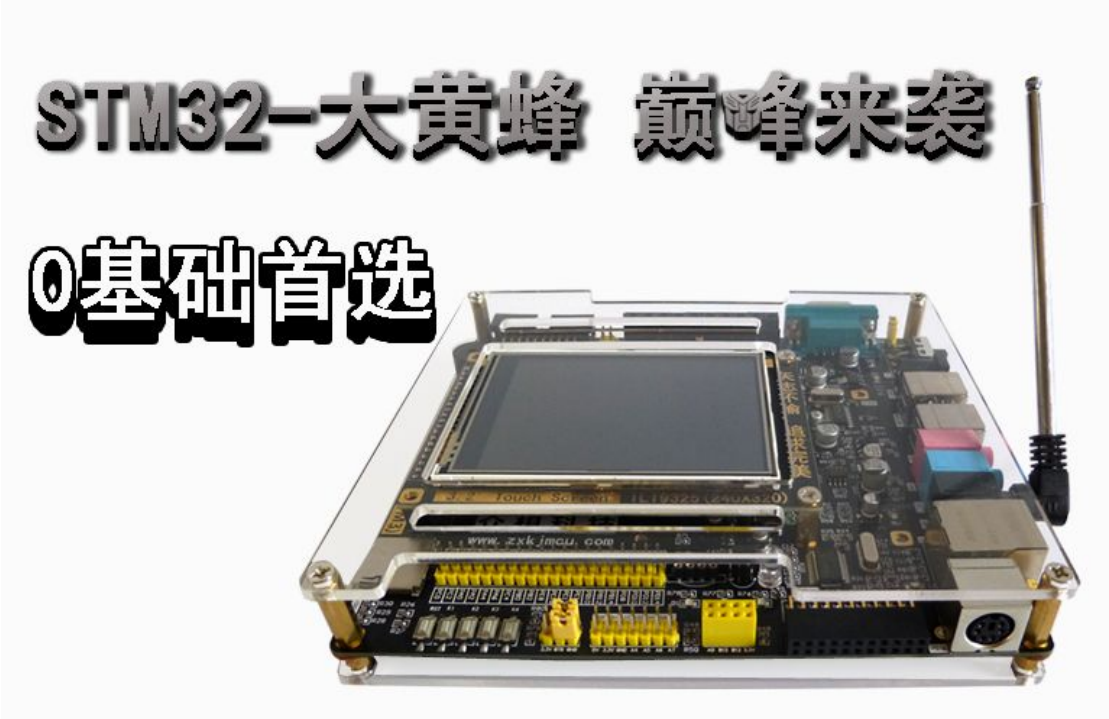
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.35 STM32 SD 存储卡指令协议

4.35.1 概述

4.35.1.1 SD 简介

SD 卡是基于 flash 的存储卡。

SD 卡和 MMC 卡的区别在于初始化过程不同。

SD 卡的通信协议包括 SD 总线和 SPI 两类。

SD 卡使用卡内智能控制模块进行 FLASH 操作控制，包括协议、数据存取、ECC 算法、缺陷处理和分析、电源管理、时钟管理。

通信电压范围：2.0-3.6V；工作电压范围:2.0-3.6V

最大读写速率：10Mbyte/s

最大 10 个堆叠的卡（20MHz,Vcc=2.7-3.6V)

SD 模式下允许有一个主机，多个从机(即多个卡)，主机可以给从机分配地址。主机发命令有些命令是发送给指定的从机，有些命令可以以广播形式发送。SD 模式下可以选择总线宽度，即选用几根 DAT 信号线，可以在主机初始化后设置。

4.35.1.2 SD 卡种类

MMC 卡: MultiMedia card，有 7 个触点(引脚)，分为两种操作模式，分别为 MMC 模式与 SPI 模式，两种模式对引脚的定义是不同的。SPI 模式只有 Host 具有 SPI 接口时才能使用。MMC 只具有存储功能，不像 SD 卡还具有加密功能。

SD 卡： Security Digtial card，共有 9 个触点(引脚)，多余的 2 个引脚为数据线，但使用与 MMC 卡兼容的模式时，这两个多余的引脚没有起到作用。SD 卡除了存储功能外，还有一种加密功能,但加密功能是收费的（所以开源的 linux 中只包含 mmc 的驱动目录），因为当初 SD 卡联盟中（索尼）就是发明这种卡就是用来存储音乐，并使用加密特性，防止拷贝。

TF 卡：软件上 SD 卡一致，只是在硬件的体积上比 SD 卡小，所以市场上很多的 TF 卡的 SD 外形卡套。

SDIO 卡： 这种卡并不是存储卡,可以理解为一个 SDIO 接口卡,如 WIFI(SDIO 接口)；并非 memory 卡，顾名思义，就是输入/ 输出卡，这种卡有用于 LAN 的、也有用于蓝牙的。

4.35.1.3 SD 卡通信接口

SD 卡有 9 个 pin：1 个 VDD,2 个 VSS(GND)，CLK,CMD,DATA0-DATA3, 【DATA3 可以作为卡检测脚】

SD 卡可以使用 SD 总线接口，也可以使用 SPI 通信接口；

4.35.2 SD 卡总线协议

SD 总线通信是基于命令和数据位流方式的，由一个起始位开始，以一个停止位结束：

命令——命令是开始开始操作的标记。命令从主机发送一个卡（寻址命令）或所有连接的卡（广播命令）。命令在 CMD 线上串行传送。

响应——响应是从寻址卡或所有连接的卡（同步）发送给主机用来响应接受到的命令的标记。

数据——数据可以通过数据线在卡和主机间双向传送。

总线上的通信是通过传送命令和数据实现。

在多媒体卡/SD/SD I/O 总线上的基本操作是命令/响应结构，这样的总线操作在命令或总线机制下实现信息交换；另外，某些操作还具有数据令牌。

在 SD 存储器卡上传送的数据是以数据块的形式传输；在 MMC 上传送的数据是以数据块或数据流的形式传输；在 CE-ATA 设备上传送的数据也是以数据块的形式传输。

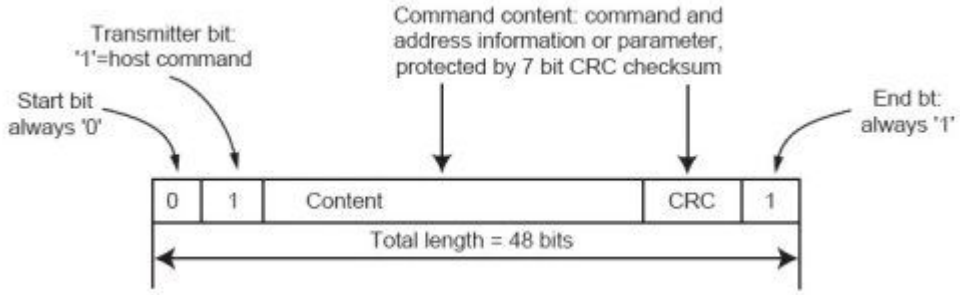


图 4.35.1 命令数据操作

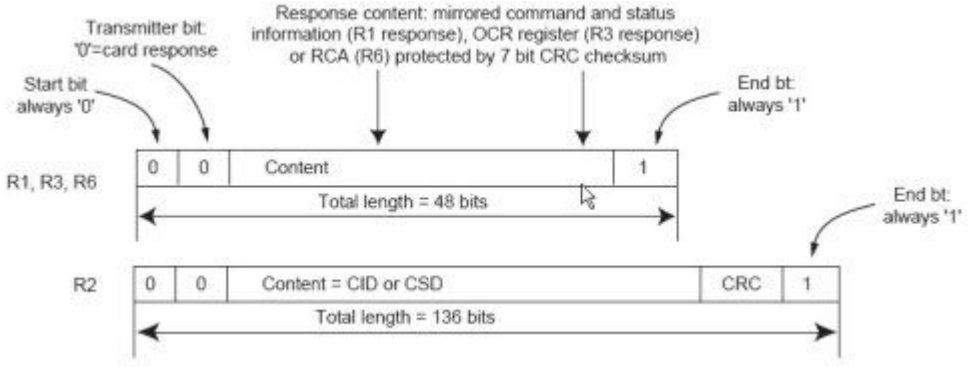


图 4.35.2 响应数据操作

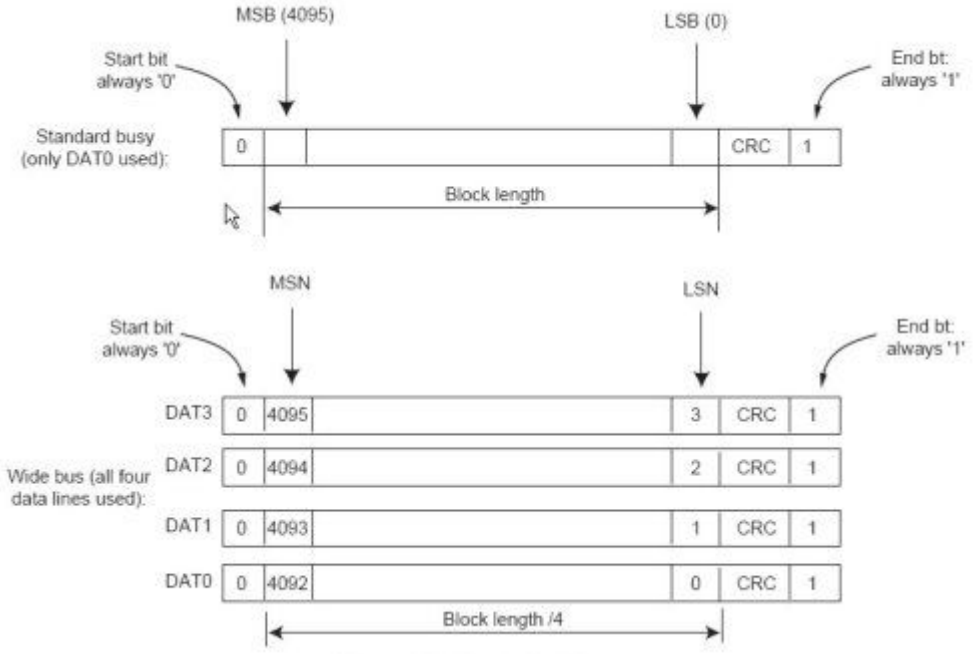


图 4.35.3 数据格式

在命令行中，MSB 位首先传送，LSB 位最后传送。

当使用宽总线模式时，数据同时在 4 根数据线上传输。开始位、结束位和 CRC 在每根数据线上传送。CRC 对每根数据线单独计算。CRC 状态响应和 Busy 信号只通过 DAT0 由卡发送给主机。

最多支持 64 个命令：CMD0~CMD63，(其中 CMD57~63 是保留的)

ACMD：Application Sepcific command：ACMD41 =cmd55 + cmd41,组合命令，CMD55 是前导命令，提醒卡后面的 CMD41 是一个特殊的命令。

4.35.3 SD 卡指令数据包

SD 卡的指令被封装成 48 位的数据包，每次传送这 48 位的数据包。数据包的内容包括起始位、结束位、传输位、命令索引、传输参数和 7 位 CRC 校验码。其具体格式分布如附表 1 数据包位定义。

附表 1 数据包位定义

Bit 位置	47	46	[45:40]	[39:08]	[07:01]	00
Bit 宽度	1	1	6	32	7	1
值	0	1	x	x	X	1
说明	起始位	传输位	命令索引	传输参数	CRC 校验码	结束位

其中的命令索引位是[45：40]，里面可以封装各种命令，具体的命令表将在下面给出。不同的命令会对应不同的回应(respond)，回应有三种(R1，R2，R3)格式，在命令表中的选项会给出。

4.35.4 协议功能描述

所有主机和 SD 卡间的通信由主机控制。主机发送下述两类命令：

广播命令：广播命令发送给所有 SD 卡，有些命令需要响应。

寻址（点对点）命令：寻址命令只发送给具有相应地址的卡，并需要从卡返回一个响应。对卡而言也有两类操作：

卡识别模式——在重置（reset）后当主机查找总线上的新卡时，处于卡识别模式。重置后 SD 卡将始终处于该模式，直到收到 SEND_RCA 命令（CMD3）。

数据传输模式——一旦卡的 REC 发布后，将进入数据传输模式。主机一旦识别了所有总线上的卡后，将进入数据传输模式。

4.35.4.1 卡识别模式

在卡识别模式，主机重置所有处于卡识别模式的 SD 卡，检验操作电压范围，识别卡并请求卡发送相对卡地址 RCA。操作对每个卡在各自的 CMD 线上单独进行，所有的数据传送只使用 CMD 线。

4.35.5 重置

GO_IDLE_STATE(CMD0)是软件重置命令，设置每个 SD 卡进入 Idle 状态。处于 Inactive 状态的卡不受此命令影响。主机上电后，所有 SD 卡进入 Idle 状态，包括处于 Inactive 状态的卡。至少 74 个时钟周期后才能开始总线传输。上电或 CMD0 后，所有 SD 卡的命令线处于输入模式，等待下一个命令的起始位。卡通过一个默认的相对卡地址 RCA（RCA=0x0000）和默认驱动寄存器设置（最低速，最高驱动电流）初始化。

4.35.5.1 卡识别过程

在识别时钟速率 fOD 下主机开始卡识别过程。SD 卡的 CMD 线输出驱动是 push-pull 驱动。总线激活后，主机要求卡发送它们的有效操作条件（ACMD41 preceding with APP_CMD — CMD55 with RCA=0x0000）。ACMD41 命令的响应是卡的操作条件寄存器。相同的命令将发送给系统中所有的卡。不兼容的卡将进入 Inactive 状态。主机然后发送命令 ALL_SEND_CID（CMD2）到每个卡以获取每个卡的唯一标识 CID 号。未识别的卡通过 CMD 线发送 CID 号作为响应。当卡发送 CID 号后，进入识别状态（Identification State）。此后，主机发送 CMD3（SEND_RELATIVE_ADDR）要求卡发布一个新的相对卡地址 RCA，地址比 CID 短，在以后的数据传输模式中用来寻址卡。一旦获得 RCA 后，卡状态变成就绪状态（Stand-by state）。此时，如果主机要求卡换成其他的 RCA 号，可以通过发送另一个 SEND_RELATIVE_ADDR 命令给卡，要求发布一个新的 RCA，最后发布的 RCA 是实际使用的 RCA。主机对系统中的每个卡重复识别过程。所有的 SD 卡初始化完以后，系统将开始初始化 MMC 卡（如果有的话），使用 MMC 卡的 CMD2 和 CMD3。

4.35.6 数据传输模式

直到主机知道所有 CSD 寄存器的内容，fpp 时钟速率必须保持在 fOD, 因为一些卡有操作频率限制。主机发送 SEND_CSD（CMD9）获取卡定义数据（Card Specific Data，CSD 寄存器），如块大小、卡存储容量、最大时钟速率等。CMD7 用来选择一个卡并将它置于传输状态（Transfer state），在任何时间只能有一个卡处于传输状态。如果已有一个卡处于传输状态，它和主机的连接将释放，并返回到 Stand-by 状态。当 CMD7 以保留相对地址“0x0000”发送时，所有卡将返回到 Stand-by 状态。这可以用来识别新的卡而不重置其他已注册的卡。在这种状态下已有一个 RCA 地址的卡不响应识别命令（ACMD41,CMD2,CMD3）。

注意：当卡接收到一个带有不匹配 RCA 的 CMD7 时，卡将取消选中。在公用 CMD 线时，选中一个卡时将自动不选中其他卡。因此，在 SD 卡系统中，主机具有如下功能：

初始化完成后，在公用 CMD 线时，不选中卡是自动完成的。如果使用单独的 CMD 线，需要关注不选中卡的操作 在主机和选择的 SD 卡之间的所有数据通信是点对点的方式。所有寻址命令都需要响应。

不同数据传输模式的关系如图 4-8 所示，使用如下步骤：

所有读数据命令可以在任何时候通过停止命令（stop command,CMD12）中止。数据传输将中止，卡回到传输状态（Transfer State）。读命令有：块读命令（CMD17），多块读命令（CMD18），发送读保护（CMD30），发送 scr(ACMD51)，以及读模式的通用命令(CMD56)。

所有写数据命令可以在任何时候通过停止命令（stop command,CMD12）

中止。在不选中卡命令 CMD7 前写命令必须停止。写命令有：块写命令 (CMD24 and CMD25), 写 CID (CMD26), 写 CSD(CMD27), lock/unlock 命令(CMD42) 以及写模式通用命令(CMD56)。

一旦数据传输完成，卡将退出数据写状态并进入 Programming State(传输成功)或 Transfer State（传输失败）。

如果一个快写操作停止，而且最后一块块长度和 CRC 是有效的，那么数据可以被操作（programmed）。 卡可能提供块写缓冲。这意味着在前一块数据被操作时，下一块数据可以传送给卡。如果所有卡写缓冲已满，只要卡在 Programming State，DAT0 将保持低电平（BUSY）。 写 CSD、CID、写保护和擦除时没有缓冲。这表明在卡因这些命令而处于忙时，不再接收其他数据传输命令。在卡忙时 DAT0 保持低电平，并处于 Programming State。实际上如果 CMD 和 DAT0 线分离，而且主机占有的忙 DAT0 线和其他 DAT0 线分开，那么在卡忙时，主机可以访问其他卡。

在卡被编程（programming）时，禁止参数设置命令。参数设置命令包括：设置块长度（CMD16），擦除块开始(CMD32)和擦除块结束（CMD33）。卡在操作时不允许读命令。

使用 CMD7 指令把另一个卡从 Stand-by 状态转移到 Transfer 状态不会中止擦除和编程（programming）操作。卡将切换到 Disconnect 状态并释放 DAT 线。 使用 CMD7 指令可以不选中处于 Disconnect 状态的卡。卡将进入 Programming 状态，重新激活忙指示。

使用 CMD0 或 CMD15 重置卡将中止所有挂起和活动的编程（programming）操作。这可能会破坏卡上的数据内容，需要主机保证避免

这样的操作。

4.35.7 时钟控制

SD 卡主机可以使用 SD 卡总线时钟信号设置卡进入节能模式或控制总线上的数据流。主机可以降低时钟频率或直接关闭。

SD 卡主机必须遵循下列约束：

总线频率可以在任何时候改变（满足最大和最小值的约束）。

ACMD41(SD_APP_OP_COND)是一个例外。发送 ACMD41 命令后，主机将执行下面步骤 1 和步骤 2 直到卡进入就绪状态：

1) 持续发送 100KHZ-400KHZ 之间的时钟频率。

2) 如果主机要停止时钟，通过 ACMD41 命令以小于 50ms 的间隔设置 busy 位。

4.35.8 命令类型

共有四类用来控制 SD 卡的命令：

- 广播命令（bc），无响应——广播命令只有在所有 CMD 线一起连接到主机时才能使用。如果分开连接，那么每个卡将单独接收命令。
- 带响应的广播命令（bcr）——所有卡同时响应。因为 SD 卡没有开漏模式，这个命令只有在所有的 CMD 线分开时采用使用。该命令将被每个卡分别接收和响应。（OPEN DRAIN 输出只能做输出口，当外部无上拉电阻时，该口为高阻状态。只有外部有上拉电阻时，才有可能输出高或低的电平。）
- 寻址（点对点）命令（ac）——DAT 上没有数据传输。寻址（点对点）
- 数据传输命令（adtc）——DAT 上传输数据。所有的命令和响应通过 CMD

线传输。

4.35.9 响应

所有响应通过 CMD 线传输，响应以 MSB 开始，不同类型的响应长度根据类型不同而不同。响应以起始位开始（通常为“0”），接着这是传输方向的位（卡为 0）。除了 R3 外其他响应都有 CRC。每个响应都以结束位（通常为“1”）结束。

