF 接收数据时，用户查询 SDIO_STS 寄存器的 RXBUFH 标志位，如置位可读取 8 字节的数据，最后再查询 RXBUF 标志位，以读完剩余小于 8 字节的未读数据。
在写入数据 BUF 发送数据时，用户查询 SDIO_STS 寄存器的 TXBUFH 标志位，如置位可写入最多 8 字节的数据，直至写完所有待发送的数据。
当在 PIO 模式下运行时，用户都必须确保轮询状态。

3.2 DMA 模式

DMA 模式是访问数据 BUF 的另一种选择。使用 DMA 控制器来代替 CPU 对数据 BUF 的访问，可以节省 CPU 运行的时间。使用 DMA 控制器之前，要先使能 SDIO 的 DMA 模式，再去设置 DMA 控制器的功能，最后使能与数据 BUF 读或写相关的中断，用以判断数据 BUF 读/写是否完成，是否有数据校验错误等。控制方式和范例可以参照 BSP 里的 demo。
SDIO 用 DMA 模式读或写数据 BUF 时，只能以 WORD 为最小传输单位。DMA 的长度需换算成 WORD 单位，数据宽度也必须选择 WORD。

DMA 模式相关配置函数

SDIO 的 DMA 模式使能

```
void sdio_dma_enable(sdio_type *sdio_x, confirm_state new_state);
```

SDIO 的中断使能

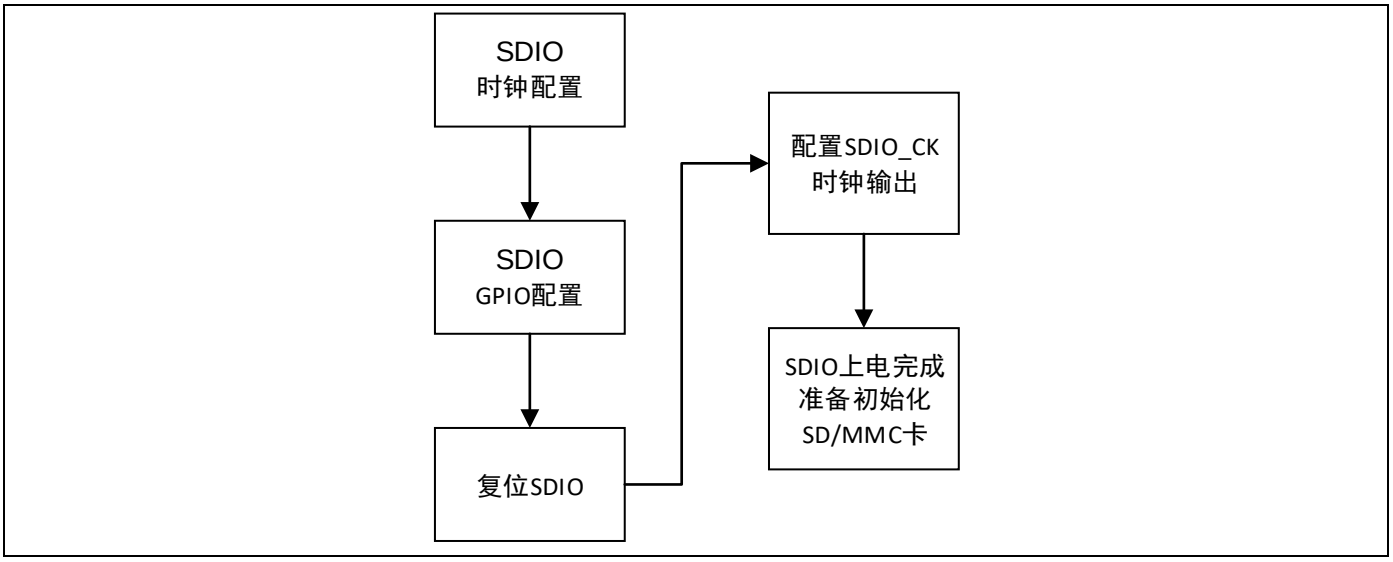
```
void sdio_interrupt_enable(sdio_type *sdio_x, uint32_t int_opt, confirm_state new_state);
```

4 SDIO 主机接口初始化

4.1 初始化 SDIO 主机

这部分介绍了如何去初始化 SD/MMC 主机接口来建立命令通道和数据通道去初始化 SD/MMC 卡。

图 13. 初始化 SD/MMC 主机接口流程图



时钟配置

SDIO 使用一个时钟信号：SDIO 适配器时钟（SDIOCLK = AHB 总线时钟（HCLK））。

SDIO_CK 的时钟配置

对于 SD/MMC 卡初始化配置时，该时钟范围在 100 到 400KHz。

4.2 SD 卡初始化

SD 卡上电和初始化的流程可参考协议”SD Physical Layer Specification Version 2.00”来实现，并由此来配置命令通道和数据通道去识别和初始化 SD 卡。

