目 录

第1章 ZLG/SD 软件包使用手册

SD/MMC 卡是一种大容量（最大可达 4GB）、性价比高、体积小、访问接口简单的存储卡。SD/MMC 卡大量应用于数码相机、MP3 机、手机、大容量存储设备，做为这些便携式设备的存储载体，它还具有低功耗、非易失性、保存数据无需消耗能量等特点。

SD 卡接口向下兼容 MMC（MutliMediaCard 多媒体卡）卡，访问 SD 卡的 SPI 协议及部分命令也适用于 MMC 卡。

ZLG/SD 软件包是 ZLG 系列软件包的重要成员之一。该软件包可用于来访问 SD/MMC 卡，目前最新版本为 2.00，本版本不仅能读写 SD 卡，还可以读写 MMC 卡；不仅能在前后台系统（无实时操作系统）中使用，还可以在嵌入式操作系统 μCOS-II 中使用。本软件包只支持 SD/MMC 卡的 SPI 模式。

在本章中，除了特别说明以外，“ 卡 ”都是指 SD 卡或 MMC 卡。

1.1 SD/MMC 卡的外部物理接口

SD 和 MMC 卡的外形和接口触点如图 1 所示。其中 SD 卡的外形尺寸为：24mm x 32mm x 2.1mm（普通）或 24mm x 32mm x 1.4mm（薄 SD 存储卡），MMC 卡的外形尺寸为 24mm x 32mm x 1.4mm。

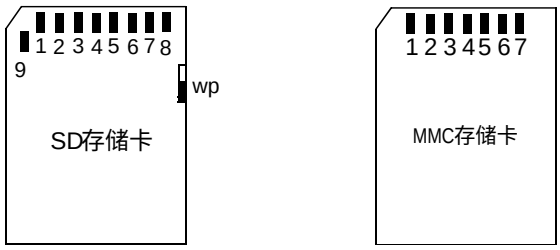

图 1 SD 卡和 MMC 卡的形状和接口（上视图）

表 1 为 SD/MMC 卡各触点的名称及作用，其中 MMC 卡只使用了 1 ~ 7 触点。

表 1 SD/MMC 卡的焊盘分配

引脚	SD 模式			SPI 模式		
	名称[1]	类型	描述	名称	类型	描述
1	CD/DAT3[2]	I/O/PP[3]	卡的检测/数据线[Bit 3]	CS	I	片选（低电平有效）
2	CMD	PP[4]	命令／响应	DI	I[5]	数据输入
3	V_{SS1}	S	电源地	VSS	S	电源地
4	V_{DD}	S	电源	VDD	S	电源
5	CLK	I	时钟	SCLK	I	时钟
6	V_{SS2}	S	电源地	VSS2	S	电源地
7	DAT0	I/O/PP	数据线[Bit 0]	DO	O/PP	数据输出
8	DAT1	I/O/PP	数据线[Bit 1]	RSV		
9	DAT2	I/O/PP	数据线[Bit 2]	RSV		

注：1. S：电源；I：输入；O：推挽输出；PP：推挽 I/O。

2. 扩展的 DAT 线（DAT1 ~ DAT3）在上电后处于输入状态。它们在执行 SET_BUS_WIDTH 命令后作为 DAT 线操作。当不使用 DAT1 ~ DAT3 线时，主机应使自己的 DAT1~DAT3 线处于输入模式。

这样定义是为了与 MMC 卡保持兼容。

3. 上电后，这条线为带 50KΩ上拉电阻的输入线（可以用于检测卡是否存在或选择 SPI 模式）。用户可以在正常的数据传输中用 SET_CLR_CARD_DETECT（ACMD42）命令断开上拉电阻的连接。MMC 卡的该引脚在 SD 模式下为保留引脚，在 SD 模式下无任何作用。
4. MMC 卡在 SD 模式下为：I/O/PP/OD。
5. MMC 卡在 SPI 模式下为：I/PP。

由表 1 可见，SD 卡和 MMC 卡在不同的通信模式下，各引脚的功能也不相同。这里的通信模式是指微控制器（主机）访问卡时使用的通信协议，分为两种：SD 模式及 SPI 模式。

在具体通信过程中，主机只能选择其中一种通信模式。通信模式的选择对于主机来说是透明的。卡将会自动检测复位命令的模式（即自动检测复位命令使用的协议），而且要求以后双方的通信都按相同的通信模式进行。所以，在只使用一种通信模式的时候，无需明白另一种模式。下面先简单介绍这两种模式。

