

## 目 录

|                                    |    |
|------------------------------------|----|
| 第 1 章 SD/MMC 卡读写模块.....            | 1  |
| 1.1 SD/MMC 卡的外部物理接口.....           | 1  |
| 1.1.1 SD 模式.....                   | 2  |
| 1.1.2 SPI 模式.....                  | 3  |
| 1.2 访问 SD/MMC 卡的 SPI 模式硬件电路设计..... | 4  |
| 1.2.1 SPI 总线.....                  | 5  |
| 1.2.2 卡供电控制.....                   | 5  |
| 1.2.3 卡检测电路.....                   | 5  |
| 1.3 SD/MMC 卡读写模块的文件结构及整体构架.....    | 5  |
| 1.3.1 SD/MMC 卡读写模块的文件组成.....       | 5  |
| 1.3.2 SD/MMC 读写模块整体框架.....         | 6  |
| 1.4 SD/MMC 卡读写模块的使用说明.....         | 6  |
| 1.4.1 SD/MMC 卡读写模块的硬件配置.....       | 6  |
| 1.4.2 SD/MMC 卡读写模块提供的 API 函数.....  | 9  |
| 1.5 SD/MMC 卡读写模块的应用示例一.....        | 11 |
| 1.5.1 硬件连接与配置.....                 | 11 |
| 1.5.2 实现方法.....                    | 11 |
| 1.6 SD/MMC 卡读写模块的使用示例二.....        | 18 |
| 1.6.1 实现方法.....                    | 18 |
| 1.6.2 例子建立与运行步骤.....               | 20 |
| 1.6.3 参考程序.....                    | 24 |
| 1.7 SD/MMC 软件包应用总结.....            | 27 |

## 第1章 SD/MMC 卡读写模块

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，作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。

SD/MMC 卡读写模块是 ZLG 系列中间件的重要成员之一，又称为 ZLG/SD。该模块是一个用来访问 SD/MMC 卡的软件读写模块，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  $\mu\text{C}/\text{OS-II}$  中使用。本文模块只支持 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 1.1 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 1.1 SD 卡和 MMC 卡实物图

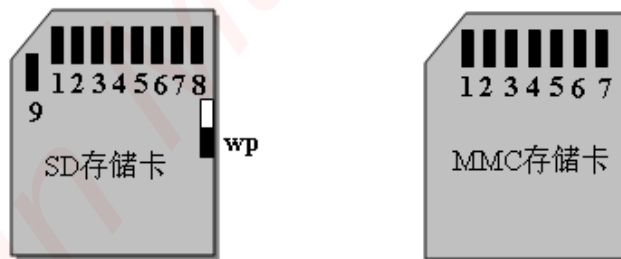


图 1.2 SD 卡和 MMC 卡接口示意图（上视图）

表 1.1 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 1.1 SD/MMC 卡的焊盘分配

| 引脚 | SD 模式                |                     |                 | SPI 模式 |                |           |
|----|----------------------|---------------------|-----------------|--------|----------------|-----------|
|    | 名称 <sup>1</sup>      | 类型                  | 描述              | 名称     | 类型             | 描述        |
| 1  | CD/DAT3 <sup>2</sup> | I/O/PP <sup>3</sup> | 卡的检测/数据线[Bit 3] | CS     | I              | 片选（低电平有效） |
| 2  | CMD                  | PP <sup>4</sup>     | 命令 / 响应         | DI     | I <sup>5</sup> | 数据输入      |
| 3  | V <sub>SS1</sub>     | S                   | 电源地             | VSS    | S              | 电源地       |

续上表

| 引脚 | SD 模式            |        |            | SPI 模式 |      |      |
|----|------------------|--------|------------|--------|------|------|
|    | 名称 <sup>1</sup>  | 类型     | 描述         | 名称     | 类型   | 描述   |
| 4  | V <sub>DD</sub>  | S      | 电源         | VDD    | S    | 电源   |
| 5  | CLK              | I      | 时钟         | SCLK   | I    | 时钟   |
| 6  | V <sub>SS2</sub> | 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 SPI 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 1.1 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, 分别为 SD 模式和 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, 无需使用另一种模式。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, 其总线拓扑结构如图 1.3 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。

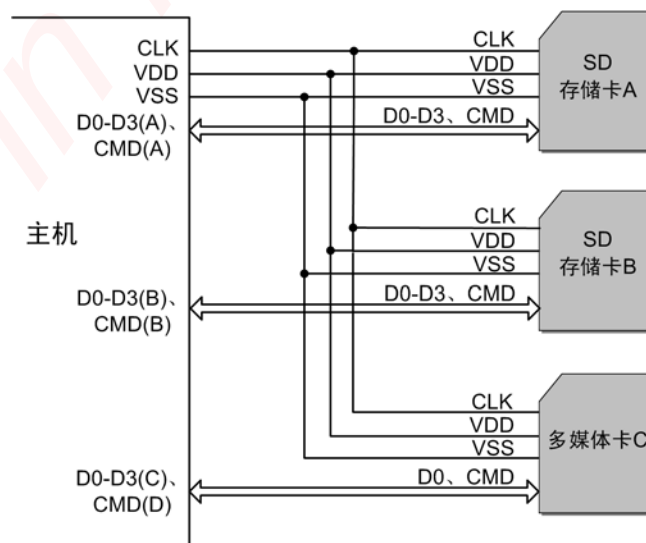


图 1.3 SD 存储卡系统 (SD 模式) 的总线拓扑结构

SD 总线上的信号线的详细功能描述如表 1.2 所示。

表 1.2 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) |
| 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 1.3)。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 1.3 所示。

表 1.3 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 1.4 所示。

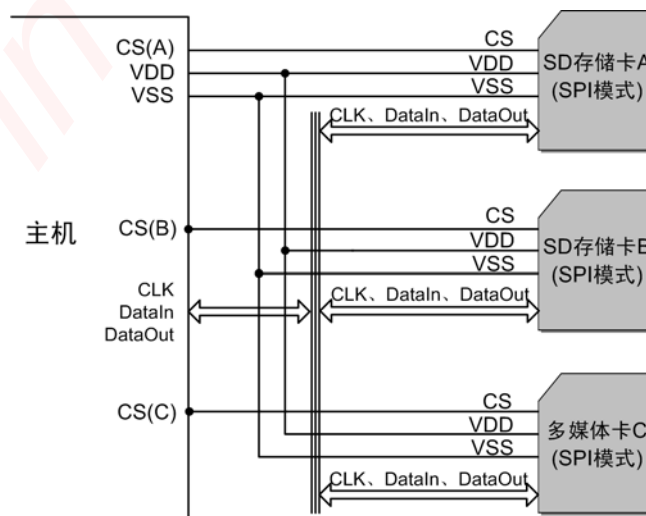


图 1.4 SD 存储卡系统 (SPI 模式) 的总线拓扑结构

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 1.4 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<sub>(A)</sub> 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）期间必须持续有效（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 1.4 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 LPC2103 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。LPC2103 微控制器与 SD/MMC 卡卡座接口电路如图 1.5 所示。