图 14. SD 卡识别和初始化流程图

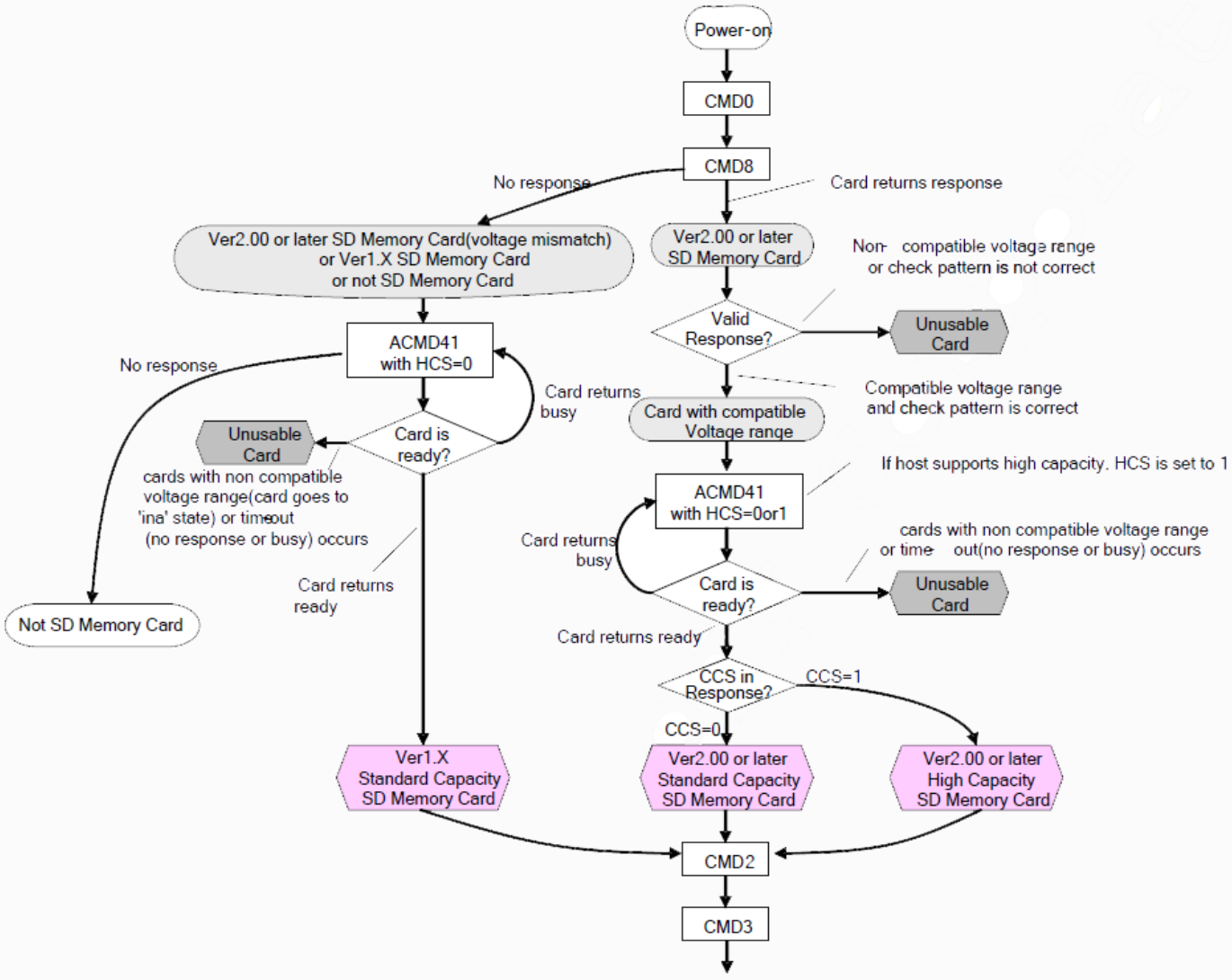


Figure 4-2: Card Initialization and Identification Flow (SD mode)

SD 卡的上电和初始化

发送 CMD0：软件复位命令，将卡置于空闲状态。

发送 CMD8：根据卡端的响应识别 SD 卡的版本型号和支持的电压范围。

- 如果卡端回复了响应，这说明是 V2.00 或更新的 SD 卡。再判断响应是否有效，如响应无效，则支持的电压范围不兼容；如有效，则支持的电压范围是兼容的。
- 如果卡端没有回复响应。则说明是 V2.00 或更新的 SD 卡（没有匹配支持的电压），或是 V1.X 的 SD 卡，或是没有连接 SD 卡。

发送 ACMD41：获取 SD 卡的 OCR(Operation Conditions Register)。

- 读取 OCR 中的 Busy(Card power up status) bit，判断 SD 卡的上电过程是否完成，直到该位置 1 说明上电完成。
- 读取 OCR 中的 CCS(Card Capacity Status) bit，判断该卡是高容量或是标准容量的 V2.00 的 SD 卡。

SD 卡的识别过程

发送 CMD2：获取 SD 卡的 CID(Card IDentification) Register。

发送 CMD3：获取 SD 卡的相对地址。

发送 CMD9：获取 SD 卡的 CSD(Card Specific Data) Register。

配置 SD 卡的数据总线宽度

1）发送 ACMD6：改变 SD 卡的数据总线宽度（可支持 1-bit 或 4-bit 线宽）

2）配置 SDIO 主机的数据总线宽度线宽。

总线宽度设置函数

```
void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
```

配置 SD 卡数据总线宽度举例：

```
status = sd_bus_wide_enable(TRUE);
if(status == SD_OK)
{
  sdio_bus_width_config(SDIOx, mode);
}
```

4.3 MMC 卡初始化

MMC 卡上电和初始化的流程可参考协议” MultiMediaCard (MMC) Electrical Standard(MMCA, 4.2)”来实现，并由此来配置命令通道和数据通道去识别和初始化 MMC 卡。