1.1.1 SD 模式

在 SD 模式下，主机使用 SD 总线访问 SD 卡，其拓朴结构如图 2 所示。由图可见，SD 总线上不仅可以挂接 SD 卡，还可以挂接 MMC 卡。

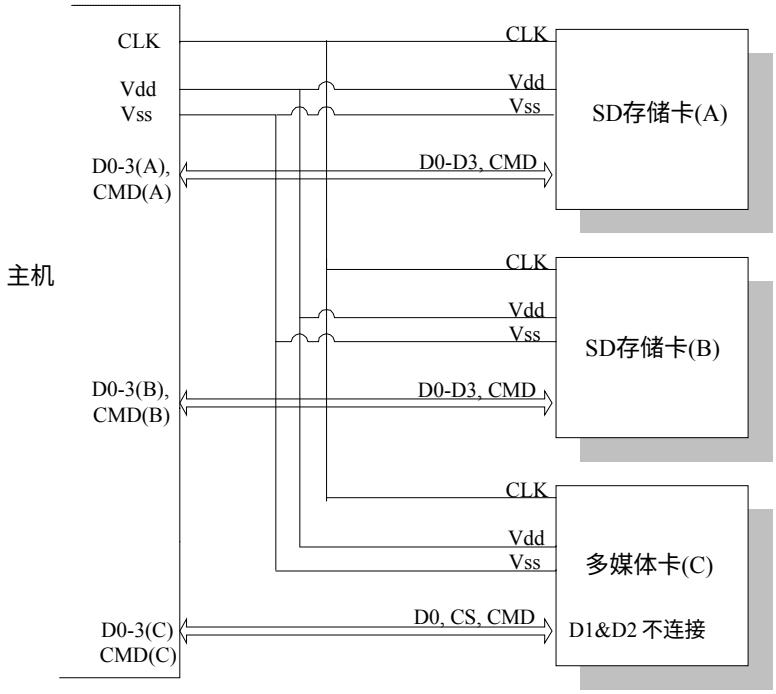

图 2 SD 存储卡系统（SD 模式）的总线拓扑结构

SD 总线上的信号线的详细功能描述如表 2 所示。

表 2 SD 总线信号线功能描述

信号线	功能描述
CLK	主机向卡发送的用于同步双方通信的时钟信号
CMD	双向的命令／响应信号
DAT0 ~ DAT3	4 个双向的数据信号（MMC 卡只有 DAT0 信号线）

续上表

信号线	功能描述
VDD	电源正极，一般电压范围为 2.7～3.6V
VSS1、VSS2	电源地

SD 存储卡系统（SD 模式）的总线拓扑结构为: 一个主机（如微控制器）、多个从机（卡）和同步的星形拓扑结构（参考图 2）。所有卡共用时钟 CLK、电源和地信号。而命令线（CMD）和数据线（DAT0 ～ DAT3）则是卡的专用线，即每张卡都独立拥有这些信号线。请注意，MMC 卡只能使用 1 条数据线 DAT0。

1.1.2 SPI 模式

在 SPI 模式下，主机使用 SPI 总线访问卡，当今大部分微控制器本身都带有硬件 SPI 接口，所以使用微控制器的 SPI 接口访问卡是很方便的。微控制器在卡上电后的第 1 个复位命令就可以选择卡进入 SPI 模式或 SD 模式，但在卡上电期间，它们之间的通信模式不能更改为 SD 模式。

卡的 SPI 接口与大多数微控制器的 SPI 接口兼容。卡的 SPI 总线的信号线如表 3 所示。

表 3 SD 卡与 MMC 卡的 SPI 接口描述

信号线	功能描述
CS	主机向卡发送的片选信号
CLK	主机向卡发送的时钟信号
DataIn	主机向卡发送的单向数据信号
DataOut:	卡向主机发送的单向数据信号