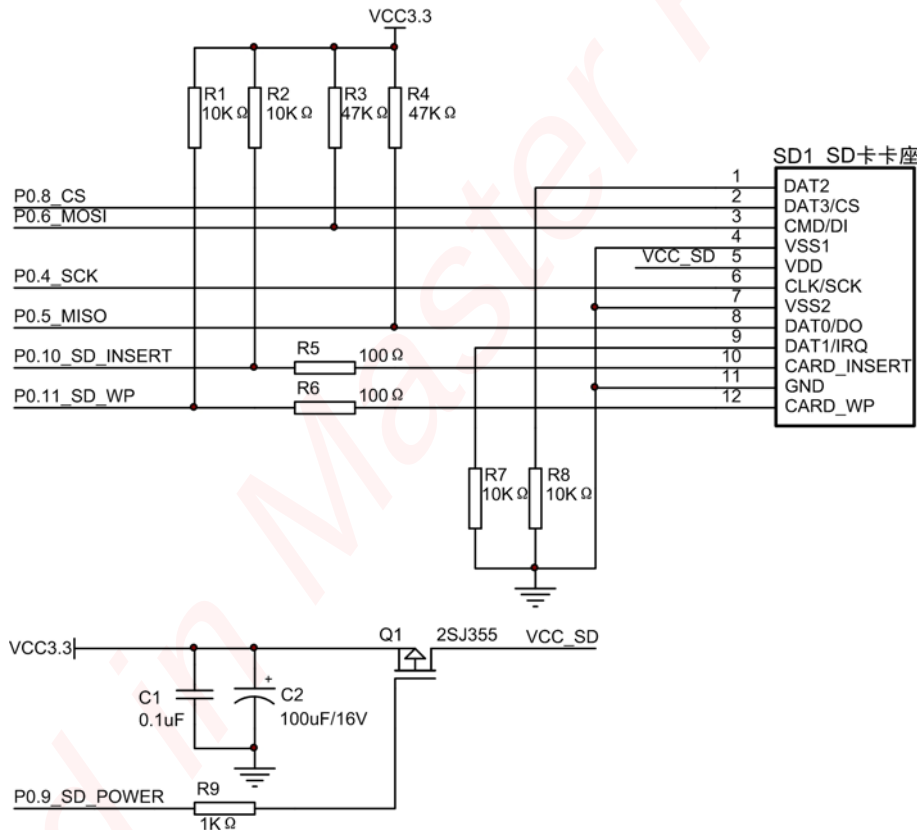


图 1.5 SD 卡卡座与 LPC2103 接口电路（SPI 模式）

图中，LPC2103 与 SD/MMC 卡卡座的连接引脚如表 1.4 所示。

表 1.4 LPC2103 与 SD/MMC 卡卡座的连接引脚

| LPC2103 接口 | 含 义                               |
|------------|-----------------------------------|
| P0.8_CS    | SPI 片选信号，用于选择 SPI 从机，该引脚为普通 I/O 口 |
| P0.4_SCK   | 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_MOSI  | SPI 主机输出，从机输入信号                   |

续上表

| LPC2103 接口      | 含 义                               |
|-----------------|-----------------------------------|
| P0.5_MISO       | SPI 主机输入，从机输出信号                   |
| P0.9_SD_POWER   | 卡供电控制，当 LPC2103 的 P0.9 输出低电平时给卡供电 |
| P0.10_SD_INSERT | 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 |
| P0.11_SD_WP     | 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。

## 1.2.1 SPI 总线

如图 1.5 所示，LPC2103 SPI 接口的 P0.8\_CS、P0.4\_SCK、P0.6\_MOSI、P0.5\_MISO 直接连接到卡座的相应接口，其中 SPI 的两个数据线 P0.6\_MOSI、P0.5\_MISO 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 LPC2103 的 GPIO 口 P0.9\_SD\_POWER 进行控制，当 P0.9\_SD\_POWER 输出高电平时，2SJ355 关断，不给卡供电；当 P0.9\_SD\_POWER 输出低电平时，2SJ355 开通，VCC3.3 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，P0.10\_CARD\_INSERT（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 LPC2103 的输入引脚 GPIO(P0.10\_SD\_INSERT)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 SD/MMC 卡读写模块的文件结构及整体构架

本小节介绍本模块的组成文件以及它们之间的关系。

### 1.3.1 SD/MMC 卡读写模块的文件组成

SD/MMC 卡读写模块包括的文件如表 1.5 所示。

表 1.5 SD/MMC 卡读写模块包含的文件

| 文 件        | 作 用   |
|------------|---|
| sdconfig.h | 卡读写模块硬件配置头文件                                    |
| sdsphal.c  | 读写模块硬件抽象层，实现 SPI 接口初始化，SPI 字节的收、发等与 SPI 硬件相关的函数 |

续上表

| 文 件        | 作 用                                    |
|------------|--|
| sdspihal.h | sdspihal.c 头文件                         |
| sdcmd.c    | 读写模块命令层，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 | 读写模块应用层，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                            |

表 1.5 中这些文件构成了本模块，下面说明由这些文件构成的整体框架。

### 1.3.2 SD/MMC 读写模块整体框架

考虑到该模块的可移植性及易用性，将模块分为 3 个层，如图 1.6 所示。图中的实时操作系统并不是必须的，也就是说，本模块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，本模块提供在前后台系统和实时操作系统  $\mu\text{C}/\text{OS-II}$  中接口统一的 API 函数。

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

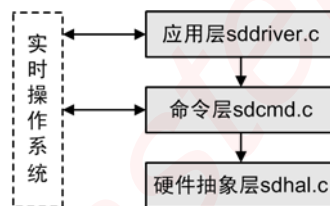


图 1.6 SD/MMC 卡读写模块结构图

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。如果采用操作系统，这一层由实时操作系统控制。

### 1.4 SD/MMC 卡读写模块的使用说明

使用本模块之前，必须配置好本模块使用的硬件条件，如果硬件条件与 1.2 小节中的硬件条件一样，那么无须配置本软件包就可以立即使用。下面说明怎样配置本模块的硬件条件，才能将其用于 LPC2103 系列微控制器。

#### 1.4.1 SD/MMC 卡读写模块的硬件配置

SD/MMC 卡读写模块在 LPC2103 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模块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. 模块参数配置

模块的参数配置如程序清单 1.1 所示，配置选项如下：

- (1) **是否运行于  $\mu\text{C}/\text{OS-II}$  中。**本模块既可以运行于前后台系统中，又可以运行于实时操作系统  $\mu\text{C}/\text{OS-II}$  中。当运行于  $\mu\text{C}/\text{OS-II}$  中时，宏定义 `SD_UCOSII_EN` 的值应置为 1，否则置为 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读写模块的数据传输 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，由于本模块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。
- (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 1.1 模块参数配置

```
#define SD_UCOSII_EN      1          /* 是否在  $\mu\text{C}/\text{OS-II}$  上运行本模块 */
#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

模块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 1.2 的宏定义用于使能编译读写模块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读写模块代码大小的目的。

程序清单 1.2 模块函数使能

```
/* 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 LPC2103 的 4 个 I/O 口为 SPI 接口的宏定义如程序清单 1.3(1) 所示。对于 SD 卡卡座供电引脚的控制如程序清单 1.3(2) 所示。用户阅读 LPC2103 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，详见 `sdconfig.h` 文件。



程序清单 1.3 LPC2103 的 IO 口配置宏定义

```

/* 初始化 IO 口为 SPI 接口 */
#define SPI_INIT()    PINSEL0 &= ~((0x03 << 8) + (0x03 << 10) + (0x03 << 12)); \
                    PINSEL0 |= (0x01 << 8) + (0x01 << 10) + (0x01 << 12);      (1)
/* 电源控制引脚 */                                       (2)
#define SD_POWER          (0x01 << 9)
#define SD_POWER_GPIO()   PINSEL0 &= ~(0x03 << 18)    /* 设置 POWER 口为 GPIO 口 */
#define SD_POWER_OUT()    IODIR |= SD_POWER          /* 设置 POWER 口为输出口 */
#define SD_POWER_OFF()    IOSET = SD_POWER           /* 置 POWER 为高电平 */
#define SD_POWER_ON()     IOCLR = SD_POWER           /* 置 POWER 为低电平 */
    
```

配置头文件的全部内容就介绍到此，如果要本模块移植到其它 MCU，则还需修改 sdhal.c 文件，这一部分与 MCU 的 SPI 接口操作相关，如果使用 LPC2103，那么无须改动该文件。