共有四类响应，格式分别为：

❖ R1(标准响应)：长度 48 位 。Bits[45:40]指示被响应的命令索引号。如果有到卡的数据传输，每个数据块传输后数据线上都会出现忙信号。主机在数据块传输后检测忙信号。

附表 2 R1 组合

Bit 位置	47	46	[45:40]	[39:08]	[07:01]	00
Bit 宽度	1	1	6	32	7	1
值	“0”	“0”	x	x	x	“1”
说明	起始位	传输位	命令索引	卡状态	CRC 校验码	结束位

❖ Rb1 和 R1 相同，带有一个可选的忙信号传输。根据接收到命令前的状态和接收到的命令可能变成忙。主机可以在响应时检测忙信号。

❖ R2(CID,CSD)：响应长度为 136 位 。CID 寄存器内容作为 CMD2 和 CMD10 的响应发送。CSD 寄存器内容作为 CMD9 的响应发送。只传输 CID 和 CSD 的[127…1]位，寄存器的[0]位被响应的结束位取代。

附表 3 R2 组合

Bit 位置	135	134	[133:128]	[127:01]	00
Bit 宽度	1	1	6	127	1
值	“0”	“0”	“111111”	x	“1”
说明	起始位	传输位	保留	CID 或 CSD 寄存器内容和 CRC 校验	结束位

返回 CID（Card Identification）或 CSD（Card Specific Data）寄存器的内容。

4.35.10 SD 卡初始化与数据传输

一、前文我们介绍过 SD 卡数据传世模式：

- ❖ 无活动模式：空闲状态
- ❖ 卡识别模式：主机被复位或者在总线上寻找新卡时，主机处于该状态下。卡在复位以后和收到 SEND_RCA 命令以前都处于此模式下。
- ❖ 数据传输模式：卡在它们的 RCA 第一次发布后进入数据传输模式。主机识别总线上所有的卡后进入数据传输模式。

二、SD 卡状态和操作模式的对应关系

附表 4 卡状态和操作模式的对应关系

卡状态	操作模式
无活动状态	无活动
空闲态	卡识别模式
准备态	
识别态	
等待态	数据传输模式
传输态	
发送数据态	
接收数据态	
编程态	
断开态	

上面表说明卡的状态与操作模式之间的依赖关系，SD 的每种状态都关联一种操作模式，其状态图将在随后进行介绍。

三、卡的初始化和识别处理

当总线被激活后，主机就开始卡的初始化和识别处理。初始化处理设置它的操作状态和设置 OCR 中的 HCS 比特位命令 SD_SEND_OP_COND（ACMD41）开始。HCS 比特位被设置为 1 表示主机支持高容量 SD 卡。HCS 被设置为 0 表示主机不支持高容量 SD 卡。

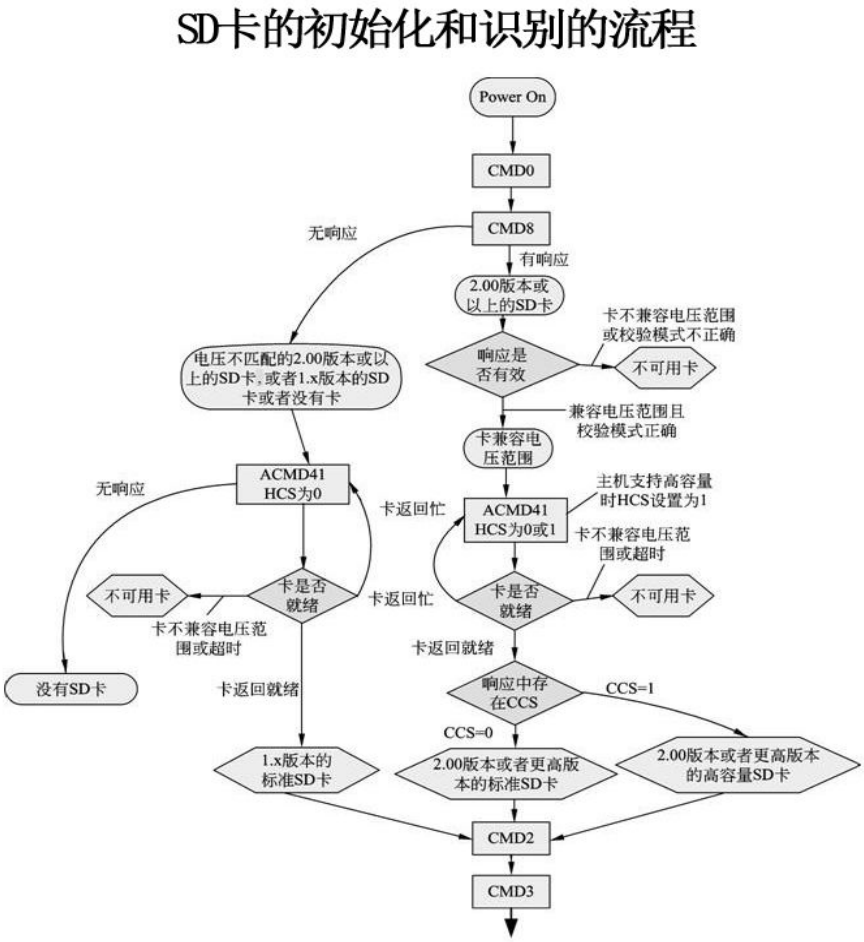


图 4.35.4　SD 卡的初始化和识别的流程

4.35.11　数据传输模式

卡的识别模式结束后，主机时钟 fpp（数据传输时钟速率）将保持为 fOD（卡识别模式下的时钟），因为有些卡对操作时钟有限制。主机必须发送 SEND_CSD（CMD9）来获得卡规格数据寄存器内容，如块大小、卡容量等。广播命令 SET_DSR（CMD4）配置所有识别卡的驱动阶段。它对 DSR 寄存器进行编程以适应应用总线布局、总线上的卡数目和数据传输频率。

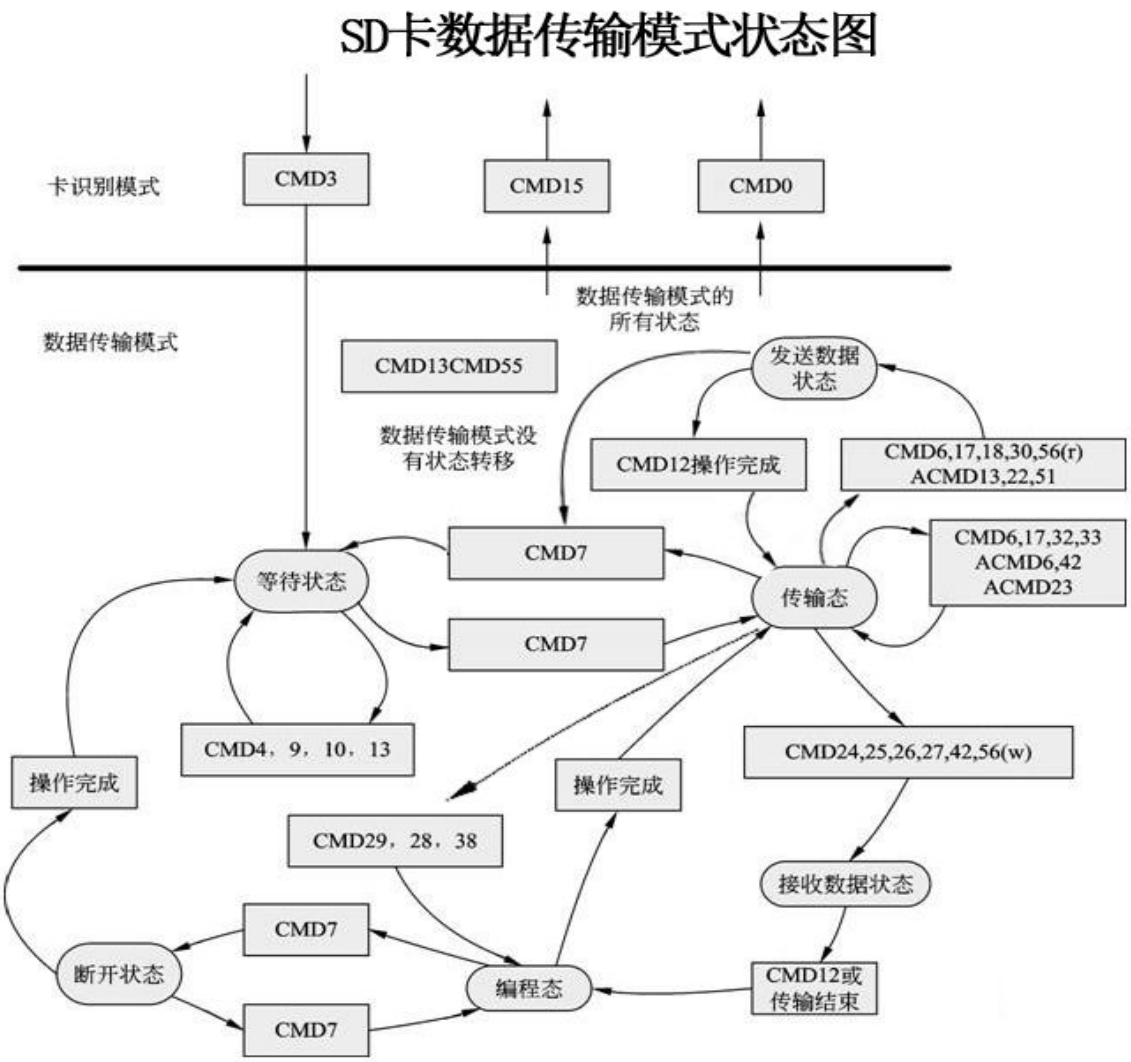


图 4.35.5 SD 卡数据传输模式状态图

4.35.12 实验目的

我们这节实验的目的是把 SD 卡的信息都读取出来，通过串口程序打印到屏幕上显示出来。包括 SD 卡的容量、厂家、ID 号等基础信息。

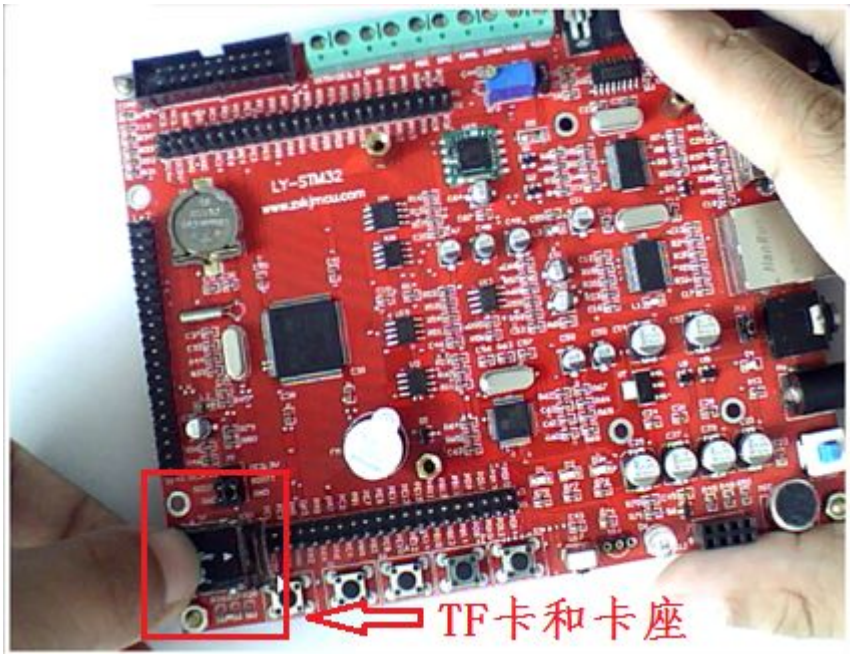


图 4.35.6 SD 卡位置

4.35.13 硬件设计

SD 卡（TF 卡）和 CPU 之间采用的是 SPI 通讯方式，从硬件电路设计上可以看出，SD 卡（TF 卡）集成在大黄蜂实验板上。具体的引脚规定见下图。

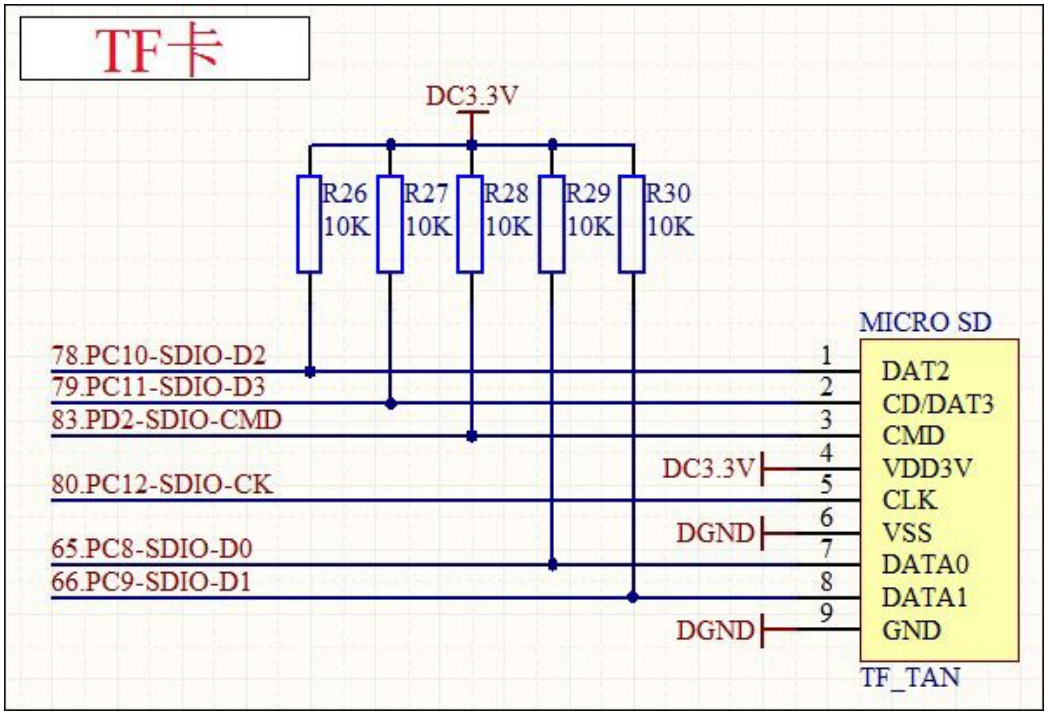


图 4.35.7 SD 卡原理图

4.35.14 软件设计

4.35.14.1 软件设计说明

1、采用 SPI 通讯方式。

2、这套程序严格按照 SPI 的工作时序编写。

3、大家要掌握模板形成的方法，严格讲模板没有固定的模式，每个人都有自己独立思考，要从结构上考虑才好。

在这节程序设计中，用到了外部中断函数；USART 串口通讯函数；定时器函数等。

4.35.14.2 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
stm32f10x_dma.c
stm32f10x_sdio.c
```

以上库文件包含了本次实验所有要用到的函数功能。

4.35.14.3 自定义头文件

```
pbdata.h
pbdata.c
```

我们已经创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.35.14.4 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
```

```
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_dma.h"  //自定义的 stm32_fsmc 专属函数
#include "stm32f10x_sdio.h" //自定义的 stm32_fsmc 专属函数
#include "stdio.h"
#include "sdio_sdcard.h"

//定义变量

extern u8 dt;

//定义函数

void RCC_HSE_Configuration(void);
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。“stm32_fsmc.h”和“lcd_ILI9325.h”是我们自定义的，为了是在大程序设计中方便移植和功能分类而独立新建。

4.35.14.5 pbdata.c 文件里的内容是

下面是 pbdata.c 文件详细内容，其中的内容和以前的内容一致，没什么变化，在这就省略程序代码，具体可以引用《4.32 STM32 外设篇-LCD 彩色液晶屏显示汉字、英文、数字程序设计》中的 pbdata.c 文件代码。

4.35.15 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟，这点很重要。

4.35.16 GPIO 引脚时钟使能

```
        SystemInit();//72m
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟
使能
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟
使能
          }
```

本节实验用到了 PA 端口，所以要打开这个端口的使能；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.35.17 stm32f10x_it.c 文件里的内容

在中断处理 stm32f10x_it.c 文件里中有串口 1 子函数。

```
    /* Includes
------------------------------------------------------------------*/
    #include "stm32f10x_it.h"
    #include "stm32f10x_exti.h"
    #include "stm32f10x_rcc.h"
    #include "misc.h"
    #include "pbdata.h"

    void NMI_Handler(void)
    {
    }

    void USART1_IRQHandler(void)
    {
       if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
       {
          USART_SendData(USART1,USART_ReceiveData(USART1));
          while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
       }
    }
```

```
/*******************************************************************************
* 名    称：void EXTI3_IRQHandler(void)
* 功    能：EXTI3 中断处理程序
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void EXTI3_IRQHandler(void)
{