SPI 总线以字节为单位进行数据传输，所有数据令牌都是字节（8 位）的倍数，而且字节通常与 CS 信号对齐。SD 卡存储卡系统如图 3 所示。

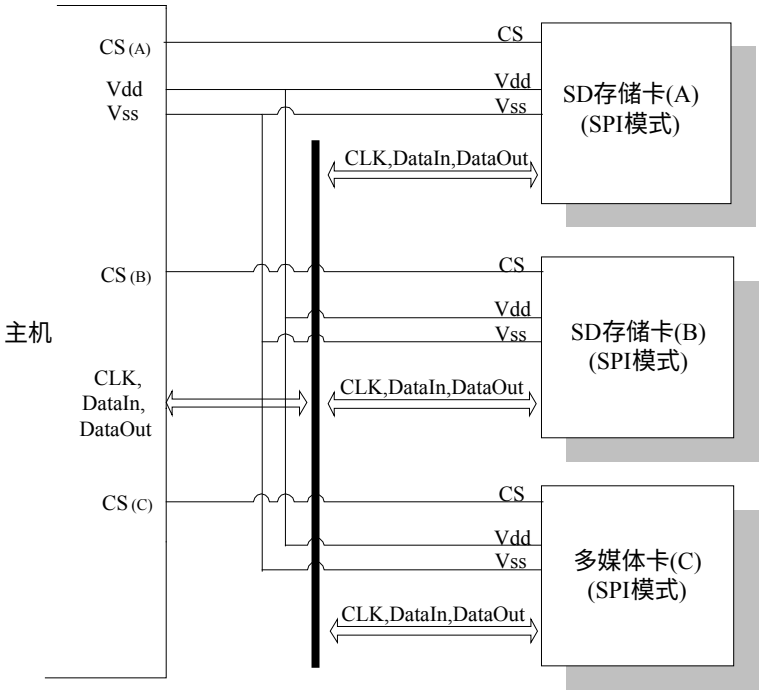

图 3 SD 存储卡系统（SPI 模式）的总线拓扑结构

当主机外部连接有多张 SD 卡或 MMC 卡时，主机利用 CS 信号线对卡进行寻址。例如：在图 3 中，当主机需要向 SD 存储卡（A）传输数据或需要从该卡接收数据时，必须将 $CS_{(A)}$ 置为低电平（同时其它卡的 CS 信号线必须置为高电平）。

CS 信号在 SPI 处理（命令、响应和数据）期间必须续持有效（低电平）。唯一例外的情况是在对卡编程的过程。在这个过程中，主机可以使 CS 信号为高电平，但不影响卡的编程。

由图 3 还可见，当 SPI 总线上挂接 N 张卡时，需要 N 条 CS 片选线。

1.2 访问 SD/MMC 卡的 SPI 模式硬件电路设计

SD/MMC 卡可以采用 SD 总线访问，也可以采用 SPI 总线访问，考虑到大部分微控制器都有 SPI 接口而没有 SD 总线接口，而且如果采用 I/O 口模拟 SD 总线，不但增加了软件的开销，而且对大多数微控制器而言，模拟总线远不如真正的 SD 总线速度快，这将大大降低总线数据传输的速度。

基于以上的考虑，采用 LPC2210 微控制器的 SPI 接口为例子，设计访问 SD/MMC 卡的硬件接口电路。LPC2210 微控制器与 SD/MMC 卡卡座接口电路如图 4 所示。

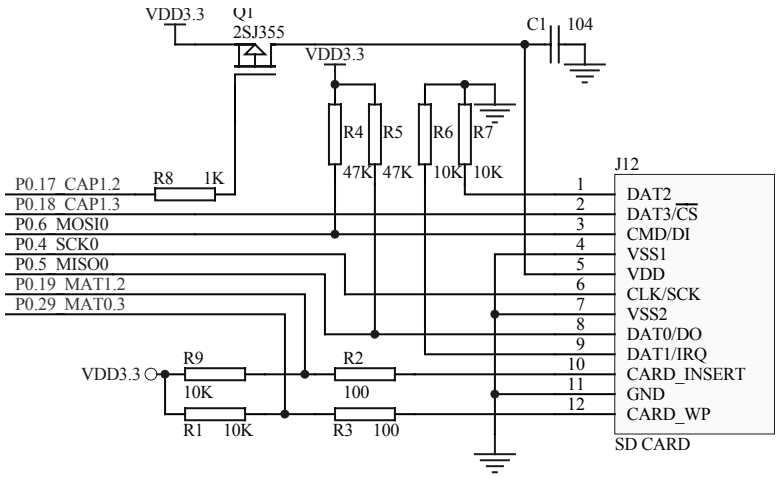

图 4 SD 卡卡座与 LPC2210 接口电路（SPI 模式）

图中，LPC2210 与 SD/MMC 卡卡座的连接引脚如表 4 所示。

表 4 LPC2210 与 SD/MMC 卡卡座的连接引脚

LPC2210 接口	含 义
P0.18_CAP1.3	SPI 片选信号，用于选择 SPI 从机，该引脚为普通 I/O 口
P0.4_SCK0	SPI 时钟信号，由主机发出，用于同步主机之间的数据传输
P0.6_MOSI0	SPI 主机输出，从机输入信号
P0.5_MISO0	SPI 主机输入，从机输出信号
P0.17_CAP1.2	卡供电控制，当该引脚输出低电平时给卡供电
P0.19_MAT1.2	卡完全插入到卡座中检测线，完全插入时卡座输出低电平，否则输出高电平
P0.29_MAT0.3	卡是否写保护检测，写保护时卡座输出高电平，否则输出低电平