还有一点容易忽略的，就是必须将 LPC2103 的外设时钟频率 Fpclk 调至最高（在允许范围内），这样，读写模块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。LPC2103 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，这样本模块才能达到兼容 MMC 卡的目的。该函数如程序清单 1.4 所示（见 sdhal.c 文件）。该函数只是修改 LPC2103 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 1.4 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 LPC2103 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：

$$44.2368 / 128 = 0.3456 \text{ MHz} = 345.6 \text{ kHz} < 400 \text{ kHz}。$$

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：

$$44.2368 / 8 = 5.5296 \text{ MHz}。$$

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 LPC2103 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，来优化读写模块对卡的访问速度。

配置好了硬件，那么可以使用本模块了，那么本模块提供了哪些 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 SD/MMC 卡读写模块提供的 API 函数

用户可以利用本模块提供的 API 函数对 SD/MMC 卡进行访问，见表 1.6 至表 1.11。

表 1.6 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 1.12） |
| 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 1.7 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<br>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 1.12）  |
| 特殊说明和注意点 | recbuf 的长度必须是 512 字节   |

表 1.8 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<br>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 1.12）   |
| 特殊说明和注意点 | sendbuf 的长度必须是 512 字节   |

表 1.9 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  |

续上表

|          |  |
|----------|--|
| 函数参数     | blockaddr: 以块为单位的块地址<br>blocknum: 块数<br>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 1.12)                                      |
| 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 1.10 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<br>blocknum: 块数<br>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 1.12)                                       |
| 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 1.11 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<br>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 1.12)  |
| 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 1.6 至表 1.11 函数返回值所代表的含义如表 1.12 所示。

表 1.12 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 (与本模块不兼容)     |
| 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                |

续上表

| 错误码宏定义                  | 宏定义值 | 含义                     |
|-------------------------|------|------------------------|
| 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            |

下面给出使用 SD/MMC 卡读写模块的一个例子。

## 1.5 SD/MMC 卡读写模块的应用示例一

下面给出在 LPC2103 微处理器上使用 SD/MMC 卡读写模块对 SD/MMC 卡进行读、写数据的例子。该例子利用 LPC2103 提供的 SPI 接口读写 SD/MMC 卡，并将写入的数据读出后与原始数据做比较，验证读写操作的正确性。

### 1.5.1 硬件连接与配置

用杜邦线将 EasyARM2103 与 SD CARD PACK 连接起来，连线方法如表 1.13 所示。

表 1.13 EasyARM2103 与 SD CARD PACK 连接关系

| EasyARM2103(JP5) | SD CARD PACK 引脚 (J1) | 引线含义                   |
|------------------|----------------------|------------------------|
| 3.3V             | 3.3V                 | SD CARD PACK 供电电源      |
| GND              | GND                  | 电源地                    |
| P0.9             | POW_C                | 控制 3.3V 电源供给卡          |
| P0.8             | CS                   | 选择 SD/MMC 卡            |
| P0.6             | MOSI                 | 主机 SPI 数据输出，卡 SPI 数据输入 |
| P0.4             | SCK                  | SPI 总线时钟               |
| P0.5             | MISO                 | 主机 SPI 数据输入，卡 SPI 数据输出 |
| P0.10            | INSERT               | 卡完全插入卡座检测              |
| P0.11            | WP                   | 卡写保护机械开关检测             |

SD/MMC 卡读写模块默认的硬件配置和表 1.13 的硬件条件相符。因此，不需要对读写模块进行配置。

### 1.5.2 实现方法

本例子将对 SD/MMC 卡进行读、写和擦除等常用操作。例子的软件结构如图 1.7 所示。

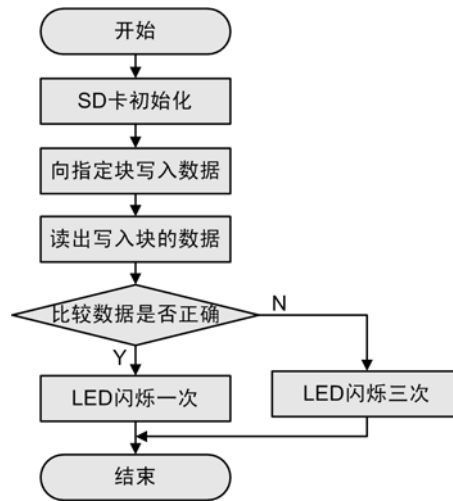


图 1.7 主函数流程图

SD 卡应用示例主函数如程序清单 1.5 所示。

程序清单 1.5 SD 卡操作主函数代码

```
int main (void)
{
    uint32 i;
    uint8 status;
    uint8 sdbuf[512]={0}; /* 存放写入数据缓冲区 */
    uint8 sdbuf2[512]={0}; /* 存放读出数据缓冲区 */

    PINSEL1 = 0x00000000; /* 设置管脚连接 GPIO */
    IO0DIR |= BEEP; /* 设置 BEEP 控制口为输出 */
    IO0SET = BEEP;

    for(i=0;i<512;i++){ /* 初始化写入数据 */
        sdbuf[i] = i&0xff;
    }
    status = SD_Initialize(&sds); /* SD 初始化 */
    if (status != SD_NO_ERR){
        while(1);
    }
    status = SD_WriteBlock(&sds,0,sdbuf); /* 将 sdbuf 缓冲区数据写入第 0 块中*/
    if (status != SD_NO_ERR){
        while(1);
    }
    status = SD_ReadBlock(&sds,0,sdbuf2); /* 读第 0 块的数据 */
    if (status != SD_NO_ERR){
        while(1);
    }
    status = memcmp(sdbuf,sdbuf2,512); /* 对 sdbuf2 与 sdbuf 的内容进行比较*/
}
```

```
if(status!=0){ /* 数据比较错误,蜂鸣器蜂鸣三声 */
    BeepOnOff(3);
}
else{ /* 数据比较正确,蜂鸣一声 */
    BeepOnOff(1);
}
while(1);
return 0;
}
```

下文将对程序清单 1.5 中的 SD 卡初始化、SD 卡单块读和单块写等常用命令进行简要分析讲解。

## 1. SD 卡初始化

SD 卡初始化流程如图 1.8 所示，首先初始化访问卡的硬件条件，SdSpiHal\_Initialize() 函数代码如程序清单 1.6 所示。

程序清单 1.6 初始化卡访问卡的硬件条件

```
INT8U SdSpiHal_Initialize(sd_struct *sds)
{
    SD_Power(); /* 对卡先下电,再上电 */
    SPI_INIT(); /* 初始化 SPI 接口 */
    SD_INSERT_GPIO();
    SD_INSERT_IN(); /* 检测卡完全插入口为输入口 */ (1)
    SD_WP_GPIO();
    SD_WP_IN(); /* 写保护检测口为输入口 */ (2)
    SPI_CS_SET(); /* CS 置高 */
    SdSpiHal_SetMCIClock(sds, SD_RATE_SLOW); /* 设置 SPI 频率小于等于 400kHz */
    SPI_SPCR = 0 << 3 | /* CPHA = 0 第一个时钟采样 */
              1 << 4 | /* CPOL = 1, SCK 低有效 */
              1 << 5 | /* MSTR = 1, 设置为主模式 */
              0 << 6 | /* LSBF = 0, SPI 传输 MSB 在先 */
              0 << 7; /* SPIE = 0, SPI 中断禁止 */
    return SD_NO_ERR;
}
```

对 SD 卡先下电，再上电后，对 SPI 总线接口进行初始化。程序清单 1.6 (1) 将用于检测卡是否完全插入的卡座的 I/O 引脚初始化为 GPIO，并且设置为输入口。程序清单 1.6 (2) 将用于检测卡是否写保护的 I/O 引脚初始化为 GPIO，并且设置为输入口。

设置 SPI 的 SCK 引脚输出频率小于等于 400KHz，是因为 MMC 卡在复位阶段要求 SPI 的时钟频率要小于等于 400KHz。

注意：在本软件包中没有用到 LPC2103 的 SPI 模块的 P0.7 (SPI 从机选择输入) 引脚，由于芯片特性要求，在 LPC2103 的 SPI 模块不充当从机时，不需要将此引脚初始化为 SPI 模式，如果初始化，SPI 模块将自动转入从机模式。