}
//当数据传输结束时产生中断
void SDIO_IRQHandler(void)
{
  SD_ProcessIRQSrc();
}
```

4.35.18 sdio_sdcard.h 文件里的内容

程序 sdio_sdcard.h 文件内容。

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __SDIO_SDCARD_H
#define __SDIO_SDCARD_H

/* Includes
------------------------------------------------------------------*/
#include "stm32f10x.h"

/* Exported types
------------------------------------------------------------*/
typedef enum
{
/**
  * @brief  SDIO specific error defines
  */
  SD_CMD_CRC_FAIL                    = (1), /*!< Command response received
(but CRC check failed) */
  SD_DATA_CRC_FAIL                   = (2), /*!< Data bock sent/received
(CRC check Failed) */
```

```
      SD_CMD_RSP_TIMEOUT                    = (3), /*!< Command response timeout
*/
      SD_DATA_TIMEOUT                       = (4), /*!< Data time out */
      SD_TX_UNDERRUN                        = (5), /*!< Transmit FIFO under-run */
      SD_RX_OVERRUN                         = (6), /*!< Receive FIFO over-run */
      SD_START_BIT_ERR                      = (7), /*!< Start bit not detected on
all data signals in widE bus mode */
      SD_CMD_OUT_OF_RANGE                   = (8), /*!< CMD's argument was out of
range.*/
      SD_ADDR_MISALIGNED                    = (9), /*!< Misaligned address */
      SD_BLOCK_LEN_ERR                      = (10), /*!< Transferred block length
is not allowed for the card or the number of transferred bytes does not match
the block length */
      SD_ERASE_SEQ_ERR                      = (11), /*!< An error in the sequence
of erase command occurs.*/
      SD_BAD_ERASE_PARAM                    = (12), /*!< An Invalid selection for
erase groups */
      SD_WRITE_PROT_VIOLATION               = (13), /*!< Attempt to program a
write protect block */
      SD_LOCK_UNLOCK_FAILED                 = (14), /*!< Sequence or password
error has been detected in unlock command or if there was an attempt to access
a locked card */
      SD_COM_CRC_FAILED                     = (15), /*!< CRC check of the previous
command failed */
      SD_ILLEGAL_CMD                        = (16), /*!< Command is not legal for
the card state */
      SD_CARD_ECC_FAILED                    = (17), /*!< Card internal ECC was
applied but failed to correct the data */
      SD_CC_ERROR                           = (18), /*!< Internal card controller
error */
      SD_GENERAL_UNKNOWN_ERROR              = (19), /*!< General or Unknown error
*/
      SD_STREAM_READ_UNDERRUN               = (20), /*!< The card could not
sustain data transfer in stream read operation. */
      SD_STREAM_WRITE_OVERRUN               = (21), /*!< The card could not
sustain data programming in stream mode */
      SD_CID_CSD_OVERWRITE                  = (22), /*!< CID/CSD overwrite error
*/
      SD_WP_ERASE_SKIP                      = (23), /*!< only partial address
space was erased */
      SD_CARD_ECC_DISABLED                  = (24), /*!< Command has been executed
without using internal ECC */
      SD_ERASE_RESET                        = (25), /*!< Erase sequence was
cleared before executing because an out of erase sequence command was received
```

```
*/
      SD_AKE_SEQ_ERROR                    = (26), /*!< Error in sequence of
authentication. */
      SD_INVALID_VOLTRANGE                = (27),
      SD_ADDR_OUT_OF_RANGE                = (28),
      SD_SWITCH_ERROR                     = (29),
      SD_SDIO_DISABLED                    = (30),
      SD_SDIO_FUNCTION_BUSY               = (31),
      SD_SDIO_FUNCTION_FAILED             = (32),
      SD_SDIO_UNKNOWN_FUNCTION            = (33),

    /**
      * @brief  Standard error defines
      */
      SD_INTERNAL_ERROR,
      SD_NOT_CONFIGURED,
      SD_REQUEST_PENDING,
      SD_REQUEST_NOT_APPLICABLE,
      SD_INVALID_PARAMETER,
      SD_UNSUPPORTED_FEATURE,
      SD_UNSUPPORTED_HW,
      SD_ERROR,
      SD_OK = 0
    } SD_Error;

    /**
      * @brief  SDIO Transfer state
      */
    typedef enum
    {
      SD_TRANSFER_OK  = 0,
      SD_TRANSFER_BUSY = 1,
      SD_TRANSFER_ERROR
    } SDTransferState;

    /**
      * @brief  SD Card States
      */
    typedef enum
    {
      SD_CARD_READY                  = ((uint32_t)0x00000001),
      SD_CARD_IDENTIFICATION         = ((uint32_t)0x00000002),
      SD_CARD_STANDBY                = ((uint32_t)0x00000003),
      SD_CARD_TRANSFER               = ((uint32_t)0x00000004),
```

```
    SD_CARD_SENDING                 = ((uint32_t)0x00000005),
    SD_CARD_RECEIVING               = ((uint32_t)0x00000006),
    SD_CARD_PROGRAMMING             = ((uint32_t)0x00000007),
    SD_CARD_DISCONNECTED            = ((uint32_t)0x00000008),
    SD_CARD_ERROR                   = ((uint32_t)0x000000FF)
  }SDCardState;


  /**
    * @brief  Card Specific Data: CSD Register
    */
  typedef struct
  {
    __IO uint8_t  CSDStruct;            /*!< CSD structure */
    __IO uint8_t  SysSpecVersion;       /*!< System specification version */
    __IO uint8_t  Reserved1;            /*!< Reserved */
    __IO uint8_t  TAAC;                 /*!< Data read access-time 1 */
    __IO uint8_t  NSAC;                 /*!< Data read access-time 2 in CLK
cycles */
    __IO uint8_t  MaxBusClkFrec;        /*!< Max. bus clock frequency */
    __IO uint16_t CardComdClasses;      /*!< Card command classes */
    __IO uint8_t  RdBlockLen;           /*!< Max. read data block length */
    __IO uint8_t  PartBlockRead;        /*!< Partial blocks for read allowed
*/
    __IO uint8_t  WrBlockMisalign;      /*!< Write block misalignment */
    __IO uint8_t  RdBlockMisalign;      /*!< Read block misalignment */
    __IO uint8_t  DSRImpl;              /*!< DSR implemented */
    __IO uint8_t  Reserved2;            /*!< Reserved */
    __IO uint32_t DeviceSize;           /*!< Device Size */
    __IO uint8_t  MaxRdCurrentVDDMin;   /*!< Max. read current @ VDD min */
    __IO uint8_t  MaxRdCurrentVDDMax;   /*!< Max. read current @ VDD max */
    __IO uint8_t  MaxWrCurrentVDDMin;   /*!< Max. write current @ VDD min */
    __IO uint8_t  MaxWrCurrentVDDMax;   /*!< Max. write current @ VDD max */
    __IO uint8_t  DeviceSizeMul;        /*!< Device size multiplier */
    __IO uint8_t  EraseGrSize;          /*!< Erase group size */
    __IO uint8_t  EraseGrMul;           /*!< Erase group size multiplier */
    __IO uint8_t  WrProtectGrSize;      /*!< Write protect group size */
    __IO uint8_t  WrProtectGrEnable;    /*!< Write protect group enable */
    __IO uint8_t  ManDeflECC;           /*!< Manufacturer default ECC */
    __IO uint8_t  WrSpeedFact;          /*!< Write speed factor */
    __IO uint8_t  MaxWrBlockLen;        /*!< Max. write data block length */
    __IO uint8_t  WriteBlockPaPartial;  /*!< Partial blocks for write allowed
*/
    __IO uint8_t  Reserved3;            /*!< Reserded */
```

```
    __IO uint8_t  ContentProtectAppli;  /*!< Content protection application
*/
    __IO uint8_t  FileFormatGrouop;     /*!< File format group */
    __IO uint8_t  CopyFlag;             /*!< Copy flag (OTP) */
    __IO uint8_t  PermWrProtect;        /*!< Permanent write protection */
    __IO uint8_t  TempWrProtect;        /*!< Temporary write protection */
    __IO uint8_t  FileFormat;           /*!< File Format */
    __IO uint8_t  ECC;                  /*!< ECC code */
    __IO uint8_t  CSD_CRC;              /*!< CSD CRC */
    __IO uint8_t  Reserved4;            /*!< always 1*/
  } SD_CSD;

  /**
    * @brief  Card Identification Data: CID Register
    */
  typedef struct
  {
    __IO uint8_t  ManufacturerID;       /*!< ManufacturerID */
    __IO uint16_t OEM_AppliID;          /*!< OEM/Application ID */
    __IO uint32_t ProdName1;            /*!< Product Name part1 */
    __IO uint8_t  ProdName2;            /*!< Product Name part2*/
    __IO uint8_t  ProdRev;              /*!< Product Revision */
    __IO uint32_t ProdSN;               /*!< Product Serial Number */
    __IO uint8_t  Reserved1;            /*!< Reserved1 */
    __IO uint16_t ManufactDate;         /*!< Manufacturing Date */
    __IO uint8_t  CID_CRC;              /*!< CID CRC */
    __IO uint8_t  Reserved2;            /*!< always 1 */
  } SD_CID;

  /**
    * @brief SD Card Status
    */
  typedef struct
  {
    __IO uint8_t DAT_BUS_WIDTH;
    __IO uint8_t SECURED_MODE;
    __IO uint16_t SD_CARD_TYPE;
    __IO uint32_t SIZE_OF_PROTECTED_AREA;
    __IO uint8_t SPEED_CLASS;
    __IO uint8_t PERFORMANCE_MOVE;
    __IO uint8_t AU_SIZE;
    __IO uint16_t ERASE_SIZE;
    __IO uint8_t ERASE_TIMEOUT;
    __IO uint8_t ERASE_OFFSET;
```

```
} SD_CardStatus;

/**
  * @brief SD Card information
  */
typedef struct
{
  SD_CSD SD_csd;
  SD_CID SD_cid;
  uint32_t CardCapacity;  /*!< Card Capacity */
  uint32_t CardBlockSize; /*!< Card Block Size */
  uint16_t RCA;
  uint8_t CardType;
} SD_CardInfo;

/*宏定义*/
#define SDIO_FIFO_ADDRESS                ((uint32_t)0x40018080)
//SDIO_FIOF 地址=SDIO 地址+0x80 至 sdio 地址+0xfc
/**
  * @brief  SDIO Intialization Frequency (400KHz max)
  */
#define SDIO_INIT_CLK_DIV                ((uint8_t)0xB2)
/**
  * @brief  SDIO Data Transfer Frequency (25MHz max)
  */
/*!< SDIOCLK = HCLK, SDIO_CK = HCLK/(2 + SDIO_TRANSFER_CLK_DIV) */
#define SDIO_TRANSFER_CLK_DIV            ((uint8_t)0x01)


/**
  * @brief SDIO Commands  Index
  */
#define SD_CMD_GO_IDLE_STATE                       ((uint8_t)0)
#define SD_CMD_SEND_OP_COND                        ((uint8_t)1)
#define SD_CMD_ALL_SEND_CID                        ((uint8_t)2)
#define SD_CMD_SET_REL_ADDR                        ((uint8_t)3) /*!<
SDIO_SEND_REL_ADDR for SD Card */
#define SD_CMD_SET_DSR                             ((uint8_t)4)
#define SD_CMD_SDIO_SEN_OP_COND                    ((uint8_t)5)
#define SD_CMD_HS_SWITCH                           ((uint8_t)6)
#define SD_CMD_SEL_DESEL_CARD                      ((uint8_t)7)
#define SD_CMD_HS_SEND_EXT_CSD                     ((uint8_t)8)
#define SD_CMD_SEND_CSD                            ((uint8_t)9)
#define SD_CMD_SEND_CID                            ((uint8_t)10)
```

```
    #define SD_CMD_READ_DAT_UNTIL_STOP                  ((uint8_t)11) /*!< SD
Card doesn't support it */
    #define SD_CMD_STOP_TRANSMISSION                    ((uint8_t)12)
    #define SD_CMD_SEND_STATUS                          ((uint8_t)13)
    #define SD_CMD_HS_BUSTEST_READ                      ((uint8_t)14)
    #define SD_CMD_GO_INACTIVE_STATE                    ((uint8_t)15)
    #define SD_CMD_SET_BLOCKLEN                         ((uint8_t)16)
    #define SD_CMD_READ_SINGLE_BLOCK                    ((uint8_t)17)
    #define SD_CMD_READ_MULT_BLOCK                      ((uint8_t)18)
    #define SD_CMD_HS_BUSTEST_WRITE                     ((uint8_t)19)
    #define SD_CMD_WRITE_DAT_UNTIL_STOP                 ((uint8_t)20) /*!< SD
Card doesn't support it */
    #define SD_CMD_SET_BLOCK_COUNT                      ((uint8_t)23) /*!< SD
Card doesn't support it */
    #define SD_CMD_WRITE_SINGLE_BLOCK                   ((uint8_t)24)
    #define SD_CMD_WRITE_MULT_BLOCK                     ((uint8_t)25)
    #define SD_CMD_PROG_CID                             ((uint8_t)26) /*!<
reserved for manufacturers */
    #define SD_CMD_PROG_CSD                             ((uint8_t)27)
    #define SD_CMD_SET_WRITE_PROT                       ((uint8_t)28)
    #define SD_CMD_CLR_WRITE_PROT                       ((uint8_t)29)
    #define SD_CMD_SEND_WRITE_PROT                      ((uint8_t)30)
    #define SD_CMD_SD_ERASE_GRP_START                   ((uint8_t)32) /*!< To
set the address of the first write
                                                                    block
to be erased. (For SD card only) */
    #define SD_CMD_SD_ERASE_GRP_END                     ((uint8_t)33) /*!< To
set the address of the last write block of the

continuous range to be erased. (For SD card only) */
    #define SD_CMD_ERASE_GRP_START                      ((uint8_t)35) /*!< To
set the address of the first write block to be erased.
                                                                    (For
MMC card only spec 3.31) */

    #define SD_CMD_ERASE_GRP_END                        ((uint8_t)36) /*!< To
set the address of the last write block of the

continuous range to be erased. (For MMC card only spec 3.31) */

    #define SD_CMD_ERASE                                ((uint8_t)38)
    #define SD_CMD_FAST_IO                              ((uint8_t)39) /*!< SD
Card doesn't support it */
    #define SD_CMD_GO_IRQ_STATE                         ((uint8_t)40) /*!< SD
```

```
Card doesn't support it */
    #define SD_CMD_LOCK_UNLOCK                          ((uint8_t)42)
    #define SD_CMD_APP_CMD                              ((uint8_t)55)
    #define SD_CMD_GEN_CMD                              ((uint8_t)56)
    #define SD_CMD_NO_CMD                               ((uint8_t)64)

    /**
      * @brief Following commands are SD Card Specific commands.
      *        SDIO_APP_CMD : CMD55 should be sent before sending these
commands.
      */
    #define SD_CMD_APP_SD_SET_BUSWIDTH                  ((uint8_t)6)  /*!< For
SD Card only */
    #define SD_CMD_SD_APP_STAUS                         ((uint8_t)13) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS         ((uint8_t)22) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_OP_COND                       ((uint8_t)41) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SET_CLR_CARD_DETECT           ((uint8_t)42) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SEND_SCR                      ((uint8_t)51) /*!< For
SD Card only */
    #define SD_CMD_SDIO_RW_DIRECT                       ((uint8_t)52) /*!< For
SD I/O Card only */
    #define SD_CMD_SDIO_RW_EXTENDED                     ((uint8_t)53) /*!< For
SD I/O Card only */

    /**
      * @brief Following commands are SD Card Specific security commands.
      *        SDIO_APP_CMD should be sent before sending these commands.
      */
    #define SD_CMD_SD_APP_GET_MKB                       ((uint8_t)43) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_GET_MID                       ((uint8_t)44) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SET_CER_RN1                   ((uint8_t)45) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_GET_CER_RN2                   ((uint8_t)46) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SET_CER_RES2                  ((uint8_t)47) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_GET_CER_RES1                  ((uint8_t)48) /*!< For
SD Card only */
```

```
    #define SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK   ((uint8_t)18) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK  ((uint8_t)25) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SECURE_ERASE                 ((uint8_t)38) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_CHANGE_SECURE_AREA           ((uint8_t)49) /*!< For
SD Card only */
    #define SD_CMD_SD_APP_SECURE_WRITE_MKB             ((uint8_t)48) /*!< For
SD Card only */

    /* Uncomment the following line to select the SDIO Data transfer mode */
    #define SD_DMA_MODE
((uint32_t)0x00000000)
    /*#define SD_POLLING_MODE
((uint32_t)0x00000002)*/

    /**
      * @brief  SD detection on its memory slot
      */
    #define SD_PRESENT                                 ((uint8_t)0x01)
    #define SD_NOT_PRESENT                             ((uint8_t)0x00)

    /**
      * @brief Supported SD Memory Cards
      */
    #define SDIO_STD_CAPACITY_SD_CARD_V1_1
((uint32_t)0x00000000)
    #define SDIO_STD_CAPACITY_SD_CARD_V2_0
((uint32_t)0x00000001)
    #define SDIO_HIGH_CAPACITY_SD_CARD
((uint32_t)0x00000002)
    #define SDIO_MULTIMEDIA_CARD
((uint32_t)0x00000003)
    #define SDIO_SECURE_DIGITAL_IO_CARD
((uint32_t)0x00000004)
    #define SDIO_HIGH_SPEED_MULTIMEDIA_CARD
((uint32_t)0x00000005)
    #define SDIO_SECURE_DIGITAL_IO_COMBO_CARD
((uint32_t)0x00000006)
    #define SDIO_HIGH_CAPACITY_MMC_CARD
((uint32_t)0x00000007)
```

```
    /* Exported functions
------------------------------------------------------- */
    void SD_DeInit(void);
    SD_Error SD_Init(void);
    SDTransferState SD_GetStatus(void);
    SDCardState SD_GetState(void);
    uint8_t SD_Detect(void);
    SD_Error SD_PowerON(void);
    SD_Error SD_PowerOFF(void);
    SD_Error SD_InitializeCards(void);
    SD_Error SD_GetCardInfo(SD_CardInfo *cardinfo);
    SD_Error SD_GetCardStatus(SD_CardStatus *cardstatus);
    SD_Error SD_EnableWideBusOperation(uint32_t WideMode);
    SD_Error SD_SelectDeselect(uint32_t addr);
    SD_Error SD_ReadBlock(uint8_t *readbuff, uint32_t ReadAddr, uint16_t
BlockSize);
    SD_Error SD_ReadMultiBlocks(uint8_t *readbuff, uint32_t ReadAddr, uint16_t
BlockSize, uint32_t NumberOfBlocks);
    SD_Error SD_WriteBlock(uint8_t *writebuff, uint32_t WriteAddr, uint16_t
BlockSize);
    SD_Error SD_WriteMultiBlocks(uint8_t *writebuff, uint32_t WriteAddr,
uint16_t BlockSize, uint32_t NumberOfBlocks);
    SDTransferState SD_GetTransferState(void);
    SD_Error SD_StopTransfer(void);
    SD_Error SD_Erase(uint32_t startaddr, uint32_t endaddr);
    SD_Error SD_SendStatus(uint32_t *pcardstatus);
    SD_Error SD_SendSDStatus(uint32_t *psdstatus);
    SD_Error SD_ProcessIRQSrc(void);
    SD_Error SD_WaitReadOperation(void);
    SD_Error SD_WaitWriteOperation(void);

    #endif /* __SDCARD_H */
```