图 15. MMC 卡识别和初始化流程图

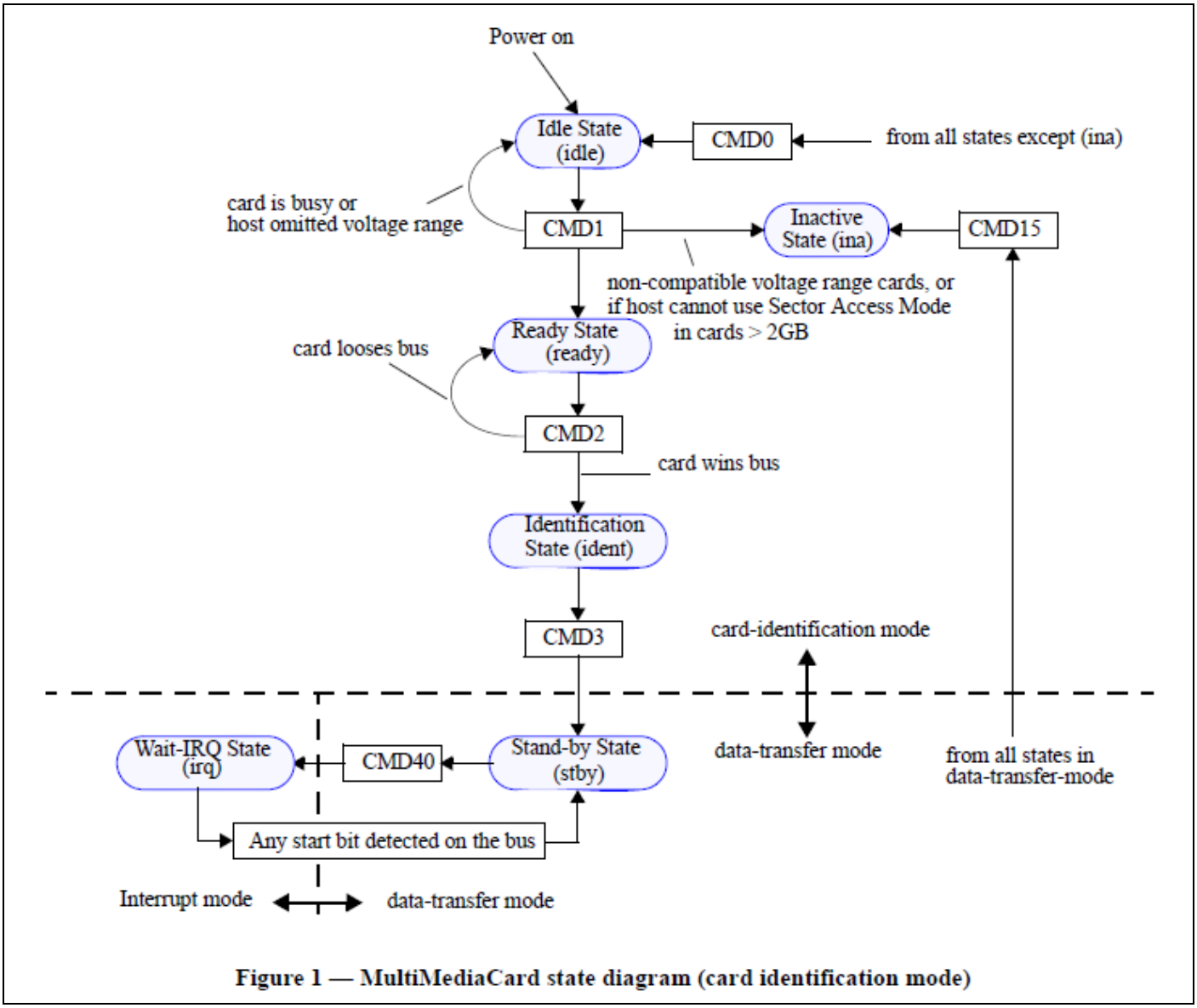


Figure 1 — MultiMediaCard state diagram (card identification mode)