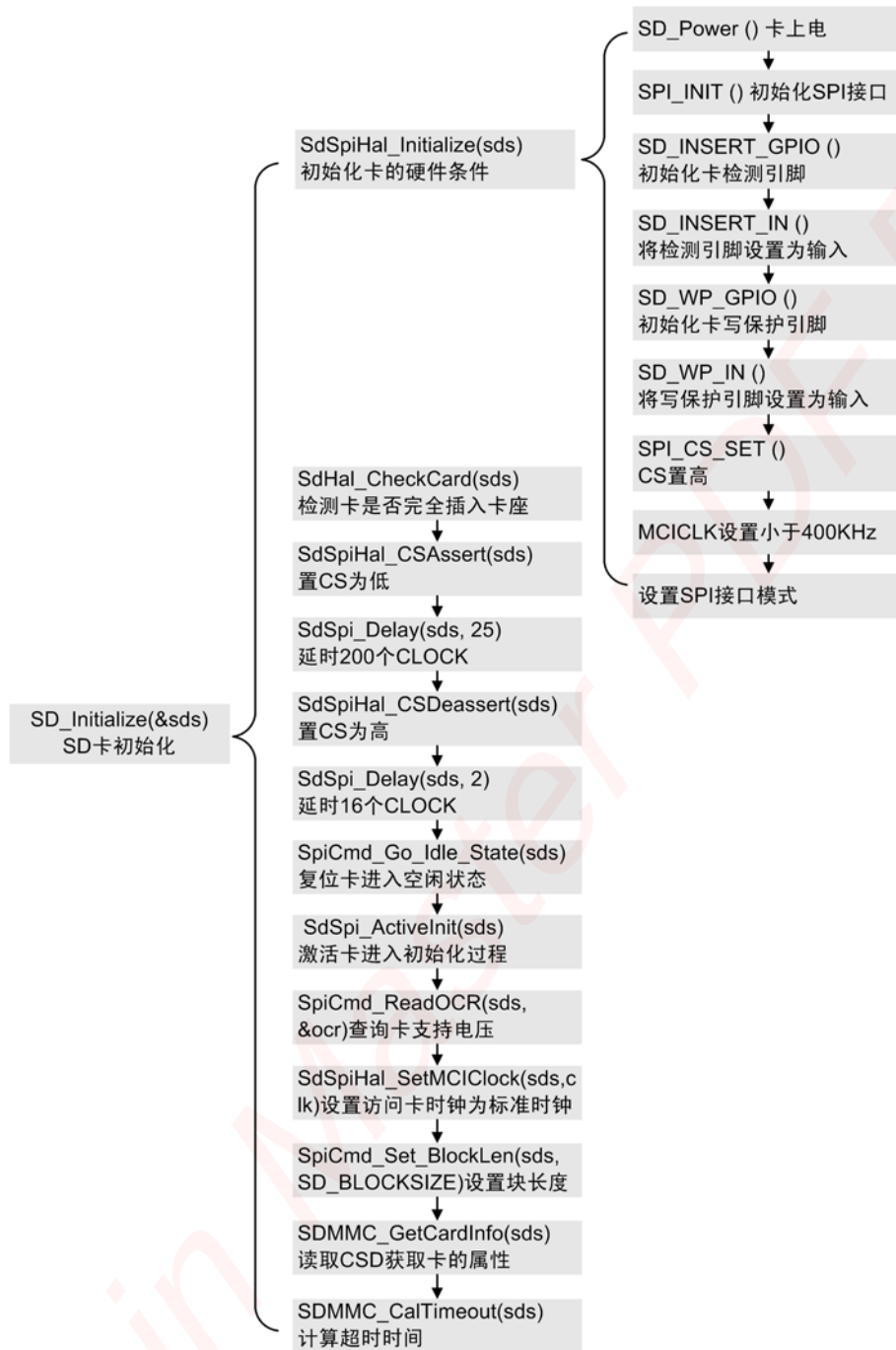


图 1.8 SD 卡初始化流程

卡完全插入卡座与卡写保护检测只须检测相关 I/O 口的电平即可。卡完全插入卡座检测函数如程序清单 1.7 所示。该函数返回 0 表示卡未完全插入，此时，SD/MMC 卡读/写软件包不能对卡进行操作。当卡完全插入时，P0.10\_SD\_INSERT 引脚输出低电平。

程序清单 1.7 卡完全插入卡座检测函数

```

INT8U SdHal_CheckCard(sd_struct *sds)
{
    if (SD_INSERT_STATUS() != 0)
        return 0; /* 未完全插入 not insert entirely */
}
    
```

```

else
    return 1; /* 完全插入 insert entirely */
}
    
```

对于 SD 卡初始化流程中的其它操作如复位卡进入空闲状态、激活卡进入初始化以及设置块长度读取卡信息等操作，是访问应用 SD/MMC 卡前规定必须要进行的准备性工作，只有获取相应的信息，并将卡设置为合适的状态，才可对卡进行读写访问。

## 2. SD 卡单块写操作

SD/MMC 卡在 SPI 模式下的写操作包括两种：写单块和写多块。本示例重点介绍写单块操作。卡初始化函数 SD\_Initialize()已经调用了 SpiCmd\_Set\_BlockLen()函数设定了读/写数据块的长度 SD\_BLOCKSIZE 字节，卡在初始化后，读/写都是以块为单位，一次写操作至少要写 SD\_BLOCKSIZE 字节。SD\_BLOCKSIZE 字节一般都为 512 字节。

SD 卡单块写操作流程如图 1.9 所示。写单块是这样进行的：

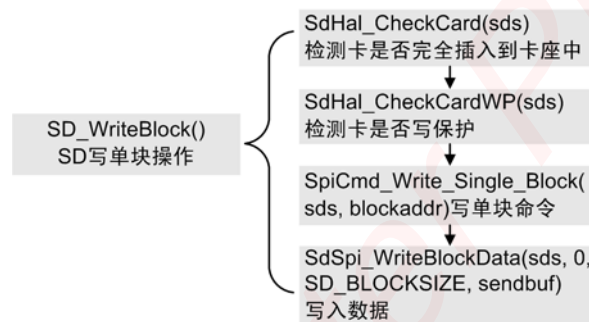


图 1.9 SD 卡单块写操作流程

- (1) 主机检测卡是否完全插入到卡座中；
- (2) 主机检测卡是否写保护；
- (3) 以上条件满足后，主机向卡发送写单块命令，写入地址为 blockaddr 的一个数据块；
- (4) 主机紧跟这向卡发送要写入的数据块，数据块长度为 SD\_BLOCKSIZE。

SD 卡单块写操作程序清单如程序清单 1.8 所示，建议客户的应用时直接调用软件包的 API 函数操作。

程序清单 1.8 SD 卡单块写操作代码

```

/*****
** 函数名称: SD_WriteBlock
** 功能描述: SPI 模式下, 向 SD/MMC 卡中写入一个块
** 输入: sd_struct *sds: SD/MMC 卡信息结构体
**      INT32U 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
**      INT8U *sendbuf : 发送缓冲区,长度固定为 512 字节
** 输出: 无
** 返回值: 0: 正确 >0: 错误码, 见 sddriver.h 文件
*****/
INT8U SD_WriteBlock(sd_struct *sds, INT32U blockaddr, INT8U *sendbuf)
{
    INT8U ret,tmp[2];
    
```



```
SD_RequestOSSem(sds); /* 向 OS 申请访问卡信号量 */
if (!SdHal_CheckCard(sds)) {
    SD_ReleaseOSSem(sds);
    return SD_ERR_NO_CARD; /* 卡没完全插入卡中 */
}
if (blockaddr > sds->block_num) {
    SD_ReleaseOSSem(sds);
    return SD_ERR_OVER_CARDRANGE; /* 操作超出卡容量范围 */
}
if (SdHal_CheckCardWP(sds)) {
    SD_ReleaseOSSem(sds);
    return SD_ERR_WRITE_PROTECT; /* 卡有写保护 */
}
ret = SpiCmd_Write_Single_Block(sds, blockaddr); /* 写单块命令 */
if (ret != SD_NO_ERR) {
    SD_ReleaseOSSem(sds);
    return ret;
}
ret = SdSpi_WriteBlockData(sds, 0, SD_BLOCKSIZE, sendbuf); /* 写入数据 */
if (ret == SD_NO_ERR) { /* 读寄存器, 检查写入是否成功 */
    ret = SpiCmd_Send_Status(sds, 2, tmp);
    if (ret != SD_NO_ERR) {
        SD_ReleaseOSSem(sds);
        return ret; /* 读寄存器失败 */
    }
    if ((tmp[0] != 0) || (tmp[1] != 0)) {
        SD_ReleaseOSSem(sds);
        ret = SD_ERR_WRITE_BLK; /* 响应指示写失败 */
    }
}
SD_ReleaseOSSem(sds);
return ret; /* 返回写入结果 */
}
```