4.35.19 sdio_sdcard.c 文件里的内容

核心程序是 sdio_sdcard.c 文件里是 SD 卡工作的重点核心，整个工作都在其中，整个程序非常庞大，编辑的非常严谨，在整个初始化过程中采用了大量的结构体，通过各种判断完成 SD 卡的初始化过程。

```
    #include "pbdata.h"
```

```
#define NULL 0
#define SDIO_STATIC_FLAGS               ((uint32_t)0x000005FF)
#define SDIO_CMD0TIMEOUT                ((uint32_t)0x00010000)
/**
  * @brief  Mask for errors Card Status R1 (OCR Register)
  */
#define SD_OCR_ADDR_OUT_OF_RANGE        ((uint32_t)0x80000000)
#define SD_OCR_ADDR_MISALIGNED          ((uint32_t)0x40000000)
#define SD_OCR_BLOCK_LEN_ERR            ((uint32_t)0x20000000)
#define SD_OCR_ERASE_SEQ_ERR            ((uint32_t)0x10000000)
#define SD_OCR_BAD_ERASE_PARAM          ((uint32_t)0x08000000)
#define SD_OCR_WRITE_PROT_VIOLATION     ((uint32_t)0x04000000)
#define SD_OCR_LOCK_UNLOCK_FAILED       ((uint32_t)0x01000000)
#define SD_OCR_COM_CRC_FAILED           ((uint32_t)0x00800000)
#define SD_OCR_ILLEGAL_CMD              ((uint32_t)0x00400000)
#define SD_OCR_CARD_ECC_FAILED          ((uint32_t)0x00200000)
#define SD_OCR_CC_ERROR                 ((uint32_t)0x00100000)
#define SD_OCR_GENERAL_UNKNOWN_ERROR    ((uint32_t)0x00080000)
#define SD_OCR_STREAM_READ_UNDERRUN     ((uint32_t)0x00040000)
#define SD_OCR_STREAM_WRITE_OVERRUN     ((uint32_t)0x00020000)
#define SD_OCR_CID_CSD_OVERWRIETE       ((uint32_t)0x00010000)
#define SD_OCR_WP_ERASE_SKIP            ((uint32_t)0x00008000)
#define SD_OCR_CARD_ECC_DISABLED        ((uint32_t)0x00004000)
#define SD_OCR_ERASE_RESET              ((uint32_t)0x00002000)
#define SD_OCR_AKE_SEQ_ERROR            ((uint32_t)0x00000008)
#define SD_OCR_ERRORBITS                ((uint32_t)0xFDFFE008)

/**
  * @brief  Masks for R6 Response
  */
#define SD_R6_GENERAL_UNKNOWN_ERROR     ((uint32_t)0x00002000)
#define SD_R6_ILLEGAL_CMD               ((uint32_t)0x00004000)
#define SD_R6_COM_CRC_FAILED            ((uint32_t)0x00008000)

#define SD_VOLTAGE_WINDOW_SD            ((uint32_t)0x80100000)
#define SD_HIGH_CAPACITY                ((uint32_t)0x40000000)
#define SD_STD_CAPACITY                 ((uint32_t)0x00000000)
#define SD_CHECK_PATTERN                ((uint32_t)0x000001AA)

#define SD_MAX_VOLT_TRIAL               ((uint32_t)0x0000FFFF)
#define SD_ALLZERO                      ((uint32_t)0x00000000)

#define SD_WIDE_BUS_SUPPORT             ((uint32_t)0x00040000)
```

```
    #define SD_SINGLE_BUS_SUPPORT           ((uint32_t)0x00010000)
    #define SD_CARD_LOCKED                  ((uint32_t)0x02000000)

    #define SD_DATATIMEOUT                  ((uint32_t)0xFFFFFFFF)
    #define SD_0TO7BITS                     ((uint32_t)0x000000FF)
    #define SD_8TO15BITS                    ((uint32_t)0x0000FF00)
    #define SD_16TO23BITS                   ((uint32_t)0x00FF0000)
    #define SD_24TO31BITS                   ((uint32_t)0xFF000000)
    #define SD_MAX_DATA_LENGTH              ((uint32_t)0x01FFFFFF)

    #define SD_HALFFIFO                     ((uint32_t)0x00000008)
    #define SD_HALFFIFOBYTES                ((uint32_t)0x00000020)

    /**
      * @brief  Command Class Supported
      */
    #define SD_CCCC_LOCK_UNLOCK             ((uint32_t)0x00000080)
    #define SD_CCCC_WRITE_PROT              ((uint32_t)0x00000040)
    #define SD_CCCC_ERASE                   ((uint32_t)0x00000020)

    /**
      * @brief  Following commands are SD Card Specific commands.
      *         SDIO_APP_CMD should be sent before sending these commands.
      */
    #define SDIO_SEND_IF_COND               ((uint32_t)0x00000008)



    /* Private variables
------------------------------------------------------------*/
    static uint32_t CardType =  SDIO_STD_CAPACITY_SD_CARD_V1_1;    //存储卡
的类型，先把它初始化为 1.1 协议的卡
    static uint32_t CSD_Tab[4], CID_Tab[4], RCA = 0;//存储 CSD，DID，寄存器和
卡相对地址
    static uint8_t SDSTATUS_Tab[16]; //存储卡状态，是 CSR 的一部分
    __IO uint32_t StopCondition = 0; //用于停止卡操作的标志
    __IO SD_Error TransferError = SD_OK; //用于存储卡错误，初始化为正常状态
    __IO uint32_t TransferEnd = 0; //用于标志传输是否结束，在中断服务函数中调
用
    SD_CardInfo SDCardInfo;   //用于存储卡的信息，DSR 的一部分？

    /*用于 sdio 初始化的结构体*/
    SDIO_InitTypeDef SDIO_InitStructure;
    SDIO_CmdInitTypeDef SDIO_CmdInitStructure;
```

```
    SDIO_DataInitTypeDef SDIO_DataInitStructure;

    /* Private function prototypes
-----------------------------------------------*/
    static SD_Error CmdError(void);
    static SD_Error CmdResp1Error(uint8_t cmd);
    static SD_Error CmdResp7Error(void);
    static SD_Error CmdResp3Error(void);
    static SD_Error CmdResp2Error(void);
    static SD_Error CmdResp6Error(uint8_t cmd, uint16_t *prca);
    static SD_Error SDEnWideBus(FunctionalState NewState);
    static SD_Error IsCardProgramming(uint8_t *pstatus);
    static SD_Error FindSCR(uint16_t rca, uint32_t *pscr);

    static void GPIO_Configuration(void);
    static uint32_t SD_DMAEndOfTransferStatus(void);
    static void SD_DMA_RxConfig(uint32_t *BufferDST, uint32_t BufferSize);
    static void SD_DMA_TxConfig(uint32_t *BufferSRC, uint32_t BufferSize);

    uint8_t convert_from_bytes_to_power_of_two(uint16_t NumberOfBytes);

    /* Private functions
---------------------------------------------------------*/
    /*
     * 函数名：SD_DeInit
     * 描述  ：复位 SDIO 端口
     * 输入  ：无
     * 输出  ：无
     */
    void SD_DeInit(void)
    {
      GPIO_InitTypeDef  GPIO_InitStructure;

      /*!< Disable SDIO Clock */
      SDIO_ClockCmd(DISABLE);

      /*!< Set Power State to OFF */
      SDIO_SetPowerState(SDIO_PowerState_OFF);

      /*!< DeInitializes the SDIO peripheral */
      SDIO_DeInit();

      /*!< Disable the SDIO AHB Clock */
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_SDIO, DISABLE);
```

```
      /*!< Configure PC.08, PC.09, PC.10, PC.11, PC.12 pin: D0, D1, D2, D3, CLK
pin */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
GPIO_Pin_11 | GPIO_Pin_12;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
      GPIO_Init(GPIOC, &GPIO_InitStructure);

      /*!< Configure PD.02 CMD line */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
      GPIO_Init(GPIOD, &GPIO_InitStructure);
    }



    /**
      * @brief  Returns the DMA End Of Transfer Status.
      * @param  None
      * @retval DMA SDIO Channel Status.
      */
    uint32_t SD_DMAEndOfTransferStatus(void)
    {
      return (uint32_t)DMA_GetFlagStatus(DMA2_FLAG_TC4);    //Channel4
transfer complete flag.

    }



     /*
     * 函数名：SD_DMA_RxConfig
     * 描述  ：为 SDIO 接收数据配置 DMA2 的通道 4 的请求
     * 输入  ：BufferDST：用于装载数据的变量指针
               BufferSize：    缓冲区大小
     * 输出  ：无
     */
    void SD_DMA_RxConfig(uint32_t *BufferDST, uint32_t BufferSize)
    {
      DMA_InitTypeDef DMA_InitStructure;

      DMA_ClearFlag(DMA2_FLAG_TC4 | DMA2_FLAG_TE4 | DMA2_FLAG_HT4 |
DMA2_FLAG_GL4);//清除 DMA 标志位

      /*!< DMA2 Channel4 disable */
      DMA_Cmd(DMA2_Channel4, DISABLE); //SDIO 为第四通道
```

```
    /*!< DMA2 Channel4 Config */
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SDIO_FIFO_ADDRESS;
//外设地址，fifo
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)BufferDST; //目标地址
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;    //外设为原地址
    DMA_InitStructure.DMA_BufferSize = BufferSize / 4;  //1/4 缓存大小
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;//使能
外设地址不自增
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;    //使能存储
目标地址自增
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
//外设数据大小为字，32 位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word; //外设
数据大小为字，32 位
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;           //不循环，循环
模式主要用在 adc 上
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;    //通道优先级高
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;           //非 存储器至存
储器模式
    DMA_Init(DMA2_Channel4, &DMA_InitStructure);

    /*!< DMA2 Channel4 enable */           //不设置 dma 中断?
    DMA_Cmd(DMA2_Channel4, ENABLE);
  }

   /*
   * 函数名：SD_DMA_RxConfig
   * 描述  ：为 SDIO 发送数据配置 DMA2 的通道 4 的请求
   * 输入  ：BufferDST：装载了数据的变量指针
             BufferSize：   缓冲区大小
   * 输出  ：无
   */
  void SD_DMA_TxConfig(uint32_t *BufferSRC, uint32_t BufferSize)
  {

    DMA_InitTypeDef DMA_InitStructure;

    DMA_ClearFlag(DMA2_FLAG_TC4 | DMA2_FLAG_TE4 | DMA2_FLAG_HT4 |
DMA2_FLAG_GL4);

    /*!< DMA2 Channel4 disable */
    DMA_Cmd(DMA2_Channel4, DISABLE);
```

```
    /*!< DMA2 Channel4 Config */
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)SDIO_FIFO_ADDRESS;
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)BufferSRC;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;//外设为写入目标
    DMA_InitStructure.DMA_BufferSize = BufferSize / 4;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;    //
外设地址不自增
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA2_Channel4, &DMA_InitStructure);

    /*!< DMA2 Channel4 enable */
    DMA_Cmd(DMA2_Channel4, ENABLE);
  }



  /*
   * 函数名：GPIO_Configuration
   * 描述  ：初始化 SDIO 用到的引脚，开启时钟。
   * 输入  ：无
   * 输出  ：无
   * 调用  ：内部调用
   */
  static void GPIO_Configuration(void)
  {
    GPIO_InitTypeDef  GPIO_InitStructure;

    /*!< GPIOC and GPIOD Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD ,
ENABLE);

    /*!< Configure PC.08, PC.09, PC.10, PC.11, PC.12 pin: D0, D1, D2, D3, CLK
pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
GPIO_Pin_11 | GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /*!< Configure PD.02 CMD line */
```

```
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  /*!< Enable the SDIO AHB Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_SDIO, ENABLE);

  /*!< Enable the DMA2 Clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA2, ENABLE);
}



/*
 * 函数名：SD_Init
 * 描述  ：初始化 SD 卡，使卡处于就绪状态(准备传输数据)
 * 输入  ：无
 * 输出  ：-SD_Error SD 卡错误代码
 *         成功时则为 SD_OK
 * 调用  ：外部调用
 */
SD_Error SD_Init(void)
{
  /*重置 SD_Error 状态*/
  SD_Error errorstatus = SD_OK;

  /* SDIO 外设底层引脚初始化 */
  GPIO_Configuration();

  /*对 SDIO 的所有寄存器进行复位*/
  SDIO_DeInit();

  /*上电并进行卡识别流程，确认卡的操作电压  */
  errorstatus = SD_PowerON();

  /*如果上电，识别不成功，返回“响应超时”错误 */
  if (errorstatus != SD_OK)
  {
    /*!< CMD Response TimeOut (wait for CMDSENT flag) */
    return(errorstatus);
  }

  /*卡识别成功，进行卡初始化    */
  errorstatus = SD_InitializeCards();

  if (errorstatus != SD_OK)   //失败返回
```

```
      {
        /*!< CMD Response TimeOut (wait for CMDSENT flag) */
        return(errorstatus);
      }

      /*!< Configure the SDIO peripheral
      上电识别，卡初始化都完成后，进入数据传输模式，提高读写速度
      速度若超过 24M 要进入 bypass 模式
      !< on STM32F2xx devices, SDIOCLK is fixed to 48MHz
      !< SDIOCLK = HCLK, SDIO_CK = HCLK/(2 + SDIO_TRANSFER_CLK_DIV) */
      SDIO_InitStructure.SDIO_ClockDiv = SDIO_TRANSFER_CLK_DIV;
      SDIO_InitStructure.SDIO_ClockEdge = SDIO_ClockEdge_Rising;        //上
升沿采集数据
      SDIO_InitStructure.SDIO_ClockBypass = SDIO_ClockBypass_Disable;  //时钟
频率若超过 24M,要开启此模式
      SDIO_InitStructure.SDIO_ClockPowerSave = SDIO_ClockPowerSave_Disable;
       //若开启此功能，在总线空闲时关闭 sd_clk 时钟
      SDIO_InitStructure.SDIO_BusWide = SDIO_BusWide_1b;
    //1 位模式
      SDIO_InitStructure.SDIO_HardwareFlowControl =
SDIO_HardwareFlowControl_Disable;//硬件流，若开启，在 FIFO 不能进行发送和接收
数据时，数据传输暂停
      SDIO_Init(&SDIO_InitStructure);

      if (errorstatus == SD_OK)
      {
        /*---------------- Read CSD/CID MSD registers ------------------*/
        errorstatus = SD_GetCardInfo(&SDCardInfo);    //用来读取 csd/cid 寄存
器
      }

      if (errorstatus == SD_OK)
      {
        /*---------------- Select Card --------------------------------*/
        errorstatus = SD_SelectDeselect((uint32_t) (SDCardInfo.RCA << 16));
      //通过 cmd7  ,rca 选择要操作的卡
      }

      if (errorstatus == SD_OK)
      {
        errorstatus = SD_EnableWideBusOperation(SDIO_BusWide_4b); //开启
4bits 模式
      }
```

```
  return(errorstatus);
}

/**
  * @brief  Gets the cuurent sd card data transfer status.
  * @param  None
  * @retval SDTransferState: Data Transfer state.
  *   This value can be:
  *        - SD_TRANSFER_OK: No data transfer is acting
  *        - SD_TRANSFER_BUSY: Data transfer is acting
  */
SDTransferState SD_GetStatus(void)
{
  SDCardState cardstate =  SD_CARD_TRANSFER;

  cardstate = SD_GetState();

  if (cardstate == SD_CARD_TRANSFER)
  {
    return(SD_TRANSFER_OK);
  }
  else if(cardstate == SD_CARD_ERROR)
  {
    return (SD_TRANSFER_ERROR);
  }
  else
  {
    return(SD_TRANSFER_BUSY);
  }
}

/**
  * @brief  Returns the current card's state.
  * @param  None
  * @retval SDCardState: SD Card Error or SD Card Current State.
  */
SDCardState SD_GetState(void)
{
  uint32_t resp1 = 0;

    if (SD_SendStatus(&resp1) != SD_OK)
    {
      return SD_CARD_ERROR;
    }
```

```
        else
        {
          return (SDCardState)((resp1 >> 9) & 0x0F);
        }
    }



    /*
     * 函数名：SD_PowerON
     * 描述  ：确保 SD 卡的工作电压和配置控制时钟
     * 输入  ：无
     * 输出  ：-SD_Error SD 卡错误代码
     *         成功时则为 SD_OK
     * 调用  ：在 SD_Init() 调用
     */
    SD_Error SD_PowerON(void)
    {
      SD_Error errorstatus = SD_OK;
      uint32_t response = 0, count = 0, validvoltage = 0;
      uint32_t SDType = SD_STD_CAPACITY;

      /*!< Power ON Sequence
-----------------------------------------------------*/
      /*!< Configure the SDIO peripheral */
      /*!< SDIOCLK = HCLK, SDIO_CK = HCLK/(2 + SDIO_INIT_CLK_DIV) */
      /*!< on STM32F2xx devices, SDIOCLK is fixed to 48MHz */
      /*!< SDIO_CK for initialization should not exceed 400 KHz */
      /*初始化时的时钟不能大于 400KHz*/
      SDIO_InitStructure.SDIO_ClockDiv = SDIO_INIT_CLK_DIV;//SDIO 初始化时钟
分频系数
      SDIO_InitStructure.SDIO_ClockEdge = SDIO_ClockEdge_Rising;//在主时钟上
升沿产生 SDIO 时钟
      SDIO_InitStructure.SDIO_ClockBypass = SDIO_ClockBypass_Disable;  //时钟
旁路不使能 不使用 bypass 模式，直接用 HCLK 进行分频得到 SDIO_CK
      SDIO_InitStructure.SDIO_ClockPowerSave = SDIO_ClockPowerSave_Disable;
    //省电模式，总线空闲时 SDIO 时钟输出关闭
      SDIO_InitStructure.SDIO_BusWide = SDIO_BusWide_1b;                  //
传输 1 位数据
      SDIO_InitStructure.SDIO_HardwareFlowControl =
SDIO_HardwareFlowControl_Disable;//不使能数据流控制
      SDIO_Init(&SDIO_InitStructure);

      //给 SD 卡上电
```

```
      SDIO_SetPowerState(SDIO_PowerState_ON);

      /*!< Enable SDIO Clock */
      SDIO_ClockCmd(ENABLE);

      /*下面发送一系列命令,开始卡识别流程*/
      SDIO_CmdInitStructure.SDIO_Argument = 0x0;//命令参数为 0
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_GO_IDLE_STATE; //cmd0
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_No;  //无响应
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;//无需等待回应 等到超时
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;  //则 CPSM 在开始发
送命令之前等待数据传输结束。
      SDIO_SendCommand(&SDIO_CmdInitStructure);          //写命令进命令寄存器

      errorstatus = CmdError();//检测是否正确接收到 cmd0

      if (errorstatus != SD_OK) //命令发送出错，返回
      {
        /*!< CMD Response TimeOut (wait for CMDSENT flag) */
        return(errorstatus);
      }

      /*!< CMD8: SEND_IF_COND
-------------------------------------------------*/
      /*!< Send CMD8 to verify SD card interface operating condition */
      /*!< Argument: - [31:12]: Reserved (shall be set to '0')
                  - [11:8]: Supply Voltage (VHS) 0x1 (Range: 2.7-3.6 V)
                  - [7:0]: Check Pattern (recommended 0xAA) */
      /*!< CMD Response: R7 */
      SDIO_CmdInitStructure.SDIO_Argument = SD_CHECK_PATTERN;//检测卡类型命令
参数
      SDIO_CmdInitStructure.SDIO_CmdIndex = SDIO_SEND_IF_COND;//识别卡命令
cmd8
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;//短响应 r7
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;          //无需等待
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;      //使能命令通
道状态机
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      /*检查是否接收到命令*/
      errorstatus = CmdResp7Error();

      if (errorstatus == SD_OK) //有响应则 SD 卡遵循 SD 协议 2.0 版本
      {
```

```
        CardType = SDIO_STD_CAPACITY_SD_CARD_V2_0; //先把它定义成 sdsc 类型的
卡
        SDType = SD_HIGH_CAPACITY; //这个变量用作 acmd41 的参数，用来询问是
sdsc 卡还是 sdhc 卡
      }
      else //无响应，说明是 1.x 的或 mmc 的卡
      {
        /*!< CMD55 */
        SDIO_CmdInitStructure.SDIO_Argument = 0x00;
        SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
        SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
        SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
        SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
        SDIO_SendCommand(&SDIO_CmdInitStructure);
        errorstatus = CmdResp1Error(SD_CMD_APP_CMD);
      }
      /*!< CMD55 */  //为什么在 else 里和 else 外面都要发送 CMD55?
      //发送 cmd55，用于检测是 sd 卡还是 mmc 卡，或是不支持的卡
      SDIO_CmdInitStructure.SDIO_Argument = 0x00;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short; //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);
      errorstatus = CmdResp1Error(SD_CMD_APP_CMD);  //是否响应，没响应的是
mmc 或不支持的卡

      /*!< If errorstatus is Command TimeOut, it is a MMC card */
      /*!< If errorstatus is SD_OK it is a SD card: SD card 2.0 (voltage range
mismatch)
         or SD card 1.x */
      if (errorstatus == SD_OK) //响应了 cmd55，是 sd 卡，可能为 1.x,可能为 2.0
      {
       /*下面开始循环地发送 sdio 支持的电压范围，循环一定次数*/

        /*!< SD CARD */
        /*!< Send ACMD41 SD_APP_OP_COND with Argument 0x80100000 */
        while ((!validvoltage) && (count < SD_MAX_VOLT_TRIAL))
        {
       //因为下面要用到 ACMD41，是 ACMD 命令，在发送 ACMD 命令前都要先向卡发送
CMD55
          /*!< SEND CMD55 APP_CMD with RCA as 0 */
          SDIO_CmdInitStructure.SDIO_Argument = 0x00;
          SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;      //CMD55
```

```
          SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
          SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
          SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
          SDIO_SendCommand(&SDIO_CmdInitStructure);

          errorstatus = CmdResp1Error(SD_CMD_APP_CMD); //检测响应

          if (errorstatus != SD_OK)
          {
            return(errorstatus);//没响应 CMD55，返回
          }
         //acmd41，命令参数由支持的电压范围及 HCS 位组成，HCS 位置一来区分卡是
SDSc 还是 sdhc
          SDIO_CmdInitStructure.SDIO_Argument = SD_VOLTAGE_WINDOW_SD | SDType;
     //参数为主机可供电压范围及 hcs 位
          SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SD_APP_OP_COND;
          SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;  //r3
          SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
          SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
          SDIO_SendCommand(&SDIO_CmdInitStructure);

          errorstatus = CmdResp3Error();  //检测是否正确接收到数据
          if (errorstatus != SD_OK)
          {
            return(errorstatus);  //没正确接收到 acmd41，出错，返回
          }
          /*若卡需求电压在 SDIO 的供电电压范围内，会自动上电并标志 pwr_up 位*/
          response = SDIO_GetResponse(SDIO_RESP1);    //读取卡寄存器，卡状态
          validvoltage = (((response >> 31) == 1) ? 1 : 0); //读取卡的 ocr 寄
存器的 pwr_up 位，看是否已工作在正常电压
          count++;              //计算循环次数
        }