MMC 卡的上电和初始化

发送 CMD0：软件复位命令，将卡置于空闲状态。

发送 CMD1：获取 MMC 卡的 OCR(Operation Conditions Register)。

- 读取 OCR 中的 Busy(Card power up status) bit，判断 MMC 卡的上电过程是否完成，直到该位置 1 说明上电完成。

MMC 卡的识别过程

发送 CMD2：获取 MMC 卡的 CID(Card IDentification) Register。

发送 CMD3：设置 MMC 卡的相对地址。

发送 CMD9：获取 MMC 卡的 CSD(Card Specific Data) Register。

配置 MMC 卡的数据总线宽度

1）发送 CMD6：改变 MMC 卡的数据总线宽度（可支持 1-bit 或 4-bit 或 8-bit 线宽）

2）配置 SDIO 主机的数据总线宽度线宽。

总线宽度设置函数

```
void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
```

配置 MMC 卡数据总线宽度举例：

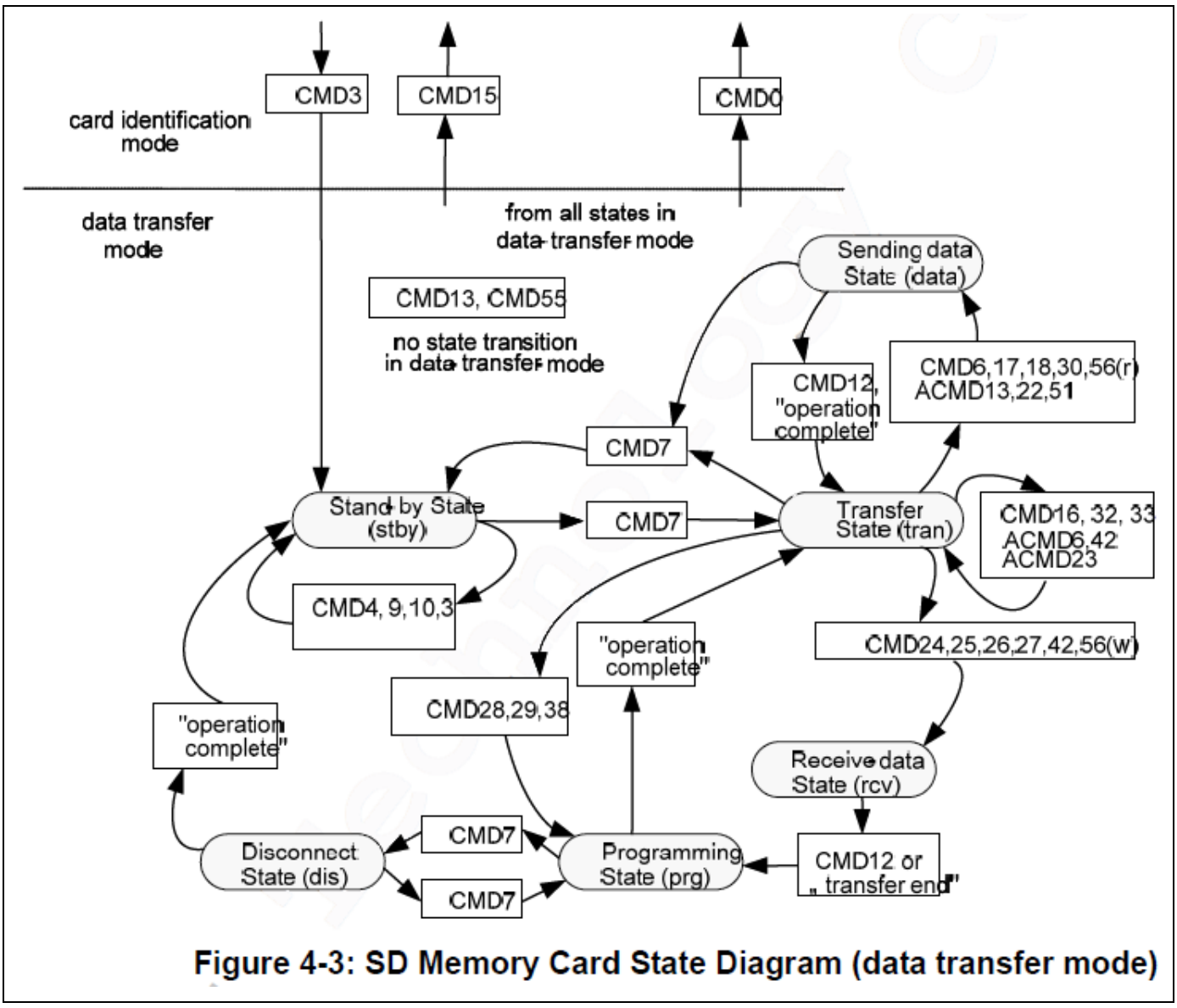
```
  status = mmc_switch(EXT_CSD_CMD_SET_NORMAL, EXT_CSD_BUS_WIDTH,
(uint8_t)mode);
  if(status == SD_OK)
  {
    sdio_bus_width_config(SDIOx, mode);
  }
```

5 SDIO 主机接口读/写 SD/MMC 卡

5.1 读/写 SD 卡

在读/写 SD 卡时，只能以数据块的方式进行读/写。当 SD 卡上电并初始化完成，由空闲状态进入传输状态后，就可以进行 SD 卡的读/写操作。

图 16. SD 卡状态图（数据传输模式）



在初始化完成后，SD 卡进入 Stand-by 状态，需发送 CMD7 根据相对地址去选中 SD 卡，此时可进入到 transfer 状态，便可以进行 SD 卡的读/写操作。