### 3. SD 卡单块读操作

SD/MMC 卡在 SPI 模式下的读操作也包括两种：读单块和读多块。本示例重点介绍读单块操作。卡初始化函数 `SD_Initialize()` 已经调用了 `SpiCmd_Set_BlockLen()` 函数设定了读/写数据块的长度 `SD_BLOCKSIZE` 字节。卡在初始化后，读/写都是以块为单位，所以一次读操作至少要读 `SD_BLOCKSIZE` 个字节。`SD_BLOCKSIZE` 字节一般都为 512 字节。

SD 卡单块读操作流程如图 1.10 所示。读单块是这样进行的：



图 1.10 SD 单块读操作流程图

- (1) 主机首先检查卡是否已经完全插入卡座中;
- (2) 检查块地址是否超出卡的容量范围;
- (3) 以上条件满足后, 向卡发送读单块命令, 读取地址为 blockaddr 的一个数据块;
- (4) 调用读取块数据函数从卡读取一个数据块。

SD 卡单块读操作程序清单如程序清单 1.9 所示, 建议客户的应用时直接调用软件包的 API 函数操作。

程序清单 1.9 SD 卡单块读操作代码

```

/*****
** 函数名称: SD_ReadBlock
** 功能描述: SPI 模式下, 从 SD/MMC 卡中读出一个数据块
** 输入: sd_struct *sds: SD/MMC 卡信息结构体
**      INT32U 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
** 输出: INT8U *recbuf: 接收缓冲区, 长度固定为 512 字节
** 返回值: 0: 正确    >0: 错误码, 见 sddriver.h 文件
*****/
INT8U SD_ReadBlock(sd_struct *sds, INT32U blockaddr, INT8U *recbuf)
{
    INT8U ret;

    SD_RequestOSSem(sds);                               /* 向 OS 申请访问卡信号量 */
    if (!SdHal_CheckCard(sds)) {
        SD_ReleaseOSSem(sds);
        return SD_ERR_NO_CARD;                          /* 卡没完全插入卡中 */
    }

    if (blockaddr > sds->block_num) {
        SD_ReleaseOSSem(sds);
        return SD_ERR_OVER_CARDRANGE;                  /* 操作超出卡容量范围 */
    }

    ret = SpiCmd_Read_Single_Block(sds, blockaddr);    /* 读单块命令 */
    if (ret != SD_NO_ERR) {

```

```

SD_ReleaseOSSem(sds);
return ret;
}

ret = SdSpi_ReadBlockData(sds, SD_BLOCKSIZE, recbuf); /* 读出数据 */
SD_ReleaseOSSem(sds); /* 归还访问卡信号量 */

return ret;
}

```

## 1.6 SD/MMC 卡读写模块的使用示例二

下面给出在 LPC2103 微处理器上使用 SD/MMC 卡读写模块对 SD/MMC 卡进行读、写、擦的例子。该例子基于 uCOS-II 操作系统，利用 LPC2103 提供的 SPI 接口读写 SD/MMC 卡，读取的数据通过 LPC2103 的 UART0 发送到 PC 机的软件界面显示出来，要写入卡的数据也可以通过 PC 机软件通过 UART0 写入到卡中，另外，本例子也提供了擦卡的演示。

### 1.6.1 实现方法

本例子最主要的任务是演示读写 SD/MMC 卡，为了实现能够实现 LPC2103 与 PC 的通信，本例子使用了串口中间件与数据队列中间件，来收发来自 PC 机的数据。例子的软件结构如图 1.11 所示。

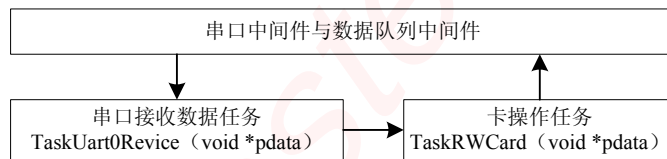


图 1.11 例子实现方法

图 1.11 中，本例子只包括两个中间件和两个任务：

- **串口中间件与数据队列中间件。**这两个中间件用于接收来自 UART0 的数据，或将数据通过 UART0 发送到 PC。
- **串口接收数据任务。**接收来自 PC 机的数据，并负责将收到的数据发送给卡操作任务。
- **卡操作任务。**接收来自串口接收数据任务的数据，根据数据中包括的命令和数据实现对 SD/MMC 卡的读、写、擦。并将执行结果或数据提交给串口中间件与数据队列中间件，由它们发送到 PC 机，报告读到的数据或执行的结果。

可见，PC 机与卡操作任务之间需要一个协议来协调，才能正确完成对 SD/MMC 卡的初始化、读、写、擦。

因此，可以制定一个简单的协议来完成：

- (1) 定义 LPC2103 的 UART0 接收触发中断深度为 8 个字节，并定义 8 个字节为一帧，数据的传输以帧为单位，PC 机发送的帧称为命令帧，LPC2103 发送的帧称为响应帧。
- (2) PC 机发送的每一帧的第 1 个字节为命令字，2 ~ 8 个字节为数据部分，如图 1.12 所示。
- (3) LPC2103 每收到 PC 机的一个帧，根据命令字进行相应的处理（如读卡），然后将处理结果打包成一帧发回 PC 机，帧头为 LPC2103 收到的命令字，2 ~ 8 字节为响

应部分。

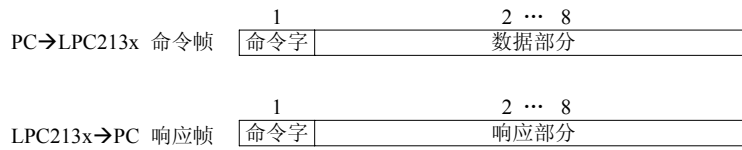


图 1.12 PC 机发送的命令帧与 LPC2103 的响应帧

- (4) 在内存中开辟一片缓冲区 INT8U sd\_buf[512]，大小为 520 字节（比 SD/MMC 卡的一个块大 8 个字节）。用于保存从卡中读到的数据或即将写进卡中的数据。
- (5) 根据以上四点，制定卡操作任务对 SD/MMC 卡进行初始化、读、写、擦的方法，其示意图如图 1.13 所示。