        if (count >= SD_MAX_VOLT_TRIAL)    //循环检测超过一定次数还没上电
        {
          errorstatus = SD_INVALID_VOLTRANGE;    //SDIO 不支持 card 的供电电压
          return(errorstatus);
        }
              /*检查卡返回信息中的 HCS 位*/
        if (response &= SD_HIGH_CAPACITY)  //判断 ocr 中的 ccs 位 ，如果是 sdsc
卡则不执行下面的语句
        {
          CardType = SDIO_HIGH_CAPACITY_SD_CARD;  //把卡类型从初始化的 sdsc 型
改为 sdhc 型
        }
```

```
    }/*!< else MMC Card */

    return(errorstatus);
  }

  /*
   * 函数名：SD_PowerOFF
   * 描述  ：关掉 SDIO 的输出信号
   * 输入  ：无
   * 输出  ：-SD_Error SD 卡错误代码
   *         成功时则为 SD_OK
   * 调用  ：外部调用
   */
  SD_Error SD_PowerOFF(void)
  {
    SD_Error errorstatus = SD_OK;

    /*!< Set Power State to OFF */
    SDIO_SetPowerState(SDIO_PowerState_OFF);

    return(errorstatus);
  }


    /*
   * 函数名：SD_InitializeCards
   * 描述  ：初始化所有的卡或者单个卡进入就绪状态
   * 输入  ：无
   * 输出  ：-SD_Error SD 卡错误代码
   *         成功时则为 SD_OK
   * 调用  ：在 SD_Init() 调用，在调用 power_on（）上电卡识别完毕后，调用此
函数进行卡初始化
   */
  SD_Error SD_InitializeCards(void)
  {
    SD_Error errorstatus = SD_OK;
    uint16_t rca = 0x01;

    if (SDIO_GetPowerState() == SDIO_PowerState_OFF)
    {
      errorstatus = SD_REQUEST_NOT_APPLICABLE;
      return(errorstatus);
    }
```

```
    if (SDIO_SECURE_DIGITAL_IO_CARD != CardType)//判断卡的类型
    {
      /*!< Send CMD2 ALL_SEND_CID */
      SDIO_CmdInitStructure.SDIO_Argument = 0x0;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_ALL_SEND_CID; //CMD2
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Long;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp2Error();

      if (SD_OK != errorstatus)
      {
        return(errorstatus);
      }

      CID_Tab[0] = SDIO_GetResponse(SDIO_RESP1);
      CID_Tab[1] = SDIO_GetResponse(SDIO_RESP2);
      CID_Tab[2] = SDIO_GetResponse(SDIO_RESP3);
      CID_Tab[3] = SDIO_GetResponse(SDIO_RESP4);
    }

    /*下面开始 SD 卡初始化流程*/
    if ((SDIO_STD_CAPACITY_SD_CARD_V1_1 == CardType) ||
(SDIO_STD_CAPACITY_SD_CARD_V2_0 == CardType) ||
(SDIO_SECURE_DIGITAL_IO_COMBO_CARD == CardType)
        ||  (SDIO_HIGH_CAPACITY_SD_CARD == CardType))  //使用的是 2.0 的卡
    {
      /*!< Send CMD3 SET_REL_ADDR with argument 0 */
      /*!< SD Card publishes its RCA. */
      SDIO_CmdInitStructure.SDIO_Argument = 0x00;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_REL_ADDR;//cmd3
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short; //r6
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp6Error(SD_CMD_SET_REL_ADDR, &rca);    //把接收
到的卡相对地址存起来。

      if (SD_OK != errorstatus)
      {
```

```
      return(errorstatus);
    }
  }

  if (SDIO_SECURE_DIGITAL_IO_CARD != CardType)
  {
    RCA = rca;

    /*!< Send CMD9 SEND_CSD with argument as card's RCA */
    SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)(rca << 16);
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SEND_CSD;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Long;
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp2Error();

    if (SD_OK != errorstatus)
    {
      return(errorstatus);
    }

    CSD_Tab[0] = SDIO_GetResponse(SDIO_RESP1);
    CSD_Tab[1] = SDIO_GetResponse(SDIO_RESP2);
    CSD_Tab[2] = SDIO_GetResponse(SDIO_RESP3);
    CSD_Tab[3] = SDIO_GetResponse(SDIO_RESP4);
  }

  errorstatus = SD_OK; /*!< All cards get intialized */

  return(errorstatus);
}

/*
 * 函数名：SD_GetCardInfo
 * 描述  ：获取 SD 卡的具体信息
 * 输入  ：-cardinfo 指向 SD_CardInfo 结构体的指针
 *         这个结构里面包含了 SD 卡的具体信息
 * 输出  ：-SD_Error SD 卡错误代码
 *         成功时则为 SD_OK
 * 调用  ：外部调用
 */
SD_Error SD_GetCardInfo(SD_CardInfo *cardinfo)
```

```
    {
      SD_Error errorstatus = SD_OK;
      uint8_t tmp = 0;

      cardinfo->CardType = (uint8_t)CardType;
      cardinfo->RCA = (uint16_t)RCA;

      /*!< Byte 0 */
      tmp = (uint8_t)((CSD_Tab[0] & 0xFF000000) >> 24);
      cardinfo->SD_csd.CSDStruct = (tmp & 0xC0) >> 6;
      cardinfo->SD_csd.SysSpecVersion = (tmp & 0x3C) >> 2;
      cardinfo->SD_csd.Reserved1 = tmp & 0x03;

      /*!< Byte 1 */
      tmp = (uint8_t)((CSD_Tab[0] & 0x00FF0000) >> 16);
      cardinfo->SD_csd.TAAC = tmp;

      /*!< Byte 2 */
      tmp = (uint8_t)((CSD_Tab[0] & 0x0000FF00) >> 8);
      cardinfo->SD_csd.NSAC = tmp;

      /*!< Byte 3 */
      tmp = (uint8_t)(CSD_Tab[0] & 0x000000FF);
      cardinfo->SD_csd.MaxBusClkFrec = tmp;

      /*!< Byte 4 */
      tmp = (uint8_t)((CSD_Tab[1] & 0xFF000000) >> 24);
      cardinfo->SD_csd.CardComdClasses = tmp << 4;

      /*!< Byte 5 */
      tmp = (uint8_t)((CSD_Tab[1] & 0x00FF0000) >> 16);
      cardinfo->SD_csd.CardComdClasses |= (tmp & 0xF0) >> 4;
      cardinfo->SD_csd.RdBlockLen = tmp & 0x0F;

      /*!< Byte 6 */
      tmp = (uint8_t)((CSD_Tab[1] & 0x0000FF00) >> 8);
      cardinfo->SD_csd.PartBlockRead = (tmp & 0x80) >> 7;
      cardinfo->SD_csd.WrBlockMisalign = (tmp & 0x40) >> 6;
      cardinfo->SD_csd.RdBlockMisalign = (tmp & 0x20) >> 5;
      cardinfo->SD_csd.DSRImpl = (tmp & 0x10) >> 4;
      cardinfo->SD_csd.Reserved2 = 0; /*!< Reserved */

      if ((CardType == SDIO_STD_CAPACITY_SD_CARD_V1_1) || (CardType ==
SDIO_STD_CAPACITY_SD_CARD_V2_0))
```

```
  {
    cardinfo->SD_csd.DeviceSize = (tmp & 0x03) << 10;

    /*!< Byte 7 */
    tmp = (uint8_t)(CSD_Tab[1] & 0x000000FF);
    cardinfo->SD_csd.DeviceSize |= (tmp) << 2;

    /*!< Byte 8 */
    tmp = (uint8_t)((CSD_Tab[2] & 0xFF000000) >> 24);
    cardinfo->SD_csd.DeviceSize |= (tmp & 0xC0) >> 6;

    cardinfo->SD_csd.MaxRdCurrentVDDMin = (tmp & 0x38) >> 3;
    cardinfo->SD_csd.MaxRdCurrentVDDMax = (tmp & 0x07);

    /*!< Byte 9 */
    tmp = (uint8_t)((CSD_Tab[2] & 0x00FF0000) >> 16);
    cardinfo->SD_csd.MaxWrCurrentVDDMin = (tmp & 0xE0) >> 5;
    cardinfo->SD_csd.MaxWrCurrentVDDMax = (tmp & 0x1C) >> 2;
    cardinfo->SD_csd.DeviceSizeMul = (tmp & 0x03) << 1;
    /*!< Byte 10 */
    tmp = (uint8_t)((CSD_Tab[2] & 0x0000FF00) >> 8);
    cardinfo->SD_csd.DeviceSizeMul |= (tmp & 0x80) >> 7;

    cardinfo->CardCapacity = (cardinfo->SD_csd.DeviceSize + 1) ;
    cardinfo->CardCapacity *= (1 << (cardinfo->SD_csd.DeviceSizeMul + 2));
    cardinfo->CardBlockSize = 1 << (cardinfo->SD_csd.RdBlockLen);
    cardinfo->CardCapacity *= cardinfo->CardBlockSize;
  }
  else if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)
  {
    /*!< Byte 7 */
    tmp = (uint8_t)(CSD_Tab[1] & 0x000000FF);
    cardinfo->SD_csd.DeviceSize = (tmp & 0x3F) << 16;

    /*!< Byte 8 */
    tmp = (uint8_t)((CSD_Tab[2] & 0xFF000000) >> 24);

    cardinfo->SD_csd.DeviceSize |= (tmp << 8);

    /*!< Byte 9 */
    tmp = (uint8_t)((CSD_Tab[2] & 0x00FF0000) >> 16);

    cardinfo->SD_csd.DeviceSize |= (tmp);
```

```
    /*!< Byte 10 */
    tmp = (uint8_t)((CSD_Tab[2] & 0x0000FF00) >> 8);

    cardinfo->CardCapacity = (cardinfo->SD_csd.DeviceSize + 1) * 512 * 1024;
    cardinfo->CardBlockSize = 512;
  }


  cardinfo->SD_csd.EraseGrSize = (tmp & 0x40) >> 6;
  cardinfo->SD_csd.EraseGrMul = (tmp & 0x3F) << 1;

  /*!< Byte 11 */
  tmp = (uint8_t)(CSD_Tab[2] & 0x000000FF);
  cardinfo->SD_csd.EraseGrMul |= (tmp & 0x80) >> 7;
  cardinfo->SD_csd.WrProtectGrSize = (tmp & 0x7F);

  /*!< Byte 12 */
  tmp = (uint8_t)((CSD_Tab[3] & 0xFF000000) >> 24);
  cardinfo->SD_csd.WrProtectGrEnable = (tmp & 0x80) >> 7;
  cardinfo->SD_csd.ManDeflECC = (tmp & 0x60) >> 5;
  cardinfo->SD_csd.WrSpeedFact = (tmp & 0x1C) >> 2;
  cardinfo->SD_csd.MaxWrBlockLen = (tmp & 0x03) << 2;

  /*!< Byte 13 */
  tmp = (uint8_t)((CSD_Tab[3] & 0x00FF0000) >> 16);
  cardinfo->SD_csd.MaxWrBlockLen |= (tmp & 0xC0) >> 6;
  cardinfo->SD_csd.WriteBlockPaPartial = (tmp & 0x20) >> 5;
  cardinfo->SD_csd.Reserved3 = 0;
  cardinfo->SD_csd.ContentProtectAppli = (tmp & 0x01);

  /*!< Byte 14 */
  tmp = (uint8_t)((CSD_Tab[3] & 0x0000FF00) >> 8);
  cardinfo->SD_csd.FileFormatGrouop = (tmp & 0x80) >> 7;
  cardinfo->SD_csd.CopyFlag = (tmp & 0x40) >> 6;
  cardinfo->SD_csd.PermWrProtect = (tmp & 0x20) >> 5;
  cardinfo->SD_csd.TempWrProtect = (tmp & 0x10) >> 4;
  cardinfo->SD_csd.FileFormat = (tmp & 0x0C) >> 2;
  cardinfo->SD_csd.ECC = (tmp & 0x03);

  /*!< Byte 15 */
  tmp = (uint8_t)(CSD_Tab[3] & 0x000000FF);
  cardinfo->SD_csd.CSD_CRC = (tmp & 0xFE) >> 1;
  cardinfo->SD_csd.Reserved4 = 1;
```

```
/*!< Byte 0 */
tmp = (uint8_t)((CID_Tab[0] & 0xFF000000) >> 24);
cardinfo->SD_cid.ManufacturerID = tmp;

/*!< Byte 1 */
tmp = (uint8_t)((CID_Tab[0] & 0x00FF0000) >> 16);
cardinfo->SD_cid.OEM_AppliID = tmp << 8;

/*!< Byte 2 */
tmp = (uint8_t)((CID_Tab[0] & 0x000000FF00) >> 8);
cardinfo->SD_cid.OEM_AppliID |= tmp;

/*!< Byte 3 */
tmp = (uint8_t)(CID_Tab[0] & 0x000000FF);
cardinfo->SD_cid.ProdName1 = tmp << 24;

/*!< Byte 4 */
tmp = (uint8_t)((CID_Tab[1] & 0xFF000000) >> 24);
cardinfo->SD_cid.ProdName1 |= tmp << 16;

/*!< Byte 5 */
tmp = (uint8_t)((CID_Tab[1] & 0x00FF0000) >> 16);
cardinfo->SD_cid.ProdName1 |= tmp << 8;

/*!< Byte 6 */
tmp = (uint8_t)((CID_Tab[1] & 0x0000FF00) >> 8);
cardinfo->SD_cid.ProdName1 |= tmp;

/*!< Byte 7 */
tmp = (uint8_t)(CID_Tab[1] & 0x000000FF);
cardinfo->SD_cid.ProdName2 = tmp;

/*!< Byte 8 */
tmp = (uint8_t)((CID_Tab[2] & 0xFF000000) >> 24);
cardinfo->SD_cid.ProdRev = tmp;

/*!< Byte 9 */
tmp = (uint8_t)((CID_Tab[2] & 0x00FF0000) >> 16);
cardinfo->SD_cid.ProdSN = tmp << 24;

/*!< Byte 10 */
tmp = (uint8_t)((CID_Tab[2] & 0x0000FF00) >> 8);
cardinfo->SD_cid.ProdSN |= tmp << 16;
```

```
    /*!< Byte 11 */
    tmp = (uint8_t)(CID_Tab[2] & 0x000000FF);
    cardinfo->SD_cid.ProdSN |= tmp << 8;

    /*!< Byte 12 */
    tmp = (uint8_t)((CID_Tab[3] & 0xFF000000) >> 24);
    cardinfo->SD_cid.ProdSN |= tmp;

    /*!< Byte 13 */
    tmp = (uint8_t)((CID_Tab[3] & 0x00FF0000) >> 16);
    cardinfo->SD_cid.Reserved1 |= (tmp & 0xF0) >> 4;
    cardinfo->SD_cid.ManufactDate = (tmp & 0x0F) << 8;

    /*!< Byte 14 */
    tmp = (uint8_t)((CID_Tab[3] & 0x0000FF00) >> 8);
    cardinfo->SD_cid.ManufactDate |= tmp;

    /*!< Byte 15 */
    tmp = (uint8_t)(CID_Tab[3] & 0x000000FF);
    cardinfo->SD_cid.CID_CRC = (tmp & 0xFE) >> 1;
    cardinfo->SD_cid.Reserved2 = 1;

    return(errorstatus);
  }

  /**
    * @brief  Enables wide bus opeartion for the requeseted card if supported
by
    *         card.
    * @param  WideMode: Specifies the SD card wide bus mode.
    *   This parameter can be one of the following values:
    *     @arg SDIO_BusWide_8b: 8-bit data transfer (Only for MMC)
    *     @arg SDIO_BusWide_4b: 4-bit data transfer
    *     @arg SDIO_BusWide_1b: 1-bit data transfer
    * @retval SD_Error: SD Card Error code.
    */
  SD_Error SD_GetCardStatus(SD_CardStatus *cardstatus)
  {
    SD_Error errorstatus = SD_OK;
    uint8_t tmp = 0;

    errorstatus = SD_SendSDStatus((uint32_t *)SDSTATUS_Tab);
```

```
  if (errorstatus  != SD_OK)
  {
    return(errorstatus);
  }

  /*!< Byte 0 */
  tmp = (uint8_t)((SDSTATUS_Tab[0] & 0xC0) >> 6);
  cardstatus->DAT_BUS_WIDTH = tmp;

  /*!< Byte 0 */
  tmp = (uint8_t)((SDSTATUS_Tab[0] & 0x20) >> 5);
  cardstatus->SECURED_MODE = tmp;

  /*!< Byte 2 */
  tmp = (uint8_t)((SDSTATUS_Tab[2] & 0xFF));
  cardstatus->SD_CARD_TYPE = tmp << 8;

  /*!< Byte 3 */
  tmp = (uint8_t)((SDSTATUS_Tab[3] & 0xFF));
  cardstatus->SD_CARD_TYPE |= tmp;

  /*!< Byte 4 */
  tmp = (uint8_t)(SDSTATUS_Tab[4] & 0xFF);
  cardstatus->SIZE_OF_PROTECTED_AREA = tmp << 24;

  /*!< Byte 5 */
  tmp = (uint8_t)(SDSTATUS_Tab[5] & 0xFF);
  cardstatus->SIZE_OF_PROTECTED_AREA |= tmp << 16;

  /*!< Byte 6 */
  tmp = (uint8_t)(SDSTATUS_Tab[6] & 0xFF);
  cardstatus->SIZE_OF_PROTECTED_AREA |= tmp << 8;

  /*!< Byte 7 */
  tmp = (uint8_t)(SDSTATUS_Tab[7] & 0xFF);
  cardstatus->SIZE_OF_PROTECTED_AREA |= tmp;

  /*!< Byte 8 */
  tmp = (uint8_t)((SDSTATUS_Tab[8] & 0xFF));
  cardstatus->SPEED_CLASS = tmp;

  /*!< Byte 9 */
  tmp = (uint8_t)((SDSTATUS_Tab[9] & 0xFF));
  cardstatus->PERFORMANCE_MOVE = tmp;
```

```
  /*!< Byte 10 */
  tmp = (uint8_t)((SDSTATUS_Tab[10] & 0xF0) >> 4);
  cardstatus->AU_SIZE = tmp;

  /*!< Byte 11 */
  tmp = (uint8_t)(SDSTATUS_Tab[11] & 0xFF);
  cardstatus->ERASE_SIZE = tmp << 8;

  /*!< Byte 12 */
  tmp = (uint8_t)(SDSTATUS_Tab[12] & 0xFF);
  cardstatus->ERASE_SIZE |= tmp;

  /*!< Byte 13 */
  tmp = (uint8_t)((SDSTATUS_Tab[13] & 0xFC) >> 2);
  cardstatus->ERASE_TIMEOUT = tmp;

  /*!< Byte 13 */
  tmp = (uint8_t)((SDSTATUS_Tab[13] & 0x3));
  cardstatus->ERASE_OFFSET = tmp;

  return(errorstatus);
}


/*
 * 函数名：SD_EnableWideBusOperation
 * 描述  ：配置卡的数据宽度(但得看卡是否支持)
 * 输入  ：-WideMode 指定 SD 卡的数据线宽
 *         具体可配置如下
 *         @arg SDIO_BusWide_8b: 8-bit data transfer (Only for MMC)
 *         @arg SDIO_BusWide_4b: 4-bit data transfer
 *         @arg SDIO_BusWide_1b: 1-bit data transfer (默认)
 * 输出  ：-SD_Error SD 卡错误代码
 *         成功时则为 SD_OK
 * 调用  ：外部调用
 */
SD_Error SD_EnableWideBusOperation(uint32_t WideMode)
{
  SD_Error errorstatus = SD_OK;

  /*!< MMC Card doesn't support this feature */
  if (SDIO_MULTIMEDIA_CARD == CardType)
  {
```

```
        errorstatus = SD_UNSUPPORTED_FEATURE;
        return(errorstatus);
      }
      else if ((SDIO_STD_CAPACITY_SD_CARD_V1_1 == CardType) ||
(SDIO_STD_CAPACITY_SD_CARD_V2_0 == CardType) || (SDIO_HIGH_CAPACITY_SD_CARD ==
CardType))
      {
        if (SDIO_BusWide_8b == WideMode)    //2.0 sd 不支持 8bits
        {
          errorstatus = SD_UNSUPPORTED_FEATURE;
          return(errorstatus);
        }
        else if (SDIO_BusWide_4b == WideMode)//4 数据线模式
        {
          errorstatus = SDEnWideBus(ENABLE);//使用 acmd6 设置总线宽度，设置卡
的传输方式

          if (SD_OK == errorstatus)
          {
            /*!< Configure the SDIO peripheral */
            SDIO_InitStructure.SDIO_ClockDiv = SDIO_TRANSFER_CLK_DIV;
            SDIO_InitStructure.SDIO_ClockEdge = SDIO_ClockEdge_Rising;
            SDIO_InitStructure.SDIO_ClockBypass = SDIO_ClockBypass_Disable;
            SDIO_InitStructure.SDIO_ClockPowerSave =
SDIO_ClockPowerSave_Disable;
            SDIO_InitStructure.SDIO_BusWide = SDIO_BusWide_4b; //这个是设置
stm32 的 sdio 的传输方式 ， 切换模式必须从卡和 sdio 都对应好
            SDIO_InitStructure.SDIO_HardwareFlowControl =
SDIO_HardwareFlowControl_Disable;
            SDIO_Init(&SDIO_InitStructure);
          }
        }
        else//单数据线模式
        {
          errorstatus = SDEnWideBus(DISABLE);

          if (SD_OK == errorstatus)
          {
            /*!< Configure the SDIO peripheral */
            SDIO_InitStructure.SDIO_ClockDiv = SDIO_TRANSFER_CLK_DIV;
            SDIO_InitStructure.SDIO_ClockEdge = SDIO_ClockEdge_Rising;
            SDIO_InitStructure.SDIO_ClockBypass = SDIO_ClockBypass_Disable;
            SDIO_InitStructure.SDIO_ClockPowerSave =
SDIO_ClockPowerSave_Disable;
```

```
            SDIO_InitStructure.SDIO_BusWide = SDIO_BusWide_1b;
            SDIO_InitStructure.SDIO_HardwareFlowControl =
SDIO_HardwareFlowControl_Disable;
            SDIO_Init(&SDIO_InitStructure);
          }
        }
      }

      return(errorstatus);
    }

    /*
     * 函数名：SD_SelectDeselect
     * 描述  ：利用 cmd7，选择卡相对地址为 addr 的卡，取消选择其它卡
     *         如果 addr = 0,则取消选择所有的卡
     * 输入  ：-addr 选择卡的地址
     * 输出  ：-SD_Error SD 卡错误代码
     *         成功时则为 SD_OK
     * 调用  ：外部调用
     */
    SD_Error SD_SelectDeselect(uint32_t addr)
    {
      SD_Error errorstatus = SD_OK;

      /*!< Send CMD7 SDIO_SEL_DESEL_CARD */
      SDIO_CmdInitStructure.SDIO_Argument =  addr;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SEL_DESEL_CARD;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SEL_DESEL_CARD);

      return(errorstatus);
    }

    /**
      * @brief  Allows to read one block from a specified address in a card. The
Data
      *         transfer can be managed by DMA mode or Polling mode.
      * @note   This operation should be followed by two functions to check if
the
      *         DMA Controller and SD Card status.
```

```
      *          - SD_ReadWaitOperation(): this function insure that the DMA
      *            controller has finished all data transfer.
      *          - SD_GetStatus(): to check that the SD Card has finished the
      *            data transfer and it is ready for data.
      * @param  readbuff: pointer to the buffer that will contain the received
data
      * @param  ReadAddr: Address from where data are to be read.
      * @param  BlockSize: the SD card Data block size. The Block size should
be 512.
      * @retval SD_Error: SD Card Error code.
      */
    SD_Error SD_ReadBlock(uint8_t *readbuff, uint32_t ReadAddr, uint16_t
BlockSize)
    {
      SD_Error errorstatus = SD_OK;
    #if defined (SD_POLLING_MODE)
      uint32_t count = 0, *tempbuff = (uint32_t *)readbuff;
    #endif

      TransferError = SD_OK;
      TransferEnd = 0;    //传输结束标置位，在中断服务置 1
      StopCondition = 0;  //怎么用的?

      SDIO->DCTRL = 0x0;


      if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)
      {
        BlockSize = 512;
        ReadAddr /= 512;
      }
      /**********************add，没有这一段容易卡死在 DMA 检测中
************************************/
        /*!< Set Block Size for Card，cmd16,若是 sdsc 卡，可以用来设置块大小，
若是 sdhc 卡，块大小为 512 字节，不受 cmd16 影响 */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) BlockSize;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;   //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);
```

```
      if (SD_OK != errorstatus)
      {
        return(errorstatus);
      }

/*************************************************************************
******/
      SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;
      SDIO_DataInitStructure.SDIO_DataLength = BlockSize;
      SDIO_DataInitStructure.SDIO_DataBlockSize = (uint32_t) 9 << 4;
      SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToSDIO;
      SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block;
      SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;
      SDIO_DataConfig(&SDIO_DataInitStructure);

      /*!< Send CMD17 READ_SINGLE_BLOCK */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)ReadAddr;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_READ_SINGLE_BLOCK;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_READ_SINGLE_BLOCK);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }

    #if defined (SD_POLLING_MODE)
      /*!< In case of single block transfer, no need of stop transfer at all.*/
      /*!< Polling mode */
      while (!(SDIO->STA &(SDIO_FLAG_RXOVERR | SDIO_FLAG_DCRCFAIL |
SDIO_FLAG_DTIMEOUT | SDIO_FLAG_DBCKEND | SDIO_FLAG_STBITERR)))
      {
        if (SDIO_GetFlagStatus(SDIO_FLAG_RXFIFOHF) != RESET)
        {
          for (count = 0; count < 8; count++)
          {
            *(tempbuff + count) = SDIO_ReadData();
          }
          tempbuff += 8;
        }
```

```
  }

  if (SDIO_GetFlagStatus(SDIO_FLAG_DTIMEOUT) != RESET)
  {
    SDIO_ClearFlag(SDIO_FLAG_DTIMEOUT);
    errorstatus = SD_DATA_TIMEOUT;
    return(errorstatus);
  }
  else if (SDIO_GetFlagStatus(SDIO_FLAG_DCRCFAIL) != RESET)
  {