发送 CMD13：获取 SD 卡状态。

发送 CMD16：设置 SD 卡单块的大小。

下面的命令是用于读/写单块或多块的数据：

- CMD1：读单块数据。
- CMD18：读多块数据。
- CMD23：写单块数据。
- CMD24：写多块数据。

5.2 读/写 MMC 卡

在读/写 MMC 卡时，可以以数据块或数据流的方式进行读/写。当 MMC 卡上电并初始化完成，由空闲状态进入传输状态后，就可以进行 MMC 卡的读/写操作。

图 17. MMC 卡状态图（数据传输模式）

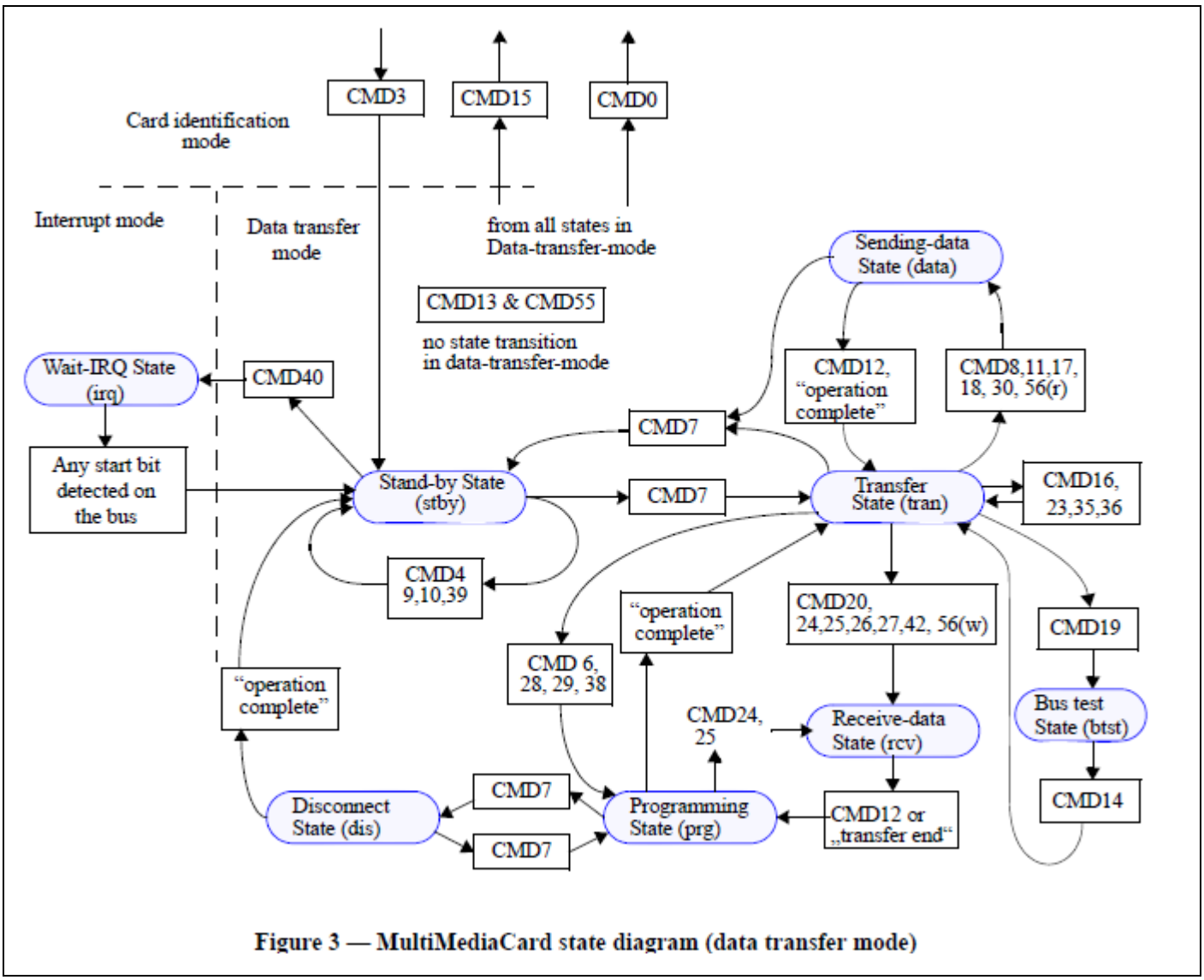


Figure 3 — MultiMediaCard state diagram (data transfer mode)

在初始化完成后，MMC 卡进入 Stand-by 状态，需发送 CMD7 根据相对地址去选中 MMC 卡，此时可进入到 transfer 状态，便可以进行 MMC 卡的读/写操作。