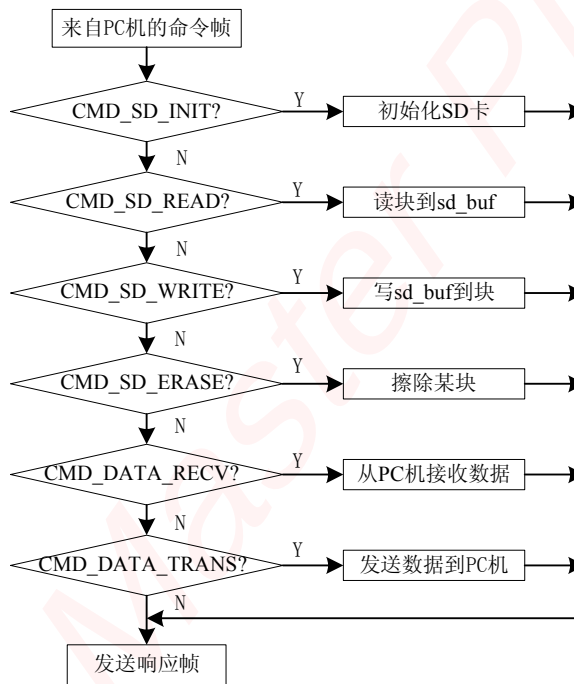


图 1.13 卡操作任务对命令帧的响应示意图

如图 1.13 所示，卡操作任务根据命令帧的第 1 个字节，执行下面的分支。

- **CMD\_SD\_INIT 命令，卡初始化命令。**卡操作任务执行对 SD/MMC 卡的初始化，然后将执行结果放入响应帧的第 2 个字节，向 PC 机报告初始化结果。
- **CMD\_SD\_READ 命令，读指定的块。**PC 机将要读的块地址放入 CMD\_SD\_READ 命令后的两个字节中，卡操作任务调用读块函数读出指定的块，暂存于 sd\_buf[] 中，读块函数的返回值放入响应帧的第 2 个字节，向 PC 机报告读操作是否成功。
- **CMD\_SD\_WRITE 命令，写指定的块。**PC 机将要写的块地址放入 CMD\_SD\_WRITE 命令后的两个字节中，卡操作任务调用写块函数将 sd\_buf[] 中的数据写入卡中，写块函数的返回值放入响应帧的第 2 个字节，向 PC 机报告写操作是否成功。
- **CMD\_SD\_ERASE 命令，擦除指定的块。**PC 机将要擦除的块的地址与要擦除的块数放在该命令字的后面，卡操作任务调用擦块函数将指定的块擦除，擦除函数的返回值放入响应帧的第 2 个字节，向 PC 报告擦除操作是否成功。

- **CMD\_SD\_RECV 命令，接收数据命令。**该命令帧的 2 ~ 3 字节指出接收数据的 sd\_buf[]地址，4 ~ 8 字节为 PC 发来的数据，卡操作任务将接收到的数据放入 sd\_buf[]的中。
  - **CMD\_SD\_TRANS 命令，发送数据命令。**该帧的第 2 和第 3 个字节指明 PC 机要读取 sd\_buf[]中的数据缓冲区地址，卡操作任务将该地址的数据发往 PC 机，数据存放于响应帧的 2 ~ 8 字节中。  
执行以上任何一个分支后，卡操作任务都发送响应帧到 PC 机。
- (6) 利用以上命令，对卡进行初始化、读、写、擦是这样进行的，如图 1.14 所示。

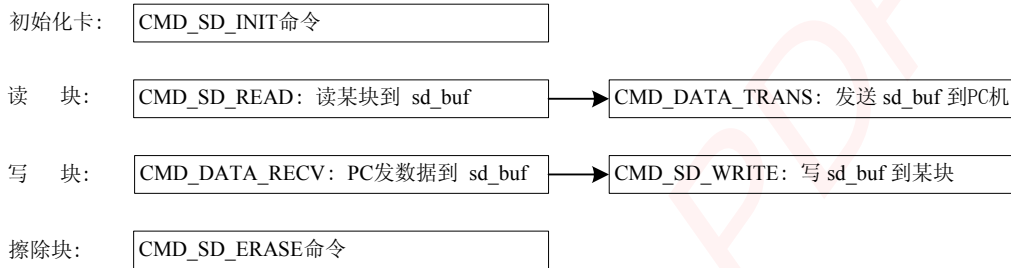


图 1.14 对卡进行各种操作需要执行的命令分支

以上几点是实现图 1.11 的具体协议。下面先给出实现本例子的实验步骤，然后给出实现以上协议的示例代码。

### 1.6.2 例子建立与运行步骤

下面简单说明本例子的建立过程。

1. 用 ADS1.2 建立一个工程，工程名为 SDMMCEXAM，建立时使用工程模板 ARM Image for uCOSII for LPC2103。建立完成后，生成文件夹 SDMMCEXAM。
2. 在 SDMMCEXAM 目录下建立一个目录 SDMMC，在该目录下放置 SD/MMC 卡读写模块的所有文件。
3. 在 SDMMCEXAM 的同级目录下建立 arm 和 Source 文件夹，Source 文件夹放置  $\mu$ C/OS-II 源代码。arm 目录中放置与 LPC2000 硬件相关的  $\mu$ C/OS-II 移植代码。
4. 在 SDMMCEXAM 目录下建立一个文件夹 uart0，将串口中间件相关文件放于该文件夹中，建立 queue 文件夹，将数据队列中间件相关文件放于该文件夹中。
5. 在本工程中建立 3 个组，分别为 SDMMC、uart、queue，将对应的模块或中间件加入这些组中。
6. 在 config.h 中删除原有的“#include "..\..\Arm\_Pc\pc.h”语句。
7. 打开工程中的 config.h 文件，将 LPC2103 外设时钟频率做以下修改。

```
#define Fpclk (Fcclk / 4) * 4
```

8. 打开 sdconfig.h 文件，将宏定义 SD\_EraseBlock\_EN 的值置为 1。

```
#define SD_EraseBlock_EN 1
```

9. 在 config.h 文件中加入相关模块和中间件的头文件及配置。如程序清单 1.10 所示。

程序清单 1.10 相关模块和中间件的头文件及配置

```
/* SD/MMC 模块头文件 */
#include "sdconfig.h"
```

```
#include "sddriver.h"

/* 数据队列的配置 */
#define QUEUE_DATA_TYPE uint8
#include "\queue\queue.h"
#define EN_QUEUE_WRITE 1 /* 禁止(0)或允许(1)FIFO 发送数据 */
#define EN_QUEUE_WRITE_FRONT 0 /* 禁止(0)或允许(1)LIFO 发送数据 */
#define EN_QUEUE_NDATA 1 /* 禁止(0)或允许(1)取得队列数据数目 */
#define EN_QUEUE_SIZE 1 /* 禁止(0)或允许(1)取得队列数据总容量 */
#define EN_QUEUE_FLUSH 0 /* 禁止(0)或允许(1)清空队列 */

/* UART0 的配置 */
#include "uart0.h"
#define UART0_SEND_QUEUE_LENGTH 60 /* 给 UART0 发送数据队列分配的空间大小 */
```

10. 打开 `os_cfg.h` 文件，将操作系统使用的最大事件数改为 3。

```
#define OS_MAX_EVENTS 3
```

11. 如果硬件接线和图 1.5 一样，那么无须配置 `sdconfig.h`，否则按 1.4.1 小节的说明进行配置。建议将 `sdconfig.h` 文件放于 `SDMMCEXAM\src` 目录下，因为这是配置 SD/MMC 读写模块的文件，是可以被用户修改的文件。
12. 在工程 `irq.s` 文件的最后添加 UART0 中断服务程序的汇编语言部分代码，如程序清单 1.11 所示

程序清单 1.11 增加 UART0 中断服务程序的代码

```
;/*定时器 0 中断*/
;/*Time0 Interrupt*/
Timer0_Handler HANDLER Timer0_Exception

;/*通用串行口 0 中断*/
UART0_Handler HANDLER UART0_Exception
```

13. 在 `Target.c` 文件的 `TargetInit()` 函数中，添加初始化 UART0 的代码，如程序清单 1.12 所示。

程序清单 1.12 添加 UART0 初始化代码

```
void TargetInit(void)
{
    OS_ENTER_CRITICAL();
    srand((uint32) TargetInit);
    VICInit();
    Timer0Init();
    UART0Init(115200);
    OS_EXIT_CRITICAL();
}
```