    SDIO_ClearFlag(SDIO_FLAG_DCRCFAIL);
    errorstatus = SD_DATA_CRC_FAIL;
    return(errorstatus);
  }
  else if (SDIO_GetFlagStatus(SDIO_FLAG_RXOVERR) != RESET)
  {
    SDIO_ClearFlag(SDIO_FLAG_RXOVERR);
    errorstatus = SD_RX_OVERRUN;
    return(errorstatus);
  }
  else if (SDIO_GetFlagStatus(SDIO_FLAG_STBITERR) != RESET)
  {
    SDIO_ClearFlag(SDIO_FLAG_STBITERR);
    errorstatus = SD_START_BIT_ERR;
    return(errorstatus);
  }
  while (SDIO_GetFlagStatus(SDIO_FLAG_RXDAVL) != RESET)
  {
    *tempbuff = SDIO_ReadData();
    tempbuff++;
  }

  /*!< Clear all the static flags */
  SDIO_ClearFlag(SDIO_STATIC_FLAGS);

#elif defined (SD_DMA_MODE)
    SDIO_ITConfig(SDIO_IT_DATAEND, ENABLE);
    SDIO_DMACmd(ENABLE);
    SD_DMA_RxConfig((uint32_t *)readbuff, BlockSize);
#endif

  return(errorstatus);
}
```

```
    /**
      * @brief  Allows to read blocks from a specified address  in a card.  The Data
      *         transfer can be managed by DMA mode or Polling mode. //分两个模式
      * @note   This operation should be followed by two functions to check if the
      *         DMA Controller and SD Card status.      //dma 模式时要调用以下两个函数
      *          - SD_ReadWaitOperation(): this function insure that the DMA
      *            controller has finished all data transfer.
      *          - SD_GetStatus(): to check that the SD Card has finished the
      *            data transfer and it is ready for data.
      * @param  readbuff: pointer to the buffer that will contain the received data.
      * @param  ReadAddr: Address from where data are to be read.
      * @param  BlockSize: the SD card Data block size. The Block size should be 512.
      * @param  NumberOfBlocks: number of blocks to be read.
      * @retval SD_Error: SD Card Error code.
      */
    SD_Error SD_ReadMultiBlocks(uint8_t *readbuff, uint32_t ReadAddr, uint16_t BlockSize, uint32_t NumberOfBlocks)
    {
      SD_Error errorstatus = SD_OK;
      TransferError = SD_OK;
      TransferEnd = 0;
      StopCondition = 1;

      SDIO->DCTRL = 0x0; //复位数据控制寄存器

      if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)//sdhc 卡的地址以块为单位，每块 512 字节
      {
        BlockSize = 512;
        ReadAddr /= 512;
      }

      /*!< Set Block Size for Card, cmd16,若是 sdsc 卡，可以用来设置块大小，若是 sdhc 卡，块大小为 512 字节，不受 cmd16 影响 */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) BlockSize;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;   //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
```

```
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);

    if (SD_OK != errorstatus)
    {
      return(errorstatus);
    }

    SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;     //等待超
时限制
    SDIO_DataInitStructure.SDIO_DataLength = NumberOfBlocks * BlockSize;
   //对于块数据传输，数据长度寄存器中的数值必须是数据块长度(见 SDIO_DCTRL)
的倍数
    SDIO_DataInitStructure.SDIO_DataBlockSize = (uint32_t) 9 << 4; //直接用
参数多好。。。SDIO_DataBlockSize_512b
    SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToSDIO;//传
输方向
    SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block; //
传输模式
    SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable; //开启数据状态机
    SDIO_DataConfig(&SDIO_DataInitStructure);

    /*!< Send CMD18 READ_MULT_BLOCK with argument data address */
    SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)ReadAddr;    //起始地
址
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_READ_MULT_BLOCK;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short; //r1
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp1Error(SD_CMD_READ_MULT_BLOCK);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    SDIO_ITConfig(SDIO_IT_DATAEND, ENABLE);  //开启数据传输结束中断 ，Data
end (data counter, SDIDCOUNT, is zero) interrupt
    SDIO_DMACmd(ENABLE); //使能 dma 方式
    SD_DMA_RxConfig((uint32_t *)readbuff, (NumberOfBlocks * BlockSize));//
```

```
配置 DMA 接收

      return(errorstatus);
    }

    /**
      * @brief  This function waits until the SDIO DMA data transfer is finished.
      *         This function should be called after SDIO_ReadMultiBlocks()
function
      *         to insure that all data sent by the card are already transferred
by
      *         the DMA controller.
      * @param  None.
      * @retval SD_Error: SD Card Error code.
      */
    SD_Error SD_WaitReadOperation(void)
    {
      SD_Error errorstatus = SD_OK;
             //等待 dma 传输结束
      while ((SD_DMAEndOfTransferStatus() == RESET) && (TransferEnd == 0) &&
(TransferError == SD_OK))
      {}

      if (TransferError != SD_OK)
      {
        return(TransferError);
      }

      return(errorstatus);
    }

    /**
      * @brief  Allows to write one block starting from a specified address in
a card.
      *         The Data transfer can be managed by DMA mode or Polling mode.
      * @note   This operation should be followed by two functions to check if
the
      *         DMA Controller and SD Card status.
      *          - SD_ReadWaitOperation(): this function insure that the DMA
      *            controller has finished all data transfer.
      *          - SD_GetStatus(): to check that the SD Card has finished the
      *            data transfer and it is ready for data.
      * @param  writebuff: pointer to the buffer that contain the data to be
transferred.
```

```
      * @param  WriteAddr: Address from where data are to be read.
      * @param  BlockSize: the SD card Data block size. The Block size should
be 512.
      * @retval SD_Error: SD Card Error code.
      */
    SD_Error SD_WriteBlock(uint8_t *writebuff, uint32_t WriteAddr, uint16_t
BlockSize)
    {
      SD_Error errorstatus = SD_OK;

    #if defined (SD_POLLING_MODE)
      uint32_t bytestransferred = 0, count = 0, restwords = 0;
      uint32_t *tempbuff = (uint32_t *)writebuff;
    #endif

      TransferError = SD_OK;
      TransferEnd = 0;
      StopCondition = 0;

      SDIO->DCTRL = 0x0;


      if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)
      {
        BlockSize = 512;
        WriteAddr /= 512;
      }

        /*******************add，没有这一段容易卡死在 DMA 检测中
************************************/
        /*!< Set Block Size for Card，cmd16,若是 sdsc 卡，可以用来设置块大小，
若是 sdhc 卡，块大小为 512 字节，不受 cmd16 影响 */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) BlockSize;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;   //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);

      if (SD_OK != errorstatus)
      {
        return(errorstatus);
```

```
      }

/*****************************************************************************
******/

      /*!< Send CMD24 WRITE_SINGLE_BLOCK */
      SDIO_CmdInitStructure.SDIO_Argument = WriteAddr;    //写入地址
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_WRITE_SINGLE_BLOCK;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;    //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_WRITE_SINGLE_BLOCK);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }
                          //配置 sdio 的写数据寄存器
      SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;
      SDIO_DataInitStructure.SDIO_DataLength = BlockSize;
      SDIO_DataInitStructure.SDIO_DataBlockSize = (uint32_t) 9 << 4;  //可用
此参数代替 SDIO_DataBlockSize_512b
      SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToCard;//写
数据，
      SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block;
      SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;  //开启数据通道
状态机
      SDIO_DataConfig(&SDIO_DataInitStructure);

      /*!< In case of single data block transfer no need of stop command at all
*/
    #if defined (SD_POLLING_MODE) //普通模式
      while (!(SDIO->STA & (SDIO_FLAG_DBCKEND | SDIO_FLAG_TXUNDERR |
SDIO_FLAG_DCRCFAIL | SDIO_FLAG_DTIMEOUT | SDIO_FLAG_STBITERR)))
      {
        if (SDIO_GetFlagStatus(SDIO_FLAG_TXFIFOHE) != RESET)
        {
          if ((512 - bytestransferred) < 32)
          {
            restwords = ((512 - bytestransferred) % 4 == 0) ? ((512 -
bytestransferred) / 4) : (( 512 -  bytestransferred) / 4 + 1);
            for (count = 0; count < restwords; count++, tempbuff++,
```

```
bytestransferred += 4)
            {
              SDIO_WriteData(*tempbuff);
            }
          }
          else
          {
            for (count = 0; count < 8; count++)
            {
              SDIO_WriteData(*(tempbuff + count));
            }
            tempbuff += 8;
            bytestransferred += 32;
          }
        }
      }
      if (SDIO_GetFlagStatus(SDIO_FLAG_DTIMEOUT) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_DTIMEOUT);
        errorstatus = SD_DATA_TIMEOUT;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_DCRCFAIL) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_DCRCFAIL);
        errorstatus = SD_DATA_CRC_FAIL;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_TXUNDERR) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_TXUNDERR);
        errorstatus = SD_TX_UNDERRUN;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_STBITERR) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_STBITERR);
        errorstatus = SD_START_BIT_ERR;
        return(errorstatus);
      }
    #elif defined (SD_DMA_MODE) //dma 模式
      SDIO_ITConfig(SDIO_IT_DATAEND, ENABLE);  //数据传输结束中断
      SD_DMA_TxConfig((uint32_t *)writebuff, BlockSize); //配置 dma，跟 rx 类
似
```

```
    SDIO_DMACmd(ENABLE);   // 使能 sdio 的 dma 请求
  #endif

    return(errorstatus);
  }

  /**
    * @brief  Allows to write blocks starting from a specified address in a
card.
    *         The Data transfer can be managed by DMA mode only.
    * @note   This operation should be followed by two functions to check if
the
    *         DMA Controller and SD Card status.
    *          - SD_ReadWaitOperation(): this function insure that the DMA
    *            controller has finished all data transfer.
    *          - SD_GetStatus(): to check that the SD Card has finished the
    *            data transfer and it is ready for data.
    * @param  WriteAddr: Address from where data are to be read.
    * @param  writebuff: pointer to the buffer that contain the data to be
transferred.
    * @param  BlockSize: the SD card Data block size. The Block size should
be 512.
    * @param  NumberOfBlocks: number of blocks to be written.
    * @retval SD_Error: SD Card Error code.
    */

    /*
   * 函数名：SD_WriteMultiBlocks
   * 描述  ：从输入的起始地址开始，向卡写入多个数据块，
          只能在 DMA 模式下使用这个函数
     注意：调用这个函数后一定要调用
           SD_WaitWriteOperation（）来等待 DMA 传输结束
           和 SD_GetStatus() 检测卡与 SDIO 的 FIFO 间是否已经完成传输
   * 输入  ：
          * @param  WriteAddr: Address from where data are to be read.
          * @param  writebuff: pointer to the buffer that contain the data
to be transferred.
          * @param  BlockSize: the SD card Data block size. The Block size
should be 512.
          * @param  NumberOfBlocks: number of blocks to be written.
   * 输出  ：SD 错误类型
   */
  SD_Error SD_WriteMultiBlocks(uint8_t *writebuff, uint32_t WriteAddr,
uint16_t BlockSize, uint32_t NumberOfBlocks)
```

```
    {
      SD_Error errorstatus = SD_OK;
      __IO uint32_t count = 0;

      TransferError = SD_OK;
      TransferEnd = 0;
      StopCondition = 1;

      SDIO->DCTRL = 0x0;

      if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)
      {
        BlockSize = 512;
        WriteAddr /= 512;
      }

        /*******************add，没有这一段容易卡死在 DMA 检测中
************************************/
        /*!< Set Block Size for Card，cmd16,若是 sdsc 卡，可以用来设置块大小，
若是 sdhc 卡，块大小为 512 字节，不受 cmd16 影响 */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) BlockSize;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;   //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);

      if (SD_OK != errorstatus)
      {
        return(errorstatus);
      }

/*************************************************************************
******/

      /*!< To improve performance  */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) (RCA << 16);
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;// cmd55
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);
```

```
      errorstatus = CmdResp1Error(SD_CMD_APP_CMD);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }
      /*!< To improve performance *///  pre-erased，在多块写入时可发送此命令进行预擦除
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)NumberOfBlocks;  //参数为将要写入的块数目
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCK_COUNT;
//cmd23
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SET_BLOCK_COUNT);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }


      /*!< Send CMD25 WRITE_MULT_BLOCK with argument data address */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)WriteAddr;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_WRITE_MULT_BLOCK;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_WRITE_MULT_BLOCK);

      if (SD_OK != errorstatus)
      {
        return(errorstatus);
      }

      SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;
      SDIO_DataInitStructure.SDIO_DataLength = NumberOfBlocks * BlockSize;
```

```
    SDIO_DataInitStructure.SDIO_DataBlockSize = (uint32_t) 9 << 4;
    SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToCard;
    SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block;
    SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;
    SDIO_DataConfig(&SDIO_DataInitStructure);

    SDIO_ITConfig(SDIO_IT_DATAEND, ENABLE);
    SDIO_DMACmd(ENABLE);
    SD_DMA_TxConfig((uint32_t *)writebuff, (NumberOfBlocks * BlockSize));

    return(errorstatus);
  }

  /**
    * @brief  This function waits until the SDIO DMA data transfer is finished.
    *         This function should be called after SDIO_WriteBlock() and
    *         SDIO_WriteMultiBlocks() function to insure that all data sent
by the
    *         card are already transferred by the DMA controller.
    * @param  None.
    * @retval SD_Error: SD Card Error code.
    */
  SD_Error SD_WaitWriteOperation(void)
  {
    SD_Error errorstatus = SD_OK;
          //等待 dma 是否传输结束
    while ((SD_DMAEndOfTransferStatus() == RESET) && (TransferEnd == 0) &&
(TransferError == SD_OK))
    {}

    if (TransferError != SD_OK)
    {
      return(TransferError);
    }

    /*!< Clear all the static flags */
    SDIO_ClearFlag(SDIO_STATIC_FLAGS);

    return(errorstatus);
  }

  /**
    * @brief  Gets the cuurent data transfer state.
    * @param  None
```

```
  * @retval SDTransferState: Data Transfer state.
  *   This value can be:
  *        - SD_TRANSFER_OK: No data transfer is acting
  *        - SD_TRANSFER_BUSY: Data transfer is acting
  */
SDTransferState SD_GetTransferState(void)
{
  if (SDIO->STA & (SDIO_FLAG_TXACT | SDIO_FLAG_RXACT))
  {
    return(SD_TRANSFER_BUSY);
  }
  else
  {
    return(SD_TRANSFER_OK);
  }
}

/**
  * @brief  Aborts an ongoing data transfer.
  * @param  None
  * @retval SD_Error: SD Card Error code.
  */
SD_Error SD_StopTransfer(void)
{
  SD_Error errorstatus = SD_OK;

  /*!< Send CMD12 STOP_TRANSMISSION  */
  SDIO->ARG = 0x0;
  SDIO->CMD = 0x44C;
  errorstatus = CmdResp1Error(SD_CMD_STOP_TRANSMISSION);

  return(errorstatus);
}

/**
  * @brief  Allows to erase memory area specified for the given card.
  * @param  startaddr: the start address.
  * @param  endaddr: the end address.
  * @retval SD_Error: SD Card Error code.
  */
SD_Error SD_Erase(uint32_t startaddr, uint32_t endaddr)
{
  SD_Error errorstatus = SD_OK;
  uint32_t delay = 0;
```

```
    __IO uint32_t maxdelay = 0;
    uint8_t cardstate = 0;

    /*!< Check if the card coomnd class supports erase command */
    if (((CSD_Tab[1] >> 20) & SD_CCCC_ERASE) == 0)
    {
      errorstatus = SD_REQUEST_NOT_APPLICABLE;
      return(errorstatus);
    }

    maxdelay = 120000 / ((SDIO->CLKCR & 0xFF) + 2);//延时，根据时钟分频设置
来计算

    if (SDIO_GetResponse(SDIO_RESP1) & SD_CARD_LOCKED)      //卡已上锁
    {
      errorstatus = SD_LOCK_UNLOCK_FAILED;
      return(errorstatus);
    }

    if (CardType == SDIO_HIGH_CAPACITY_SD_CARD)//sdhc 卡，为什么要 /512？详
见 2.0 协议 page52
    {                       //在 sdhc 卡,地址参数为块地址,每专块 512 字节,sdsc
卡地址为字节地址
      startaddr /= 512;
      endaddr /= 512;
    }

    /*!< According to sd-card spec 1.0 ERASE_GROUP_START (CMD32) and
erase_group_end(CMD33) */
    if ((SDIO_STD_CAPACITY_SD_CARD_V1_1 == CardType) ||
(SDIO_STD_CAPACITY_SD_CARD_V2_0 == CardType) || (SDIO_HIGH_CAPACITY_SD_CARD ==
CardType))
    {
      /*!< Send CMD32 SD_ERASE_GRP_START with argument as addr  */
      SDIO_CmdInitStructure.SDIO_Argument = startaddr;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SD_ERASE_GRP_START;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short; //R1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SD_ERASE_GRP_START);
      if (errorstatus != SD_OK)
      {
```

```
        return(errorstatus);
      }

      /*!< Send CMD33 SD_ERASE_GRP_END with argument as addr  */
      SDIO_CmdInitStructure.SDIO_Argument = endaddr;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SD_ERASE_GRP_END;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SD_ERASE_GRP_END);
      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }
    }

    /*!< Send CMD38 ERASE */
    SDIO_CmdInitStructure.SDIO_Argument = 0;
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_ERASE;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp1Error(SD_CMD_ERASE);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    for (delay = 0; delay < maxdelay; delay++)
    {}

    /*!< Wait till the card is in programming state */
    errorstatus = IsCardProgramming(&cardstate);

    while ((errorstatus == SD_OK) && ((SD_CARD_PROGRAMMING == cardstate) ||
(SD_CARD_RECEIVING == cardstate)))
    {
      errorstatus = IsCardProgramming(&cardstate);
    }
```

```
  return(errorstatus);
}

/**
  * @brief  Returns the current card's status.
  * @param  pcardstatus: pointer to the buffer that will contain the SD card
  *         status (Card Status register).
  * @retval SD_Error: SD Card Error code.
  */
SD_Error SD_SendStatus(uint32_t *pcardstatus)
{
  SD_Error errorstatus = SD_OK;

  SDIO->ARG = (uint32_t) RCA << 16;
  SDIO->CMD = 0x44D;

  errorstatus = CmdResp1Error(SD_CMD_SEND_STATUS);

  if (errorstatus != SD_OK)
  {
    return(errorstatus);
  }

  *pcardstatus = SDIO->RESP1;
  return(errorstatus);
}

/**
  * @brief  Returns the current SD card's status.
  * @param  psdstatus: pointer to the buffer that will contain the SD card
status
  *         (SD Status register).
  * @retval SD_Error: SD Card Error code.
  */
SD_Error SD_SendSDStatus(uint32_t *psdstatus)
{
  SD_Error errorstatus = SD_OK;
  uint32_t count = 0;

  if (SDIO_GetResponse(SDIO_RESP1) & SD_CARD_LOCKED)
  {
    errorstatus = SD_LOCK_UNLOCK_FAILED;
    return(errorstatus);
```

```
    }