发送 CMD13：获取 MMC 卡状态。

发送 CMD16：设置 MMC 卡单块的大小。

下面的命令是用于读/写单块或多块的数据：

- CMD1：读单块数据。
- CMD18：读多块数据。
- CMD23：写单块数据。
- CMD24：写多块数据。

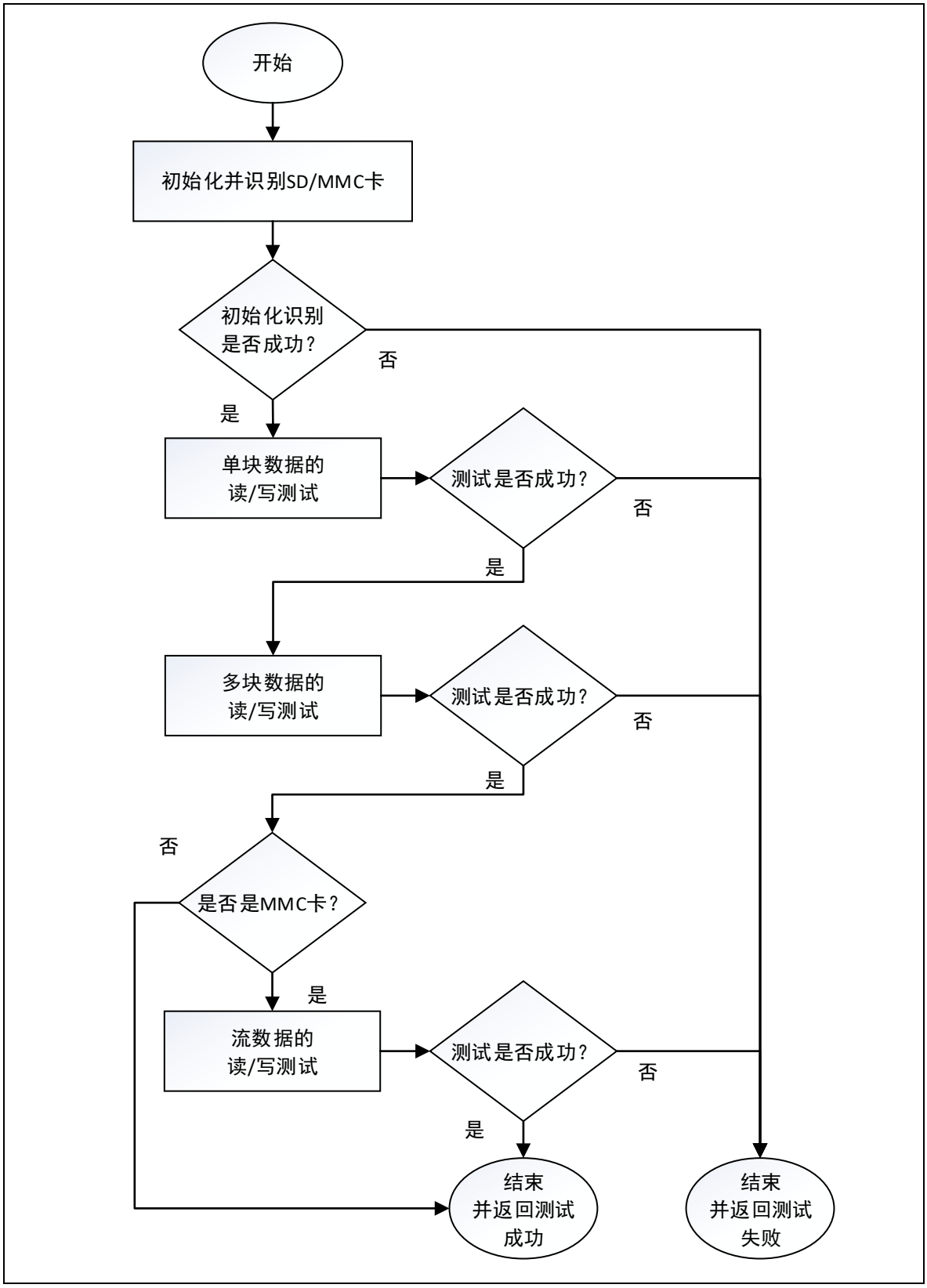
下面的命令是用于读/写数据流：

- CMD11：读数据流。
- CMD20：写数据流。

5.3 读/写 SD/MMC 卡案例

下图展示了读/写 SD/MMC 卡案例的流程图。

图 18. 读/写 SD/MMC 卡案例



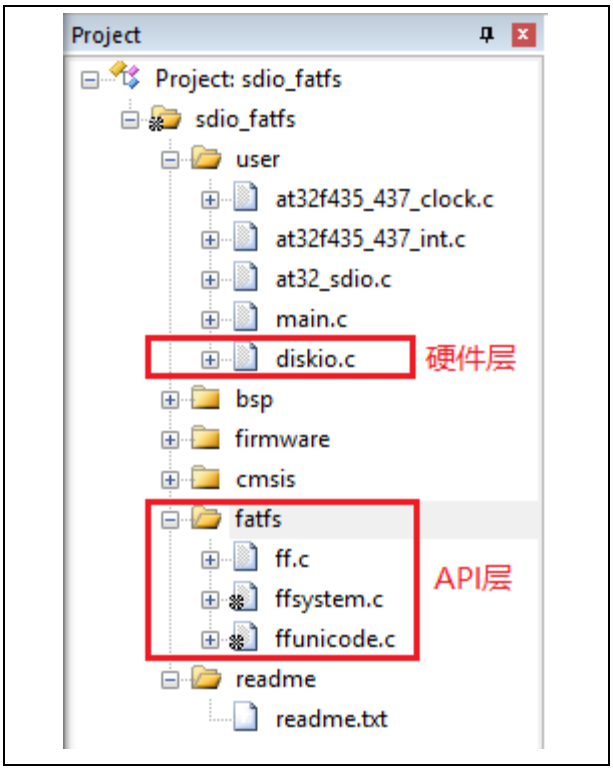
6 SDIO 主机读写基于 SD/MMC 卡的 FATFS 文件

在 SD/MMC 卡上装载 FATFS 文件系统，文件能被传送，可使用 PIO 或 DMA 的传输模式。后面有案例会具体描述如何去创建、写入然后读取一个文件在 SD/MMC 卡上。

6.1 将文件系统导入工程文件

在案例中，我们需要导入 FATFS 文件系统的 API 层和硬件层，并根据具体的存储介质来修改硬件层。通过”ffconf.h”配置 FatFs 的相关功能（可裁剪），以满足应用的需要。