注意：对于不同的卡座，卡完全插入检测电平和写保护检测电平可能有所不同。

接口电路包括以下几个部分。

1.2.1 SPI 总线

如图 4 所示，LPC2210 SPI 接口的 P0.18_CAP1.3、P0.4_SCK0、P0.6_MOSI0、P0.5_MISO0 直接连接到卡座的相应接口，其中 SPI 的两个数据线 P0.6_MOSI0、P0.5_MISO0 还分别接上拉电阻，这是为了使本电路可以与 MMC 卡的接口兼容。SPI 模式下无需用到的信号线 DAT2 和 DATA1 分别接下拉电阻。

1.2.2 卡供电控制

卡的供电采用可控方式，这是为了防止 SD/MMC 卡进入不确定状态时，可以通过对卡重新上电使卡复位而无需拔出卡。

可控电路采用 P 型 MOS 管 2SJ355，由 LPC2210 的 GPIO 口 P0.17_CAP1.2 进行控制，当 P0.17_CAP1.2 输出高电平时，2SJ355 关断，不给卡供电；当 P0.17_CAP1.2 输出低电平时，2SJ355 开通，VDD3.3 电源（电压为 3.3V）给卡供电。

采用 2SJ355 的目的是当它开通时，管子上的压降比较小。2SJ355 的相关特性请见其数据手册。用户也可以采用其它 P 型的 MOS 管，但是要考虑管子开通时，漏极与源极之间的压降要足够小（保证 SD/MMC 卡的工作电压在允许范围内），管子允许通过的电流也要满足卡的要求，一般一张 SD/MMC 卡工作时的最大电流通常为 45mA 左右，所以选用的 MOS 管要求允许通过 100mA 左右的电流。

1.2.3 卡检测电路

卡检测电路包括两部分：卡是否完全插入到卡座中和卡是否写保护。

检测信号由卡座的两个引脚以电平的方式输出。当卡插入到卡座并插入到位时，P0.19_MAT1.2（第 10 脚）由于卡座内部触点连接到 GND，输出低电平；当卡拔出时，该引脚由于上拉电阻 R2 的存在而输出高电平，该输出由 LPC2210 的输入引脚 GPIO（P0.19_MAT1.2）来检测。

卡是否写保护的检测与卡是否完全插入到卡座中的检测原理是一样的。

1.3 ZLG/SD 软件包的文件结构及整体构架

本小节介绍 ZLG/SD 软件包的组成文件以及它们之间的联系。

1.3.1 ZLG/SD 软件包的文件组成

ZLG/SD 软件包包括的文件如表 5 所示。

表 5 ZLG/SD 软件包包含的文件

文 件	作 用
sdconfig.h	软件包硬件配置头文件
sdhal.c	软件包硬件抽象层，实现 SPI 接口初始化，SPI 字节的收、发等与 SPI 硬件相关的函数
sdhal.h	sdhal.c 头文件
sdcmd.c	软件包命令层，实现卡的各种命令以及主机与卡之间的数据流控制
sdcmd.h	sdcmd.c 头文件
sddriver.c	软件包应用层，实现卡的读、写、擦 API 函数，该文件还包含一些卡操作函数
sddriver.h	sddriver.c 头文件，包括函数执行错误代码
sdcrc.c	卡相关的 CRC 运算函数。
sdcrc.h	sdcrc.h 头文件

以上这些文件构成了本软件包，下面说明由这些文件构成的整体框架。

1.3.2 ZLG/SD 软件包整体框架

考虑到该软件包的可移植性及易用性，将软件包分为 3 个层，如图 5 所示。图中的实时操作系统并不是必须的，也就是说，本软件包既可以应用于前后台系统（无实时操作系统），也可以应用于实时操作系统中，本软件包提供在前后台系统和实时操作系统 μCOS-II 中接口统一的 API 函数。

是否使用实时操作系统由本软件包 sdconfig.h 文件中的宏定义 SD_UCOSII_EN 来使能或禁止。

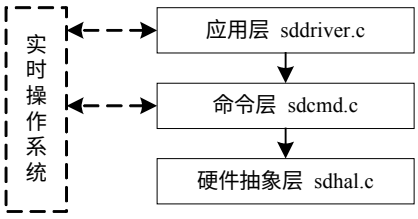

图 5 ZLG/SD 软件包结构图

各层的特点如下：

（1）硬件抽象层：读写 SD/MMC 卡的硬件条件配置，与硬件相关的函数；

（2）命令层：SD/MMC 卡的相关命令以及卡与主机之间数据流的控制，这一层与实时操作系统相关，与硬件无关。

（3）应用层：向用户应用程序或文件系统提供操作卡的 API 函数，这一层由实时操作系统控制。

1.4 ZLG/SD 软件包的使用说明

使用本软件包之前，必须配置好本软件包使用的硬件条件，如果你的硬件条件与 1.2 小节中的硬件条件一样，那么无须配置就可以立即使用。下面说明怎样配置本软件包的硬件条件，才能将其用于 LPC2210 系列微控制器。