    /*!< Set block size for card if it is not equal to current block size for
card. */
    SDIO_CmdInitStructure.SDIO_Argument = 64;
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    /*!< CMD55 */
    SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) RCA << 16;
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);
    errorstatus = CmdResp1Error(SD_CMD_APP_CMD);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;
    SDIO_DataInitStructure.SDIO_DataLength = 64;
    SDIO_DataInitStructure.SDIO_DataBlockSize = SDIO_DataBlockSize_64b;
    SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToSDIO;
    SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block;
    SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;
    SDIO_DataConfig(&SDIO_DataInitStructure);

    /*!< Send ACMD13 SD_APP_STAUS  with argument as card's RCA.*/
    SDIO_CmdInitStructure.SDIO_Argument = 0;
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SD_APP_STAUS;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
```

```
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);
    errorstatus = CmdResp1Error(SD_CMD_SD_APP_STAUS);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    while (!(SDIO->STA &(SDIO_FLAG_RXOVERR | SDIO_FLAG_DCRCFAIL |
SDIO_FLAG_DTIMEOUT | SDIO_FLAG_DBCKEND | SDIO_FLAG_STBITERR)))
    {
      if (SDIO_GetFlagStatus(SDIO_FLAG_RXFIFOHF) != RESET)
      {
        for (count = 0; count < 8; count++)
        {
          *(psdstatus + count) = SDIO_ReadData();
        }
        psdstatus += 8;
      }
    }

    if (SDIO_GetFlagStatus(SDIO_FLAG_DTIMEOUT) != RESET)
    {
      SDIO_ClearFlag(SDIO_FLAG_DTIMEOUT);
      errorstatus = SD_DATA_TIMEOUT;
      return(errorstatus);
    }
    else if (SDIO_GetFlagStatus(SDIO_FLAG_DCRCFAIL) != RESET)
    {
      SDIO_ClearFlag(SDIO_FLAG_DCRCFAIL);
      errorstatus = SD_DATA_CRC_FAIL;
      return(errorstatus);
    }
    else if (SDIO_GetFlagStatus(SDIO_FLAG_RXOVERR) != RESET)
    {
      SDIO_ClearFlag(SDIO_FLAG_RXOVERR);
      errorstatus = SD_RX_OVERRUN;
      return(errorstatus);
    }
    else if (SDIO_GetFlagStatus(SDIO_FLAG_STBITERR) != RESET)
    {
      SDIO_ClearFlag(SDIO_FLAG_STBITERR);
```

```
      errorstatus = SD_START_BIT_ERR;
      return(errorstatus);
    }

    while (SDIO_GetFlagStatus(SDIO_FLAG_RXDAVL) != RESET)
    {
      *psdstatus = SDIO_ReadData();
      psdstatus++;
    }

    /*!< Clear all the static status flags*/
    SDIO_ClearFlag(SDIO_STATIC_FLAGS);

    return(errorstatus);
  }

  /*
   * 函数名：SD_ProcessIRQSrc
   * 描述  ：数据传输结束中断
   * 输入  ：无
   * 输出  ：SD 错误类型
   */
  SD_Error SD_ProcessIRQSrc(void)
  {
    if (StopCondition == 1)  //什么时候置 1 了？
    {
      SDIO->ARG = 0x0;   //命令参数寄存器
      SDIO->CMD = 0x44C;    // 命令寄存器：  0100   01     001100
                        //                    [7:6]     [5:0]
                        //            CPSMEN  WAITRESP CMDINDEX
                        //     开启命令状态机    短响应    cmd12 STOP_
TRANSMISSION
      TransferError = CmdResp1Error(SD_CMD_STOP_TRANSMISSION);
    }
    else
    {
      TransferError = SD_OK;
    }
    SDIO_ClearITPendingBit(SDIO_IT_DATAEND); //清中断
    SDIO_ITConfig(SDIO_IT_DATAEND, DISABLE); //关闭 sdio 中断使能
    TransferEnd = 1;
    return(TransferError);
  }
```

```
 /*
 * 函数名：CmdError
 * 描述  ：对 CMD0 命令的检查。
 * 输入  ：无
 * 输出  ：SD 错误类型
 */
static SD_Error CmdError(void)
{
  SD_Error errorstatus = SD_OK;
  uint32_t timeout;

  timeout = SDIO_CMD0TIMEOUT; /*!< 10000 */

    /*检查命令是否已发送*/
  while ((timeout > 0) && (SDIO_GetFlagStatus(SDIO_FLAG_CMDSENT) == RESET))

  {
    timeout--;
  }

  if (timeout == 0)
  {
    errorstatus = SD_CMD_RSP_TIMEOUT;
    return(errorstatus);
  }

  /*!< Clear all the static flags */
  SDIO_ClearFlag(SDIO_STATIC_FLAGS);//清除静态标志位

  return(errorstatus);
}


 /*
 * 函数名：CmdResp7Error
 * 描述  ：对响应类型为 R7 的命令进行检查
 * 输入  ：无
 * 输出  ：SD 错误类型
 */
static SD_Error CmdResp7Error(void)
{
  SD_Error errorstatus = SD_OK;
  uint32_t status;
```

```
    uint32_t timeout = SDIO_CMD0TIMEOUT;

    status = SDIO->STA;   //读取 SDIO 状态寄存器 ，此状态寄存器是 stm32 的寄存器

    /* Command response received (CRC check failed) : Command response received (CRC check passed): Command response timeout */

    while (!(status & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CMDREND | SDIO_FLAG_CTIMEOUT)) && (timeout > 0))
    {
      timeout--;
      status = SDIO->STA;
    }
      //卡不响应 cmd8
    if ((timeout == 0) || (status & SDIO_FLAG_CTIMEOUT))
    {
      /*!< Card is not V2.0 complient or card does not support the set voltage range */
      errorstatus = SD_CMD_RSP_TIMEOUT;
      SDIO_ClearFlag(SDIO_FLAG_CTIMEOUT);
      return(errorstatus);
    }

    if (status & SDIO_FLAG_CMDREND)
    {
      /*!< Card is SD V2.0 compliant */
      errorstatus = SD_OK;
      SDIO_ClearFlag(SDIO_FLAG_CMDREND);
      return(errorstatus);
    }
    return(errorstatus);
  }


   /*
   * 函数名：CmdResp1Error
   * 描述  ：对响应类型为 R1 的命令进行检查
   * 输入  ：无
   * 输出  ：SD 错误类型
   */
  static SD_Error CmdResp1Error(uint8_t cmd) //传入的参数有什么用?
  {
    /*不是这些状态就等待 */
```

```
      while (!(SDIO->STA & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CMDREND |
SDIO_FLAG_CTIMEOUT)))
      {
      }

      SDIO->ICR = SDIO_STATIC_FLAGS;   //清中断标志

      return (SD_Error)(SDIO->RESP1 &  SD_OCR_ERRORBITS);       //判断是否在
供电范围
    }

     /*
     * 函数名：CmdResp3Error
     * 描述  ：对响应类型为 R3 的命令进行检查
     * 输入  ：无
     * 输出  ：SD 错误类型
     */
    static SD_Error CmdResp3Error(void)
    {
      SD_Error errorstatus = SD_OK;
      uint32_t status;

      status = SDIO->STA;

      while (!(status & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CMDREND |
SDIO_FLAG_CTIMEOUT)))
      {
        status = SDIO->STA;
      }

      if (status & SDIO_FLAG_CTIMEOUT)
      {
        errorstatus = SD_CMD_RSP_TIMEOUT;
        SDIO_ClearFlag(SDIO_FLAG_CTIMEOUT);
        return(errorstatus);
      }
      /*!< Clear all the static flags */
      SDIO_ClearFlag(SDIO_STATIC_FLAGS);
      return(errorstatus);
    }

     /*
     * 函数名：CmdResp2Error
     * 描述  ：对响应类型为 R2 的命令进行检查
```

```
     * 输入  : 无
     * 输出  : SD 错误类型
     */
    static SD_Error CmdResp2Error(void)
    {
      SD_Error errorstatus = SD_OK;
      uint32_t status;

      status = SDIO->STA;

      while (!(status & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CTIMEOUT |
SDIO_FLAG_CMDREND)))
      {
        status = SDIO->STA;
      }

      if (status & SDIO_FLAG_CTIMEOUT)
      {
        errorstatus = SD_CMD_RSP_TIMEOUT;
        SDIO_ClearFlag(SDIO_FLAG_CTIMEOUT);
        return(errorstatus);
      }
      else if (status & SDIO_FLAG_CCRCFAIL)
      {
        errorstatus = SD_CMD_CRC_FAIL;
        SDIO_ClearFlag(SDIO_FLAG_CCRCFAIL);
        return(errorstatus);
      }

      /*!< Clear all the static flags */
      SDIO_ClearFlag(SDIO_STATIC_FLAGS);

      return(errorstatus);
    }


     /*
     * 函数名：CmdResp6Error
     * 描述  : 对响应类型为 R6 的命令进行检查
     * 输入  : cmd 命令索引号，
              prca 用来存储接收到的卡相对地址
     * 输出  : SD 错误类型
     */
    static SD_Error CmdResp6Error(uint8_t cmd, uint16_t *prca)
    {
```

```
      SD_Error errorstatus = SD_OK;
      uint32_t status;
      uint32_t response_r1;

      status = SDIO->STA;

      while (!(status & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CTIMEOUT |
SDIO_FLAG_CMDREND)))
      {
        status = SDIO->STA;
      }

      if (status & SDIO_FLAG_CTIMEOUT)
      {
        errorstatus = SD_CMD_RSP_TIMEOUT;
        SDIO_ClearFlag(SDIO_FLAG_CTIMEOUT);
        return(errorstatus);
      }
      else if (status & SDIO_FLAG_CCRCFAIL)
      {
        errorstatus = SD_CMD_CRC_FAIL;
        SDIO_ClearFlag(SDIO_FLAG_CCRCFAIL);
        return(errorstatus);
      }

      /*!< Check response received is of desired command */
      if (SDIO_GetCommandResponse() != cmd)       //检测是否接收到正常命令
      {
        errorstatus = SD_ILLEGAL_CMD;
        return(errorstatus);
      }

      /*!< Clear all the static flags */
      SDIO_ClearFlag(SDIO_STATIC_FLAGS);

      /*!< We have received response, retrieve it.  */
      response_r1 = SDIO_GetResponse(SDIO_RESP1);

       /*以下状态全为 0 表明成功接收到 card 返回的 rca */
      if (SD_ALLZERO == (response_r1 & (SD_R6_GENERAL_UNKNOWN_ERROR |
SD_R6_ILLEGAL_CMD | SD_R6_COM_CRC_FAILED)))
      {
        *prca = (uint16_t) (response_r1 >> 16);//右移 16 位，就是接收到的返回
rca
```

```
    return(errorstatus);
  }

  if (response_r1 & SD_R6_GENERAL_UNKNOWN_ERROR)
  {
    return(SD_GENERAL_UNKNOWN_ERROR);
  }

  if (response_r1 & SD_R6_ILLEGAL_CMD)
  {
    return(SD_ILLEGAL_CMD);
  }

  if (response_r1 & SD_R6_COM_CRC_FAILED)
  {
    return(SD_COM_CRC_FAILED);
  }

  return(errorstatus);
}

  /*
 * 函数名：SDEnWideBus
 * 描述  ：使能或关闭 SDIO 的 4bit 模式
 * 输入  ：新状态   ENABLE 或 DISABLE
 * 输出  ：SD 错误类型
 */
static SD_Error SDEnWideBus(FunctionalState NewState)
{
  SD_Error errorstatus = SD_OK;

  uint32_t scr[2] = {0, 0};

  if (SDIO_GetResponse(SDIO_RESP1) & SD_CARD_LOCKED) //检测卡是否已上锁
  {
    errorstatus = SD_LOCK_UNLOCK_FAILED;
    return(errorstatus);
  }

  /*!< Get SCR Register */
  errorstatus = FindSCR(RCA, scr);//获取 scr 寄存器内容到 scr 数组中

  if (errorstatus != SD_OK)       //degug,crc 错误，scr 读取不了数值
  {
```

```
    return(errorstatus);
  }

  /*!< If wide bus operation to be enabled */
  if (NewState == ENABLE)
  {
    /*!< If requested card supports wide bus operation */
    if ((scr[1] & SD_WIDE_BUS_SUPPORT) != SD_ALLZERO)  //判断卡是否支持 4
位方式
    {
      /*!< Send CMD55 APP_CMD with argument as card's RCA.*/
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) RCA << 16;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_APP_CMD);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }

      /*!< Send ACMD6 APP_CMD with argument as 2 for wide bus mode */
     /*开启 4bit 模式的命令 acmd6*/
      SDIO_CmdInitStructure.SDIO_Argument = 0x2;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_SD_SET_BUSWIDTH;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_APP_SD_SET_BUSWIDTH);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }
      return(errorstatus);
    }
    else
    {
```

```
      errorstatus = SD_REQUEST_NOT_APPLICABLE;
      return(errorstatus);
    }
  }   /*!< If wide bus operation to be disabled */
  else
  {
    /*!< If requested card supports 1 bit mode operation */
    if ((scr[1] & SD_SINGLE_BUS_SUPPORT) != SD_ALLZERO)
    {
      /*!< Send CMD55 APP_CMD with argument as card's RCA.*/
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) RCA << 16;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);


      errorstatus = CmdResp1Error(SD_CMD_APP_CMD);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }

      /*!< Send ACMD6 APP_CMD with argument as 0 for single bus mode */
      SDIO_CmdInitStructure.SDIO_Argument = 0x00;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_SD_SET_BUSWIDTH;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_APP_SD_SET_BUSWIDTH);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }

      return(errorstatus);
    }
    else
    {
```

```
          errorstatus = SD_REQUEST_NOT_APPLICABLE;
          return(errorstatus);
        }
      }
    }


     /*
     * 函数名：IsCardProgramming
     * 描述  ：检测 SD 卡是不是正在进行内部读写操作
     * 输入  ：用来装载 SD state 状态的指针
     * 输出  ：SD 错误类型
     */
    static SD_Error IsCardProgramming(uint8_t *pstatus)
    {
      SD_Error errorstatus = SD_OK;
      __IO uint32_t respR1 = 0, status = 0;

        /*cmd13 让卡发送卡状态寄存器，存储到 m3 的位置为 sdio_sta 寄存器*/
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) RCA << 16; //卡相对地
址参数
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SEND_STATUS;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      status = SDIO->STA;
      while (!(status & (SDIO_FLAG_CCRCFAIL | SDIO_FLAG_CMDREND |
SDIO_FLAG_CTIMEOUT)))
      {
        status = SDIO->STA;
      }
         /*一系列的状态判断*/
      if (status & SDIO_FLAG_CTIMEOUT)
      {
        errorstatus = SD_CMD_RSP_TIMEOUT;
        SDIO_ClearFlag(SDIO_FLAG_CTIMEOUT);
        return(errorstatus);
      }
      else if (status & SDIO_FLAG_CCRCFAIL)
      {
        errorstatus = SD_CMD_CRC_FAIL;
        SDIO_ClearFlag(SDIO_FLAG_CCRCFAIL);
```

```
    return(errorstatus);
  }

  status = (uint32_t)SDIO_GetCommandResponse();

  /*!< Check response received is of desired command */
  if (status != SD_CMD_SEND_STATUS)
  {
    errorstatus = SD_ILLEGAL_CMD;
    return(errorstatus);
  }

  /*!< Clear all the static flags */
  SDIO_ClearFlag(SDIO_STATIC_FLAGS);


  /*!< We have received response, retrieve it for analysis  */
  respR1 = SDIO_GetResponse(SDIO_RESP1);

  /*!< Find out card status */
  *pstatus = (uint8_t) ((respR1 >> 9) & 0x0000000F);
//status[12:9] :cardstate

  if ((respR1 & SD_OCR_ERRORBITS) == SD_ALLZERO)
  {
    return(errorstatus);
  }

  if (respR1 & SD_OCR_ADDR_OUT_OF_RANGE)
  {
    return(SD_ADDR_OUT_OF_RANGE);
  }

  if (respR1 & SD_OCR_ADDR_MISALIGNED)
  {
    return(SD_ADDR_MISALIGNED);
  }

  if (respR1 & SD_OCR_BLOCK_LEN_ERR)
  {
    return(SD_BLOCK_LEN_ERR);
  }

  if (respR1 & SD_OCR_ERASE_SEQ_ERR)
```

```
  {
    return(SD_ERASE_SEQ_ERR);
  }

  if (respR1 & SD_OCR_BAD_ERASE_PARAM)
  {
    return(SD_BAD_ERASE_PARAM);
  }

  if (respR1 & SD_OCR_WRITE_PROT_VIOLATION)
  {
    return(SD_WRITE_PROT_VIOLATION);
  }

  if (respR1 & SD_OCR_LOCK_UNLOCK_FAILED)
  {
    return(SD_LOCK_UNLOCK_FAILED);
  }

  if (respR1 & SD_OCR_COM_CRC_FAILED)
  {
    return(SD_COM_CRC_FAILED);
  }

  if (respR1 & SD_OCR_ILLEGAL_CMD)
  {
    return(SD_ILLEGAL_CMD);
  }

  if (respR1 & SD_OCR_CARD_ECC_FAILED)
  {
    return(SD_CARD_ECC_FAILED);
  }

  if (respR1 & SD_OCR_CC_ERROR)
  {
    return(SD_CC_ERROR);
  }

  if (respR1 & SD_OCR_GENERAL_UNKNOWN_ERROR)
  {
    return(SD_GENERAL_UNKNOWN_ERROR);
  }
```

```
  if (respR1 & SD_OCR_STREAM_READ_UNDERRUN)
  {
    return(SD_STREAM_READ_UNDERRUN);
  }

  if (respR1 & SD_OCR_STREAM_WRITE_OVERRUN)
  {
    return(SD_STREAM_WRITE_OVERRUN);
  }

  if (respR1 & SD_OCR_CID_CSD_OVERWRIETE)
  {
    return(SD_CID_CSD_OVERWRITE);
  }

  if (respR1 & SD_OCR_WP_ERASE_SKIP)
  {
    return(SD_WP_ERASE_SKIP);
  }

  if (respR1 & SD_OCR_CARD_ECC_DISABLED)
  {
    return(SD_CARD_ECC_DISABLED);
  }

  if (respR1 & SD_OCR_ERASE_RESET)
  {
    return(SD_ERASE_RESET);
  }

  if (respR1 & SD_OCR_AKE_SEQ_ERROR)
  {
    return(SD_AKE_SEQ_ERROR);
  }

  return(errorstatus);
}


  /*
 * 函数名：FindSCR
 * 描述  ：读取 SD 卡的 SCR 寄存器的内容
 * 输入  ：RCA 卡相对地址
          pscr    用来装载 SCR 内容的指针
```

```
     * 输出  : SD 错误类型
     */
    static SD_Error FindSCR(uint16_t rca, uint32_t *pscr)
    {

      uint32_t index = 0;
      SD_Error errorstatus = SD_OK;
      uint32_t tempscr[2] = {0, 0};

      /*!< Set Block Size To 8 Bytes */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t)8;     //块大小如果是
sdhc 卡是无法改变块大小的    //原参数 8
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SET_BLOCKLEN; //    cmd16
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;  //r1
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_SET_BLOCKLEN);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }

      /*!< Send CMD55 APP_CMD with argument as card's RCA */
      SDIO_CmdInitStructure.SDIO_Argument = (uint32_t) RCA << 16;
      SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_APP_CMD;
      SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;
      SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
      SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
      SDIO_SendCommand(&SDIO_CmdInitStructure);

      errorstatus = CmdResp1Error(SD_CMD_APP_CMD);

      if (errorstatus != SD_OK)
      {
        return(errorstatus);
      }
      /*设置数据接收寄存器*/
      SDIO_DataInitStructure.SDIO_DataTimeOut = SD_DATATIMEOUT;
      SDIO_DataInitStructure.SDIO_DataLength = 8;  //8byte,64 位
      SDIO_DataInitStructure.SDIO_DataBlockSize = SDIO_DataBlockSize_8b  ;
//块大小 8byte
```

```
    SDIO_DataInitStructure.SDIO_TransferDir = SDIO_TransferDir_ToSDIO;
    SDIO_DataInitStructure.SDIO_TransferMode = SDIO_TransferMode_Block;
    SDIO_DataInitStructure.SDIO_DPSM = SDIO_DPSM_Enable;
    SDIO_DataConfig(&SDIO_DataInitStructure);

    /*!< Send ACMD51 SD_APP_SEND_SCR with argument as 0 */
    SDIO_CmdInitStructure.SDIO_Argument = 0x0;
    SDIO_CmdInitStructure.SDIO_CmdIndex = SD_CMD_SD_APP_SEND_SCR;
    SDIO_CmdInitStructure.SDIO_Response = SDIO_Response_Short;  //r1
    SDIO_CmdInitStructure.SDIO_Wait = SDIO_Wait_No;
    SDIO_CmdInitStructure.SDIO_CPSM = SDIO_CPSM_Enable;
    SDIO_SendCommand(&SDIO_CmdInitStructure);

    errorstatus = CmdResp1Error(SD_CMD_SD_APP_SEND_SCR);

    if (errorstatus != SD_OK)
    {
      return(errorstatus);
    }

    /*等待接收数据 */
    /*不是这些情况就循环*/

   /*上溢出错     //数据 crc 失败      //数据超时    //已接收数据块，crc 检测成
功  //没有在所有数据线上检测到起始信号*/
    while (!(SDIO->STA & (SDIO_FLAG_RXOVERR | SDIO_FLAG_DCRCFAIL |
SDIO_FLAG_DTIMEOUT | SDIO_FLAG_DBCKEND| SDIO_FLAG_STBITERR)))
     {
     if (SDIO_GetFlagStatus(SDIO_FLAG_RXDAVL) != RESET)   //接收到的数据是
否可用
     {
           *(tempscr + index) = SDIO_ReadData();
              index++;

        /*   //add。这段在官方源码没有加判断            */
           if(index > 1 )
              break;
      }

    }

    if (SDIO_GetFlagStatus(SDIO_FLAG_DTIMEOUT) != RESET)
    {
      SDIO_ClearFlag(SDIO_FLAG_DTIMEOUT);
```

```
        errorstatus = SD_DATA_TIMEOUT;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_DCRCFAIL) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_DCRCFAIL);
        errorstatus = SD_DATA_CRC_FAIL;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_RXOVERR) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_RXOVERR);
        errorstatus = SD_RX_OVERRUN;
        return(errorstatus);
      }
      else if (SDIO_GetFlagStatus(SDIO_FLAG_STBITERR) != RESET)
      {
        SDIO_ClearFlag(SDIO_FLAG_STBITERR);
        errorstatus = SD_START_BIT_ERR;
        return(errorstatus);
      }

      /*!< Clear all the static flags */
      SDIO_ClearFlag(SDIO_STATIC_FLAGS);

      *(pscr + 1) = ((tempscr[0] & SD_0TO7BITS) << 24) | ((tempscr[0] &
SD_8TO15BITS) << 8) | ((tempscr[0] & SD_16TO23BITS) >> 8) | ((tempscr[0] &
SD_24TO31BITS) >> 24);

      *(pscr) = ((tempscr[1] & SD_0TO7BITS) << 24) | ((tempscr[1] & SD_8TO15BITS)
<< 8) | ((tempscr[1] & SD_16TO23BITS) >> 8) | ((tempscr[1] & SD_24TO31BITS) >>
24);

      return(errorstatus);
    }


    /**
      * @brief  Converts the number of bytes in power of two and returns the
power.
      * @param  NumberOfBytes: number of bytes.
      * @retval None
      */
    uint8_t convert_from_bytes_to_power_of_two(uint16_t NumberOfBytes)
```

```
{
  uint8_t count = 0;

  while (NumberOfBytes != 1)
  {
    NumberOfBytes >>= 1;
    count++;
  }
  return(count);
}

/*****************************END OF FILE*****************************/
```