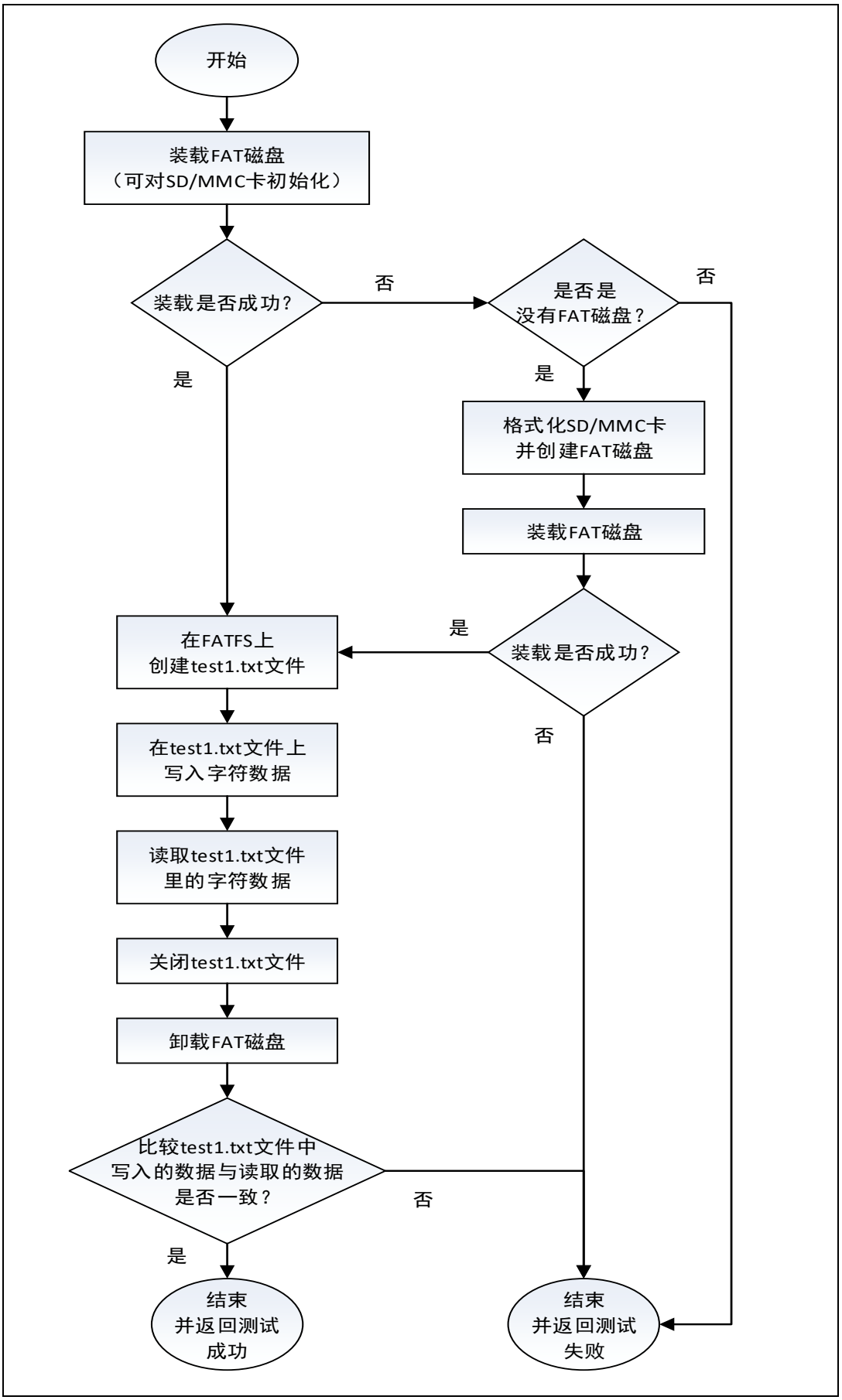
图 19. FATFS 文件系统相关文件



6.2 FATFS 文件系统案例

下图展示了 FATFS 文件系统案例的流程图。

图 20. FATFS 文件系统案例



7 案例 读/写 SD/MMC 卡

7.1 功能简介

演示初始化识别 SD 或 MMC 卡，并完成读写测试。

7.2 资源准备

1) 硬件环境：

对应产品型号的 AT-START BOARD

2) 软件环境：

project\at_start_f4xx\examples\sdio\sd_mmc_card

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

7.3 软件设计

1) 配置流程

- 开启 SDIO1 中断、开启 SDIO1 时钟。
- 初始化并识别 SD 或 MMC 卡。
- 若识别成功后，则打印 SD/MMC 卡的相关信息。
- 测试不同线宽下单块数据的读、写。
- 测试不同线宽下多块数据的读、写。
- 若识别到是 MMC 卡，则测试流数据的读、写。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  nvic_configuration();

  uart_print_init(115200);
  printf("start test..\r\n");

  /* 初始化并识别 SD/MMC 卡*/
  if(SD_OK != sd_init())
  {
    printf("sdio init fail\r\n");
    while(1)
```

```
    {
      sd_test_error();
    }
  }
  printf("sdio init ok\r\n");
  show_card_info();

  /* SD/MMC 卡单数据块测试 */
  if(TEST_SUCCESS != sd_single_block_test())
  {
    printf("sd card single block test fail\r\n");
    while(1)
    {
      sd_test_error();
    }
  }
  printf("sd card single block test ok\r\n");

  /* SD/MMC 卡多数据块测试 */
  if(TEST_SUCCESS != sd_multiple_blocks_test())
  {
    printf("sd card multiple blocks test fail\r\n");
    while(1)
    {
      sd_test_error();
    }
  }
  printf("sd card multiple blocks test ok\r\n");

  /* MMC 卡数据流传输测试 */
  if(sd_card_info.card_type == SDIO_MULTIMEDIA_CARD)
  {
    if(TEST_SUCCESS != mmc_stream_test())
    {
      printf("mmc card stream data test fail\r\n");
      while(1)
      {
        sd_test_error();
      }
    }
    printf("mmc card stream data test ok\r\n");
  }

  /* 所有测试通过，刷新 LED3 和 LED4 */
  while(1)
  {
```

```
      at32_led_toggle(LED3);
      at32_led_toggle(LED4);
      delay_ms(300);
    }
}
```

- 打印 SD/MMC 卡信息函数代码描述

```
void show_card_info(void)
{
  printf("---------------------\r\n");
  switch(sd_card_info.card_type)
  {
  case SDIO_STD_CAPACITY_SD_CARD_V1_1: printf("card type: SDSC V1.1\r\n");break;
  case SDIO_STD_CAPACITY_SD_CARD_V2_0: printf("card type: SDSC V2.0\r\n");break;
  case SDIO_HIGH_CAPACITY_SD_CARD:      printf("card type: SDHC V2.0\r\n");break;
  case SDIO_MULTIMEDIA_CARD:            printf("card type: MMC Card\r\n");break;
  }
  printf("card manufacturer_id: %d\r\n", sd_card_info.sd_cid_reg.manufacturer_id);
  printf("card rca: 0x%X\r\n", sd_card_info.rca);
  printf("card c_size = %u\r\n", sd_card_info.sd_csd_reg.device_size);
  printf("card capacity: %uMB %uKB\r\n", (uint32_t)(sd_card_info.card_capacity>>20),
(uint32_t)(sd_card_info.card_capacity>>10));
  printf("card block_size: %d\r\n", sd_card_info.card_blk_size);
  printf("---------------------\r\n");
}
```

- SD/MMC 卡测试错误函数代码描述