1.4.1 ZLG/SD 软件包的硬件配置

ZLG/SD 软件包在 LPC2210 的配置只与 sdconfig.h 文件相关，配置头文件 sdconfig.h 使用户能方便地配置本软件包的相关功能及裁剪某些对用户来说无需用到的函数。该小节提到的所有程序清单都在该文件上。下面阐述该头文件的配置方法。

1． 软件包参数配置

软件包的参数配置如程序清单 1 所示，配置选项如下：

（1）是否运行于 μCOS-II 中。本软件包既可以运行于前后台系统中，又可以运行于实时操作系统 μCOS-II 中。当运行于 μCOS-II 中时，宏定义 SD_UCOSII_EN 的值应置为 1，否则应置为 0。

（2）CRC 校验。由于 SD/MMC 卡在 SPI 通信模式下可以不需要进行数据传输的 CRC 校验，该宏用于使能或禁止本软件包的数据传输 CRC 校验功能。使能 CRC 校验则通信可靠性更高，但 CRC 运算也带来传输速度的一些损失，由于本软件包采用查表的方法计算 CRC16，所以速度只是略有损失。

（3）SPI 时钟频率。定义 SPI 总线的 CLK 线的频率，该频率值用于计算读、写、擦操

作中的超时时间对应的 CLK 个数，这样就将超时时间转换为超时计数。该频率值的单位为：Hz，该值需要用户定义，定义的方法见下面介绍（4．设置 SPI 接口的时钟频率小于 400kHz）。

（4）SD/MMC 卡块的长度。定义 SD/MMC 卡块的最大长度，当今流行的 SD/MMC 卡块的最大长度大部分都支持 512 字节。宏定义 SD_BLOCKSIZE_NBITS 值为 9，对应于 2^9 = 512 字节（对应于宏定义 SD_BLOCKSIZE 的值），SD_BLOCKSIZE_NBITS 与 SD_BLOCKSIZE 一定要有这样的对应关系。SD_BLOCKSIZE_NBITS 参数用于固件程序数据计算的方便。用户一般无须改动这两个宏定义的值。

程序清单 1 软件包参数配置

```
#define   SD_UCOSII_EN          1          /*  是否在 UCOS-II 上运行本软件包  */
#define   SD_CRC_EN             0          /*  设置数据传输时是否使用 CRC */
#define   SPI_CLOCK             5529600    /*  正常通信时,SPI 时钟频率(Hz) */
#define   SD_BLOCKSIZE          512        /* SD/MMC 卡块的长度  */
#define   SD_BLOCKSIZE_NBITS    9          /* 2 的 9 次方为 512（即 SD_BLOCKSIZE 的值）*/
```

2． 功能配置

软件包中有一些功能不是所有用户都可能用到的，所以可以裁剪去不需要的函数，以减小其代码量。程序清单 2 的宏定义用于使能编译软件包中的某些比较少用的函数，当取值为 1 时，使能编译对应的函数；为 0 时，禁止编译对应的函数。这些宏定义起到裁剪软件包代码大小的目的。

程序清单 2 软件包函数使能

```
/*  下面函数不常用,如果用户不需要,可置为  0  裁剪指定函数  */
#define   SD_ReadMultiBlock_EN      0        /*  是否使能读多块函数  */
#define   SD_WriteMultiBlock_EN     0        /*  是否使能写多块函数  */
#define   SD_EraseBlock_EN          0        /*  是否使能擦卡函数  */
#define   SD_ProgramCSD_EN          0        /*  是否使能写 CSD 寄存器函数  */
#define   SD_ReadCID_EN             0        /*  是否使能读 CID 寄存器函数  */
#define   SD_ReadSD_Status_EN       0        /*  是否使能读 SD Status 寄存器函数  */
#define   SD_ReadSCR_EN             0        /*  是否使能读 SCR 寄存器函数  */
```

3． 硬件条件配置

这部分以宏的形式对卡的 SPI 口和 IO 口相关操作进行定义，尽可能地将硬件相关的部分放于这一部分。例如，配置 LPC2210 的 4 个 IO 口为 SPI 接口的宏定义如程序清单 3（1）所示。对于 SD 卡卡座供电引脚的控制如程序清单 3（2）所示。用户阅读 LPC2210 的数据手册就可以了解这些配置的含义。该文件还包括对其它 IO 口的配置，程序清单 3 都有详细的注释。