4.35.20 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM3。2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。

本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\2,外设篇\225.SD 卡初始化实验\程序）

4.35.21 实验效果图

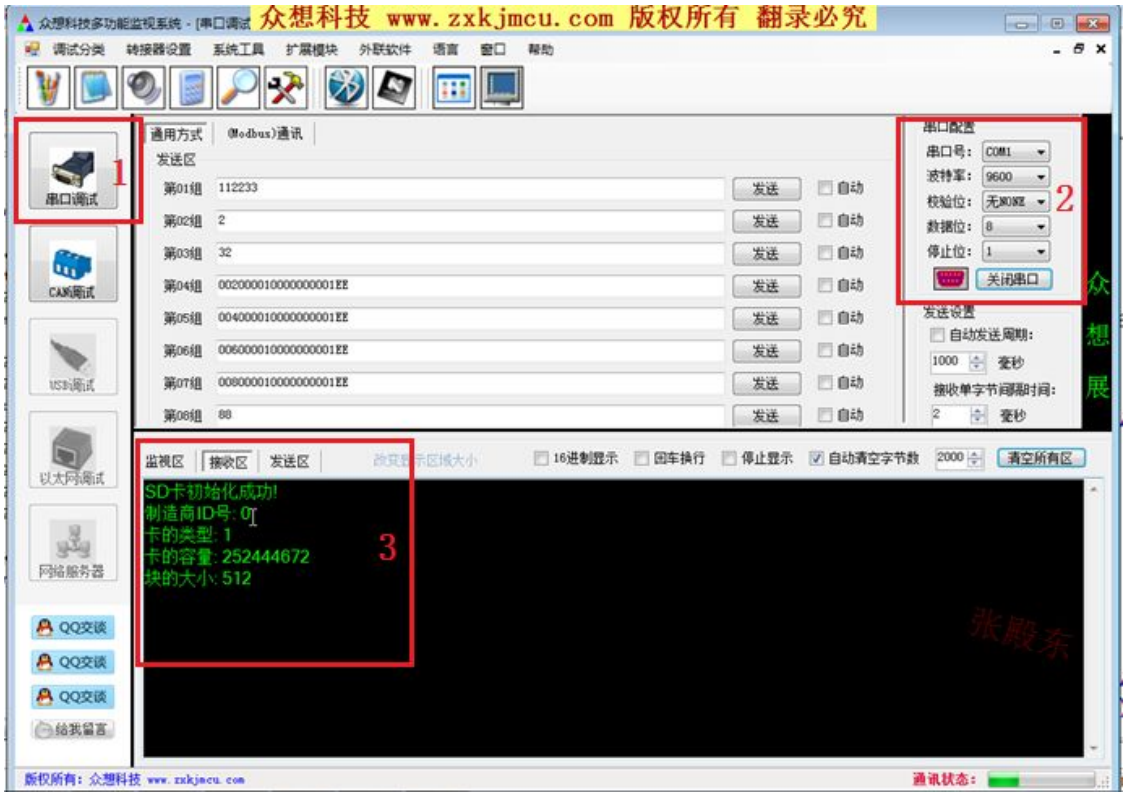


图 4.35.21 SD 卡初始化实验效果图

附件 1：详细命令描述

表 1 基本命令（类 0 和类 1）

命令索引	类型	参数	响应	缩写	描述
CMD0	bc	[31:0]无用	—	GO_IDLE_STATE	重置所有卡到 Idle 状态
CMD1	保留				
CMD2	bcr	[31:0]无用	R2	ALL_SEND_CID	要求所有卡发送 CID 号
CMD3	Bcr	[31:0]无用	R6	SEND_RELATIVE_ADDR	要求所有卡发布一个新的相对地址 RCA
CMD4	不支持				
CMD5	保留				
CMD6	保留				
CMD7	ac	[31:16]RCA	R1	选中/不选中卡	

		[15:0]无用			
CMD8			R7	发送接口状态命令	
CMD9	ac	[31:16] RCA [15:0]无用	R2	SEND_CSD	寻址卡并让其发送卡定义数据 CSD
CMD10	ac	[31:16] RCA [15:0]无用	R2	SEND_CID	寻址卡并让其发送卡识别号 CID
CMD11	adtc	[31:0]数据地址	R1	READ_DAT_UNTIL_STOP	从卡读取数据流，从给定地址开始，知道停止传输命令结束
CMD12	ac	[31:0]无用	R1b	STOP	中止多个块的读/写操作
CMD13	ac	[31:16] RCA [15:0]无用	R1	SEND_STATUS	寻址卡并发送卡状态寄存器
CMD14	保留				
CMD15	ac	[31:16] RCA [15:0]无用	_	GO_INACTIVE_STATE	设置卡到 inactive 状态

表 2 块读操作命令（类 2）

命令索引	类型	参数	响应	缩写	描述
CMD16	ac	[31:0]块长度	R1	SET_BLOCKLEN	为接下来的块操作指令设置块长度
CMD17	adtc	[31:0]数据地址	R1	READ_SINGLE_BLOCK	读取一个块
CMD18	adtc	[31:0]数据地址	R1	READ_MULTIPLE_BLOCK	连续读取多个块，直到停止命令
CMD19-23	保留				

命令索引	类型	参数	响应	缩写	描述
CMD24	adtc	[31:0]数据地址	R1	WRITE_BLOCK	写一个长度由SET_BLOCKLEN 指定的块
CMD25	adtc	[31:0]数据地址	R1	WRITE_ MULTIPLE _BLOCK	连续写多个块直到STOP_TRANSMISSION 命令
CMD26	不支持				
CMD27	adtc	[31:0]无用	R1	PROGRAM_CSD	编辑 CSD 位

表 3 块写操作命令（类 4）

表 4 写保护（类 6）

命令索引	类型	参数	响应	缩写	描述
CMD28	ac	[31:0]数据地址	R1b	SET_WRITE_PROT	设置地址组保护位。写保护由卡配置数据的WP_GRP_SIZE 指定
CMD29	ac	[31:0]数据地址	R1b	CLR_WRITE_PROT	清除保护位
CMD30	adtc	[31:0]写保护数据地址	R1	SEND_WRITE_PROT	要求卡发送写保护位状态
CMD31	保留				

表 5 擦除命令（类 5）

命令索引	类型	参数	响应	缩写	描述
CMD32	ac	[31:0]	R1	ERASE_WR_BLK_START	设置要擦除的第一个写

		数据地址			数据块地址
CMD33	ac	[31:0] 数据地址	R1	ERASE_WR_BLK_END	设置要擦除的最后一个写数据块地址
CMD34 … CMD37	保留				
CMD38	ac	[31:0] 无用	R1b	ERASE	擦除所有选中的写数据块
CMD39 … CMD41	保留				

表 6 擦除命令（类 5）

命令索引	类型	参数	响应	缩写	描述
CMD32	ac	[31:0] 数据地址	R1	ERASE_WR_BLK_START	设置要擦除的第一个写数据块地址
CMD32	ac	[31:0] 数据地址	R1	ERASE_WR_BLK_END	设置要擦除的最后一个写数据块地址
CMD34 … CMD37	保留				
CMD38	ac	[31:0] 无用	R1b	ERASE	擦除所有选中的写数据块
CMD39 … CMD41	保留				

表 7 卡锁命令（类 7）

命令索引	类型	参数	响应	缩写	描述
CMD42 …. CMD54	SDA 可选命令				

表 8 应用相关（Application Specific）命令（类 8）

命令索引	类型	参数	响应	缩写	描述
CMD55	ac	[31:16] RCA [15:0]填充位	R1	APP_CMD	告诉卡接下来的命令是应用相关命令，而非标准命令。
CMD56	adtc	[31:1] 填充位 [0]:RD/WR，1 读，0 写	R1	GEN_CMD	应用相关(通用目的)的数据块读写命令
CMD57 … CMD59	保留				
CMD60 … CMD63	厂商保留				

*命令相关命令，可能指 SD 卡专用命令

所有应用相关命令之前必须先执行 APP_CMD(CMD55)。

表 9 SD 卡使用/保留的应用相关命令

ACMD 索引	类型	参数	响应	缩写	描述

ACMD6	ac	[31:2] 填充位 [1:0]总线宽度	R1	SET_BUS_WIDTH	00:1bit 10:4bit
ACMD13	adtc	[31:0] 填充位	R1	SD_STATUS	设置 SD 卡状态
ACMD17	保留				
ACMD18	—	—	—	—	保留作为 SD 安全应用
ACMD19 … ACMD21	保留				
ACMD22	adtc	[31:0]填充位	R1	SEND_NUM_WR_BLOCKS	发送写数据块的数目。响应为 32 位+CRC
ACMD23	ac	[31:23] 填 充位 [22:0]数据块数目	R1	SET_WR_BLK_ERASE_COUNT	设置写前预擦除的数据块数目(用来加速多数据块写操作)。“1”=默认(一个块)(1)
ACMD24	保留				
ACMD25	—	—	—	—	保留作为 SD 安全应用
ACMD26	—	—	—	—	保留作为 SD 安全应用
ACMD38	—	—	—	—	保留作为 SD 安全应用
ACMD39 … ACMD40	保留				
ACMD41	bcr	[31:0]OCR without busy	R3	SD_APP_OP_COND	要求访问的卡发送它的操作条件寄存器(OCR)内容
ACMD42	ac	[31:1]填充位 [0]set_cd	R1	SET_CLR_CARD_DETECT	连接[1]/断开[0]卡上 CD/DAT3(pin 1]的 50K 欧姆上拉电阻。上拉电阻可用来检测卡

ACMD43 ACMD49	—	—	—	—	保留作为 SD 安全应用
ACMD51	adtc	[31:0]填充位	R1	SEND_SCR	读取 SD 配置寄存器 SCR

(1)不管是否使用 ACMD23，在多数据块写操作中都需要 STOP_TRAN(CMD12)命令