```
static void sd_test_error(void)
{
  at32_led_on(LED2);
  delay_ms(300);
  at32_led_off(LED2);
  delay_ms(300);
}
```

7.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 若有测试失败，会闪烁 LED2。
- 若所有测试通过后，会闪烁 LED3 和 LED4。

8 案例 FATFS 文件系统

8.1 功能简介

在 SD 或 MMC 卡上加载 FATFS 文件系统，并在该系统上创建、读写文件。

8.2 资源准备

1) 硬件环境：

对应产品型号的 AT-START BOARD

2) 软件环境：

project\at_start_f4xx\examples\sdio\sdio_fatfs

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

8.3 软件设计

1) 配置流程

- 加载 FATFS 文件系统，若 SD 或 MMC 卡上无文件系统，需格式化该卡。
- 若加载成功，则测试文件的创建及读写。
- 读取所加载文件系统的信息（包括总容量和剩余容量）。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();

  at32_board_init();

  nvic_configuration();

  uart_print_init(115200);
  printf("start test fatfs r0.14b..\r\n");

  if(TEST_SUCCESS != fatfs_test())
  {
    while(1)
    {
      sd_test_error();
    }
  }
```

```
  /* 所有测试通过，刷新 LED3 和 LED4 */
  while(1)
  {
    at32_led_toggle(LED3);
    at32_led_toggle(LED4);
    delay_ms(300);
  }
}
```

■ FATFS 文件系统测试函数代码描述

```
static test_result_type fatfs_test(void)
{
  FRESULT ret;
  char filename[] = "1:/test1.txt";
  const char wbuf[] = "this is my file for test fatfs!\r\n";
  char rbuf[50];
  UINT bytes_written = 0;
  UINT bytes_read = 0;
  DWORD fre_clust, fre_sect, tot_sect;
  FATFS* pt_fs;

  ret = f_mount(&fs, "1:", 1);
  if(ret){
    printf("fs mount err:%d.\r\n", ret);
    if(ret == FR_NO_FILESYSTEM){
      printf("create fatfs..\r\n");
      ret = f_mkfs("1:", 0, work, sizeof(work));
      if(ret){
        printf("creates fatfs err:%d.\r\n", ret);
        return TEST_FAIL;
      }
      else{
        printf("creates fatfs ok.\r\n");
      }
      ret = f_mount(NULL, "1:", 1);
      ret = f_mount(&fs, "1:", 1);
      if(ret){
        printf("fs mount err:%d.\r\n", ret);
        return TEST_FAIL;
      }
      else{
        printf("fs mount ok.\r\n");
      }
    }
    else{
```

```
        return TEST_FAIL;
    }
  }
  else{
    printf("fs mount ok.\r\n");
  }

  ret = f_open(&file, filename, FA_READ | FA_WRITE | FA_CREATE_ALWAYS);
  if(ret){
    printf("open file err:%d.\r\n", ret);
  }
  else{
    printf("open file ok.\r\n");
  }
  ret = f_write(&file, wbuf, sizeof(wbuf), &bytes_written);
  if(ret){
    printf("write file err:%d.\r\n", ret);
  }
  else{
    printf("write file ok, byte:%u.\r\n", bytes_written);
  }
  f_lseek(&file, 0);
  ret = f_read(&file, rbuf, sizeof(rbuf), &bytes_read);
  if(ret){
    printf("read file err:%d.\r\n", ret);
  }
  else{
    printf("read file ok, byte:%u.\r\n", bytes_read);
  }
  ret = f_close(&file);
  if(ret){
    printf("close file err:%d.\r\n", ret);
  }
  else{
    printf("close file ok.\r\n");
  }

  pt_fs = &fs;
  /* 获取磁盘“1：”的信息 */
  ret = f_getfree("1:", &fre_clust, &pt_fs);
  if(ret == FR_OK)
  {
    /* 获取总扇区和可用扇区的数目 */
    tot_sect = (pt_fs->n_fatent - 2) * pt_fs->csize;
    fre_sect = fre_clust * pt_fs->csize;
```

```
    printf("%10u KiB total drive space.\r\n%10u KiB available.\r\n", tot_sect / 2, fre_sect / 2);
  }

  ret = f_mount(NULL, "1:", 1);

  /* 比较写入文件的数据和读取该文件的数据是否一致 */
  if(1 == buffer_compare((uint8_t*)rbuf, (uint8_t*)wbuf, sizeof(wbuf))){
    printf("r/w file data test ok.\r\n");
  }
  else{
    printf("r/w file data test fail.\r\n");
    return TEST_FAIL;
  }

  return TEST_SUCCESS;
}
```

- SD/MMC 卡测试错误函数代码描述

```
static void sd_test_error(void)
{
  at32_led_on(LED2);
  delay_ms(300);
  at32_led_off(LED2);
  delay_ms(300);
}
```

8.4 实验效果

- 信息通过串口打印出来，在电脑上通过串口助手观看打印信息。
- 若有测试失败，会闪烁 LED2。
- 若所有测试通过后，会闪烁 LED3 和 LED4。

9 文档版本历史

表 10. 文档版本历史

日期	版本	变更
2022.02.07	2.0.0	最初版本

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。