程序清单 3 LPC2210 的 IO 口配置宏定义

```
/* SCK 引脚  */
#define   SPI_SCK               (0x01 << 4)
#define   SPI_SCK_GPIO()        PINSEL0 &= ~(0x03 << 8)     /*  设置  SCK  口为 GPIO 口  */
#define   SPI_SCK_OUT()         IO0DIR |= SPI_SCK           /*  设置  SCK  口为输出口  */
```

```
#define   SPI_SCK_CLR()           IO0CLR = SPI_SCK            /* 置 SCK 为低电平 */

/* MISO 引脚 */
#define   SPI_MISO                (0x01 << 5)
#define   SPI_MISO_GPIO()         PINSEL0 &= ~(0x03 << 10)    /* 设置 MISO 口为 GPIO 口 */
#define   SPI_MISO_OUT()          IO0DIR |= SPI_MISO          /* 设置 MISO 口为输出口 */
#define   SPI_MISO_CLR()          IO0CLR = SPI_MISO           /* 置 MISO 为低电平 */

/* MOSI 引脚 */
#define   SPI_MOSI                (0x01 << 6)
#define   SPI_MOSI_GPIO()         PINSEL0 &= ~(0x03 << 12)    /* 设置 MOSI 口为 GPIO 口 */
#define   SPI_MOSI_OUT()          IO0DIR |= SPI_MOSI          /* 设置 MOSI 口为输出口 */
#define   SPI_MOSI_CLR()          IO0CLR = SPI_MOSI           /* 置 MISO 为低电平 */

/* CS 引脚 */
#define   SPI_CS                  (0x01 << 18)
#define   SPI_CS_GPIO()           PINSEL1 &= ~(0x03 << 4)      /* 设置 CS 口为 GPIO 口 */
#define   SPI_CS_OUT()            IO0DIR |= SPI_CS;            /* 设置 CS 口为输出口 */
#define   SPI_CS_SET()            IO0SET |= SPI_CS;            /* 置 CS 为高电平 */
#define   SPI_CS_CLR()            IO0CLR |= SPI_CS;            /* 置 CS 为低电平 */

/* 初始化 IO 口为 SPI 接口 */
#define   SPI_INIT()        PINSEL0 &= ~((0x03 << 8) + (0x03 << 10) + (0x03 << 12) + (0x03 << 14)); \
                            PINSEL0 |= (0x01 << 8) + (0x01 << 10) + (0x01 << 12) + (0x01 << 14);          (1)

/* 电源控制引脚 */
#define   SD_POWER                (0x01 << 17)                                                          (2)
#define   SD_POWER_GPIO()         PINSEL1 &= ~(0x03 << 2)     /* 设置 POWER 口为 GPIO 口 */
#define   SD_POWER_OUT()          IO0DIR |= SD_POWER          /* 设置 POWER 口为输出口 */
#define   SD_POWER_OFF()          IO0SET = SD_POWER           /* 置 POWER 为高电平 */
#define   SD_POWER_ON()           IO0CLR = SD_POWER           /* 置 POWER 为低电平 */

/* 卡完全插入卡座检测引脚 */
#define   SD_INSERT               (0x01 << 19)
#define   SD_INSERT_GPIO()        PINSEL1 &= ~(0x03 << 6)     /* 设置 INSERT 口为 GPIO 口 */
#define   SD_INSERT_IN()          IO0DIR &= ~SD_INSERT        /* 设置 INSERT 口为输出口 */
#define   SD_INSERT_STATUS()      (IO0PIN & SD_INSERT)        /* 读取 INSERT 口的状态 */

/* 卡写保护检测引脚 */
#define   SD_WP                   (0x01 << 29)
#define   SD_WP_GPIO()            PINSEL1 &= ~(0x03 << 26)    /* 设置 WP 口为 GPIO 口 */
#define   SD_WP_IN()              IO0DIR &= ~SD_WP            /* 设置 WP 口为输出口 */
#define   SD_WP_STATUS()          (IO0PIN & SD_WP)            /* 读取 WP 口的状态 */
```

如果你的硬件条件变了，例如，如果需要将卡完全插入卡座检测引脚改用 P0.25，那么该 IO 口的配置要做以下改动：(0x01 << 19)改为(0x01 << 25)，(0x03 << 6)改为(0x03 << 18)，修改后的结果如下：

```
/* 卡完全插入卡座检测引脚 */
#define   SD_INSERT                 (0x01 << 25)
#define   SD_INSERT_GPIO()          PINSEL1 &= ~(0x03 << 18)    /* 设置 INSERT 口为 GPIO 口 */
#define   SD_INSERT_IN()            IO0DIR &= ~SD_INSERT        /* 设置 INSERT 口为输出口 */
#define   SD_INSERT_STATUS()        (IO0PIN & SD_INSERT)        /* 读取 INSERT 口的状态 */
```