14. 在 Target.c 文件的 VICInit()函数中，添加 UART0 向量中断的初始化代码，如程序清单 1.13 所示。

程序清单 1.13 UART0 向量中断初始化

```
void VICInit(void)
{
    extern void IRQ_Handler(void);
    extern void Timer0_Handler(void);
    extern void UART0_Handler(void);

    VICIntEnClr = 0xffffffff;
    VICDefVectAddr = (uint32)IRQ_Handler;

    VICVectAddr14 = (uint32)UART0_Handler;
    VICVectCntl14 = (0x20 | 0x06);
    VICIntEnable = 1 << 6;

    VICVectAddr15 = (uint32)Timer0_Handler;
    VICVectCntl15 = (0x20 | 0x04);
    VICIntEnable = 1 << 4;
}
```

15. 根据例子的协议分析，在 main.c 文件中编写通过串口读写 SD/MMC 卡的相关函数。
16. 选择 DebugInFlash 生成目标，然后编译连接工程。在 ADS1.2 集成开发环境中选择 Project→Debug，启动 AXD 进行 JTAG 仿真调试，并全速运行程序。
17. 确保硬件连接正确，SD 或 MMC 卡已插入到卡座中。
18. 将产品光盘中提供的 PC 机端可执行软件 SDEExample.exe 复制到硬盘，在 ADS1.2 的程序运行后运行该软件。软件界面如图 1.15 所示。
19. 在图 1.15 中，选择串口号及通信波特率，然后按“连接 LPC2103”按键，“执行结果”框中将显示该软件是否能与 LPC2103 成功通信。
20. 如果打开串口成功，那么请按“初始化 SD/MMC 卡”，是否初始化成功将在“执行结果”中显示出来。如果初始化失败，该“执行结果”将显示出错误代码，显示的错误码与表 1.12 一一对应，给出错误码的目的是方便用户调试程序。
21. SD/MMC 卡初始化成功后，在“读写擦”框中填写要读的块的块地址，然后，按“读”按键，读得的块的内容在“数据显示”框中显示出来，“执行结果”框将显示执行结果，如果执行结果有错，将给出错误代码。

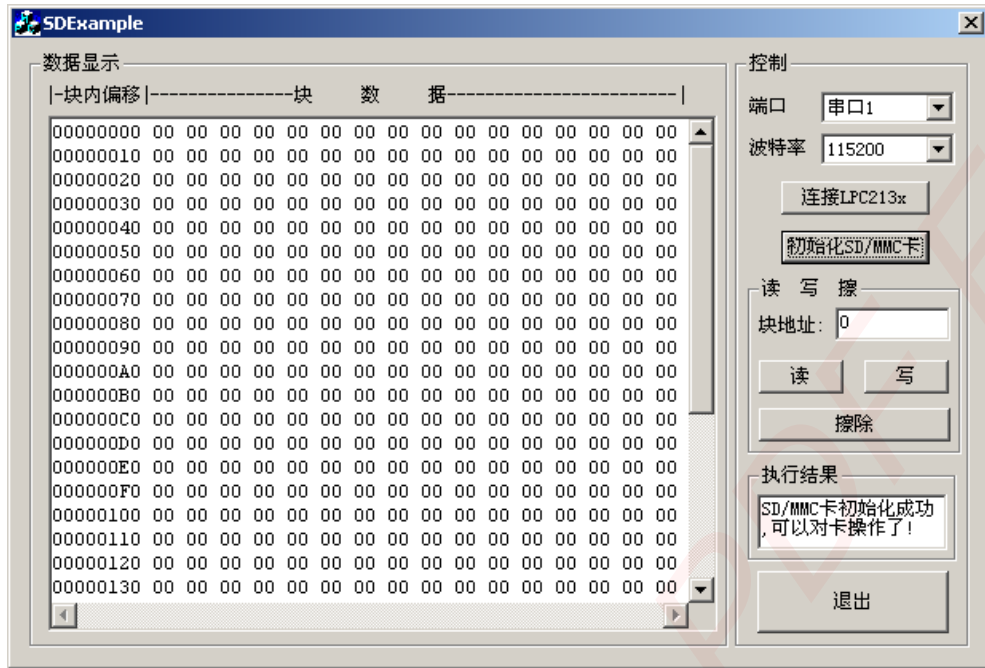


图 1.15 连接 LPC2103 并初始化 SD/MMC 卡

22. 如果想要写某块，则需要用鼠标点击在“数据显示”框中的“块数据”区，然后用键盘键入要写入的数据，然后按“写”按键，软件将该块写入到 SD/MMC 卡中，“执行结果”框将显示执行结果。如图 1.16 所示。

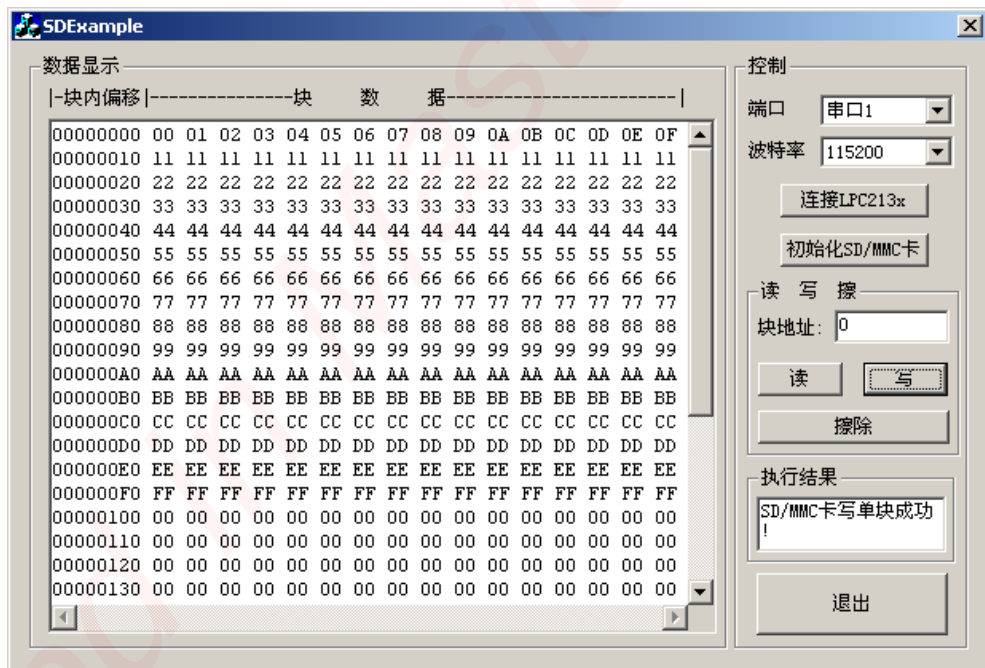


图 1.16 写 SD/MMC 卡

如果想要擦 SD/MMC 卡，那么请按“擦除”按键，将弹出如图 1.17 所示的擦除对话框，填写好起始块地址和要擦除的块数，然后按“擦除”。擦除操作执行结果将在图 1.16 所示的“执行结果”中显示出来。擦除限制 5000 块的目的是防止擦除时间太长导致串行通信超时，并非说 SD/MMC 卡一次擦除操作只能擦 5000 块。



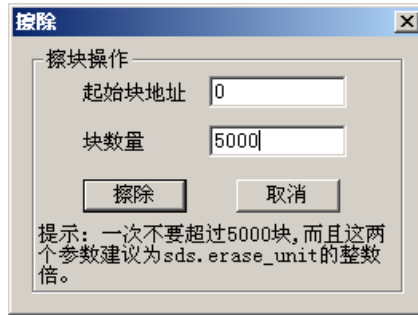


图 1.17 擦除操作对话框

图 1.17 还提示用户擦卡的“起始块地址”和“块数量”建议为 `sds.erase_unit` 的整数倍。这是因为有一些卡擦卡的单位为 `sds.erase_unit` 块，也就是说，即使命令卡只擦除“起始块地址”这一块，但 SD/MMC 卡将擦除“起始块地址”开始的 `sds.erase_unit` 块。