配置头文件的全部内容就介绍到此，如果你想要将本软件包移植到其它 MCU，则还需修改 sdhal.c 文件，这一部分与 MCU 的 SPI 接口操作相关，如果你使用的是 LPC2200/LPC2100，那么无须改动该文件。

还有一点容易忽略的，就是必须将 LPC2210 的外设时钟频率 Fpclk 调至最高（在允许范围内），这样，软件包的读定速度才能达到最高。而且 SD/MMC 卡的 SPI 总线协议中，要求 SPI 主机必须能够控制 SPI 总线的时钟频率。LPC2210 对 SPI 总线时钟的控制函数说明如下。

4． 设置 SPI 接口的时钟频率小于 400kHz

该函数主要是在 SD/MMC 卡初始化阶段，用于设置 SPI 接口的时钟频率小于 400kHz，因为 MMC 卡在初始化期间 SPI 总线的时钟频率不能高于 400kHz，这样本软件包才能达到兼容 MMC 卡的目的。该函数如程序清单 4 所示（见 sdhal.c 文件）。该函数只是修改 LPC2210 SPI 接口的 SPI 时钟计数寄存器 SPI_SPCCR 的分频值来达到目的。

程序清单 4 设置 SPI 接口的时钟频率小于 400kHz

```
void SPI_Clk400k(void)
{
    SPI_SPCCR = 128;                            /* 设置 SPI 时钟分频值为 128 */
}
```

如果 LPC2210 的外部晶振频率 Fosc = 11.0592MHz，内核时钟频率 Fcclk 设置为 Fosc 的 4 倍，即 Fcclk = 44.2368 MHz。外设时钟频率设置为与内核时钟频率相同，即 Fpclk = Fcclk = 44.2368 MHz，那么 SPI 总线的时钟为 Fpclk 经过 SPI_SPCCR 寄存器分频后的时钟。所以，要使 SPI 接口的时钟频率少于 400kHz，同时又要保证 SPI_SPCCR 寄存器的值为大于 8 的偶数，该寄存器的分频值要设为 128。

得到的 SPI 接口 SCK 的频率为：

44.2368 / 128 = 0.3456 MHz = 345.6kHz < 400kHz。

同样，如果要设置 SCK 的频率为最大值，只须调用 void SPI_ClkToMax(void)函数（见 sdhal.c 文件），该函数只是将 SPI_SPCCR 的值改为 8，那么得到 SCK 的频率为：

44.2368 / 8 = 5.5296MHz。

这个值正是 sdconfig.h 文件中宏定义 SPI_CLOCK 的值：

```
#define   SPI_CLOCK                 5529600         /* 正常通信时,SPI 时钟频率(Hz) */
```

需要注意，当今流行的 SD/MMC 卡的 SPI 接口的时钟频率一般不允许超过 25MHz，所以在定义 MCU 访问 SD/MMC 卡的时钟频率时，必须注意这一点。

如果读者的外部晶振频率 Fosc 或 LPC2210 外设时钟频率 Fpclk 改变了，请注意修改这两个函数中的分频值，来优化本软件包对卡的访问速度。

配置好了硬件，那么可以使用本软件包了，那么本软件包提供了哪些 API 函数方便用户访问 SD/MMC 卡？下面介绍这些 API 函数的接口。

1.4.2 ZLG/SD 软件包提供的 API 函数

用户可以利用本软件包提供的 API 函数对 SD/MMC 卡进行访问，见表 6 至表 11。

表 6 SD_Initialize()

函数名称	SD_Initialize
函数原型	INT8U SD_Initialize(void)
功能描述	初始化 SD/MMC 卡、设置块大小为 512 字节，获取卡的相关信息
函数参数	无
函数返回值	SD_NO_ERR：初始化成功； > 0: 初始化失败（错误码，见表 12）
特殊说明和注意点	该函数设置了卡的读/写块长度为 512 字节

表 7 SD_ReadBlock ()

函数名称	SD_ReadBlock
函数原型	INT8U SD_ReadBlock(INT32U blockaddr, INT8U *recbuf)
功能描述	读 SD/MMC 卡的一个块
函数参数	blockaddr：以块为单位的块地址。例如，卡开始的 0 ~ 511 字节为块地址 0，512 ~ 1023 字节的块地址为 1 recbuf： 接收缓冲区，长度固定为 512 字节
函数返回值	SD_NO_ERR：读成功； > 0: 读失败（错误码，见表 12）
特殊说明和注意点	recbuf 的长度必须是 512 字节

表 8 SD_WriteBlock()

函数名称	SD_WriteBlock
函数原型	INT8U SD_WriteBlock(INT32U blockaddr, INT8U *sendbuf)
功能描述	写 SD/MMC 卡的一个块
函数参数	blockaddr：以块为单位的块地址。例如，卡开始的 0 ~ 511 字节为块地址 0，512 ~ 1023 字节的块地址为 1 sendbuf： 发送缓冲区，长度固定为 512 字节
函数返回值	SD_NO_ERR：写成功； > 0: 写失败（错误码，见表 12）
特殊说明和注意点	sendbuf 的长度必须是 512 字节

表 9 SD_ReadMultiBlock()

函数名称	SD_ReadMultiBlock
函数原型	INT8U SD_ReadMultiBlock(INT32U blockaddr, INT32U blocknum, INT8U *recbuf)

功能描述	读 SD/MMC 卡的多个块
函数参数	blockaddr： 以块为单位的块地址 blocknum： 块数 recbuf： 接收缓冲区，长度为 512 * blocknum 字节
函数返回值	SD_NO_ERR：读成功； > 0: 读失败（错误码，见表 12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_ReadMultiBlock_EN 置为 1

表 10 SD_WriteMultiBlock ()

函数名称	SD_WriteMultiBlock
函数原型	INT8U SD_WriteMultiBlock(INT32U blockaddr, INT32U blocknum, INT8U *sendbuf)
功能描述	读 SD/MMC 卡的多个块
函数参数	blockaddr：以块为单位的块地址 blocknum：块数 sendbuf： 发送缓冲区，长度为 512 * blocknum 字节
函数返回值	SD_NO_ERR：写成功； > 0: 写失败（错误码，见表 12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_WriteMultiBlock_EN 置为 1

表 11 SD_EraseBlock()

函数名称	SD_EraseBlock
函数原型	INT8U SD_EraseBlock(INT32U startaddr, INT32U blocknum)
功能描述	擦除 SD/MMC 卡的多个块
函数参数	startaddr： 以块为单位的块擦除起始地址 blocknum：块数（取值范围 1 ~ sds.block_num）
函数返回值	SD_NO_ERR：擦除成功； > 0: 擦除失败（错误码，见表 12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_EraseBlock_EN 置为 1。Startaddr 和 blocknum 建议为 sds.erase_unit 的整数倍, 因为有的卡只能以 sds.erase_unit 为单位进行擦除

其它函数不常用，这里就不一一列出了。需要用到其它函数的读者可以阅读源码中的函数说明。表 6 至表 11 函数返回值所代表的含义如表 12 所示。

表 12 错误代码列表

错误码宏定义	宏定义值	含义
SD_NO_ERR	0x00	函数执行成功
SD_ERR_NO_CARD	0x01	卡没有完全插入到卡座中
SD_ERR_USER_PARAM	0x02	用户使用 API 函数时，入口参数有错误
SD_ERR_CARD_PARAM	0x03	卡中参数有错误（与本软件包不兼容）
SD_ERR_VOL_NOTSUSP	0x04	卡不支持 3.3V 供电
SD_ERR_OVER_CARDRANGE	0x05	操作超出卡存储器范围
SD_ERR_UNKNOWN_CARD	0x06	无法识别卡型
SD_ERR_CMD_RESPTYPE	0x10	命令类型错误

SD_ERR_CMD_TIMEOUT	0x11	命令响应超时
SD_ERR_CMD_RESP	0x12	命令响应错误
SD_ERR_DATA_CRC16	0x20	数据流 CRC16 校验不通过
SD_ERR_DATA_START_TOK	0x21	读单块或多块时，数据开始令牌不正确
SD_ERR_DATA_RESP	0x22	写单块或多块时，卡数据响应令牌不正确
SD_ERR_TIMEOUT_WAIT	0x30	写或擦操作时，发生超时错误
SD_ERR_TIMEOUT_READ	0x31	读操作超时错误
SD_ERR_TIMEOUT_WRITE	0x32	写操作超时错误
SD_ERR_TIMEOUT_ERASE	0x33	擦除操作超时错误
SD_ERR_TIMEOUT_WAITIDLE	0x34	初始化卡时，等待卡退出空闲状态超时错误
SD_ERR_WRITE_BLK	0x40	写块数据错误
SD_ERR_WRITE_BLKNUMS	0x41	写多块时，想要写入的块与正确写入的块数不一致
SD_ERR_WRITE_PROTECT	0x42	卡外壳的写保护开关打在写保护位置
SD_ERR_CREATE_SEMSD	0xA0	创建访问卡的信号量失败

ZLG/SD 软件包的例子可见：

《ARM 嵌入式系统实验教程（二）》

《ARM 嵌入式系统实验教程（三）》

《深入浅出 ARM7 – LPC213x/LPC214x（下册）》

（全文完）