### 1.6.3 参考程序

1. 在 `main.c` 文件中加入相关头文件及定义即将用到的全局变量。如程序清单 1.14 所示。

程序清单 1.14 加入相关头文件及定义相关全局变量

```
#include "config.h"

#define TASK_STK_SIZE      64                                (1)

#define CMD_SD_INIT        0x00                            /* 卡初始化卡命令 */ (2)
#define CMD_SD_READ        0x01                            /* 卡读命令 */
#define CMD_SD_WRITE       0x02                            /* 卡写命令 */
#define CMD_SD_ERASE       0x03                            /* 卡擦除命令 */

#define CMD_DATA_TRANS     0x04                            /* 将 sd_buf 中的数据发送到 PC 机 */
#define CMD_DATA_RECV      0x05                            /* 接收来自串口的数据,并放入 sd_buf 中 */

/* SD/MMC 卡读写缓冲区,比 SD/MMC 卡一个块大 8 字节 */
uint8 sd_buf[520];                                        (3)
```

程序清单 1.14(1)定义了任务使用的堆栈大小，程序清单 1.14(2)为 PC 机发送的命令帧的命令字定义，程序清单 1.14(3)为 SD/MMC 卡读写时使用的缓冲区，该缓冲区比 SD/MMC 卡的一个块大 8 个字节，这样定义的目的是为了串口操作的方便（串口的接收以 8 个字节作为接收中断触发深度）。

2. 定义各任务的堆栈及 UART0 接收数据邮箱，如程序清单 1.15 所示。

程序清单 1.15 堆栈及邮箱定义

```
OS_STK  TaskStk[TASK_STK_SIZE];                          /* 任务堆栈 */
OS_STK  TaskCardStk[TASK_STK_SIZE];                      /* 卡操作任务堆栈 */
OS_EVENT *Uart0ReviceMbox;                               /* 串口接收数据邮箱 */
```

3. 编写主函数 `main()`，如程序清单 1.16 所示。本函数创建了操作系统运行的第一个任务 `TaskCard()`，如程序清单 1.16(1)所示。

程序清单 1.16 `main()`函数

```
int main (void)
{
    OSInit();
    OSTaskCreate(TaskCard, (void *)0, &TaskCardStk[TASK_STK_SIZE - 1], 0);           (1)
    OSStart();
    return 0;
}
```

4. 编写卡操作任务 `TaskCard()`，该函数作为 `main()`函数创建的第 1 个任务，它将初始初始化相关硬件(如程序清单 1.17(3))，建立串口接收数据邮箱(如程序清单 1.17(1)所示)和其它任务(如程序清单 1.17(2)所示)，并将本任务做为卡操作任务。

程序清单 1.17 卡操作任务

```
void TaskCard(void *pdata)
{
    uint8 *pRec;
    uint8 err;
    uint32 bufaddr,blockaddr,blocknum;

    pdata = pdata; /* 避免编译警告 */ /* (1)
    Uart0ReviceMbox = OSMboxCreate(NULL); /* 建立邮箱 */ /* (1)
    if (Uart0ReviceMbox == NULL)
        while (1);
    OSTaskCreate(TaskUart0Revice, (void *)0,
                &TaskStk[TASK_STK_SIZE - 1], 10); /* 创建 Uart0 接收任务 */ /* (2)
    TargetInit(); /* 目标板初始化 */ /* (3)

    for (;;)
    {
        pRec = (uint8 *)OSMboxPend(Uart0ReviceMbox, 0, &err); /* 接收数据 */ /* (4)
        switch(pRec[0]) /* (5)
        {
            case CMD_SD_INIT: pRec[1] = SD_Initialize(); /* 初始化卡 */ /* (6)
            break;

            case CMD_SD_READ: /* (7)
                blockaddr = (pRec[1] << 24) + (pRec[2] << 16) + /* 计算块地址 */ /*
                    (pRec[3] << 8) + pRec[4];
                pRec[1] = SD_ReadBlock(blockaddr, sd_buf); /* 卡单块读 */ /*
            break;
```

```
case CMD_SD_WRITE:
    blockaddr = (pRec[1] << 24) + (pRec[2] << 16) +
                (pRec[3] << 8) + pRec[4];
    pRec[1] = SD_WriteBlock(blockaddr, sd_buf);    /* 卡单块写          */
    break;

case CMD_SD_ERASE:
    blockaddr = (pRec[1] << 24) + (pRec[2] << 16) +
                (pRec[3] << 8) + pRec[4];        /* 擦卡起始地址      */
    blocknum = (pRec[5] << 16) + (pRec[6] << 8) +
                (pRec[7]);                       /* 块数              */
    pRec[1] = SD_EraseBlock(blockaddr, blocknum); /* 擦除操作          */
    break;

case CMD_DATA_RECV:
    bufaddr = (pRec[1] << 8) + pRec[2];          /* 计算缓冲区地址    */
    memcpy(sd_buf + bufaddr, &pRec[3], 5);      /* 收到数据放入 pRec */
    break;

case CMD_DATA_TRANS:
    bufaddr = (pRec[1] << 8) + pRec[2];          /* 计算缓冲区地址    */
    memcpy(&pRec[1], sd_buf + bufaddr, 7);      /* sd_buf 数据放入 pRec */
    break;
default: break;
}
UART0Write(pRec, 8);                            /* 发送响应帧        */ (8)
}
}
```

程序清单 1.17(4) 邮箱等待串口接收数据任务 TaskUart0Revice()发来邮件，邮件内容即为接收到的数据的缓冲区起始地址 pRec。

程序清单 1.17(5) 邮箱中有内容，也就是收到 PC 机的一个命令帧，命令帧的第 1 个字节为命令字，那么执行各个命令分支。

程序清单 1.17(6) 执行初始化卡命令分支，执行结果保存于 pRec[1]中。

程序清单 1.17(7) 执行读块命令分支，先计算块地址，然后读出指定的块，读出的数据存于卡缓冲区 sd\_buf[]中，读块函数的返回值保存于 pRec[1]中。其它命令分支，如写卡、擦卡的操作和读命令分支的程序执行原理也一样。

程序清单 1.17(8) 各分支的执行结果通过串口中间件发送到 PC 机。

5. UART0 接收数据任务。如程序清单 1.18(1)所示，该任务调用串口中间件的接收字节函数 UART0Getch()等待来自 PC 机的数据，如果收到数据，那么从串口的接收缓冲区读出 8 个字节，读出的数据保存于 Buf[]中。然后将 Buf[]的地址放入邮箱 Uart0ReviceMbox 中（如程序清单 1.18(2)所示）通知卡操作任务 TaskCard()：已收到数据了，数据存放在 Buf[]中。这样卡操作任务就可以根据收到的数据进行相关

操作了。

程序清单 1.18 UART0 接收数据任务

```
void TaskUart0Revice(void *pdata)
{
    uint8 Buf[4],i;

    pdata = pdata; /* 避免编译警告 */
    for (;;)
    {
        Buf[0] = UART0Getch(); /* 接收数据头 */ (1)
        for (i = 1; i < 8; i++)
            Buf[i] = UART0Getch();
        OSMboxPost(Uart0ReviceMbox, (void *)Buf); (2)
    }
}
```

在以上例子中，只有几个函数与卡操作相关，大部分语句都在处理 PC 机与 LPC2103 之间的数据通信，以协调对卡的各种操作。本例子不仅给出了对 SD/MMC 读写模块 API 函数的使用方法，还示例了如何使用多个模块或中间件来组建一个工程。

## 1.7 SD/MMC 软件包应用总结

ZLG/SD 的读/写软件包完整开发思想就介绍到此，用户可以在实际应用中充分利用本软件包，采用直接调用 API 应用函数的方法实现读、写、擦除 SD/MMC 卡等操作。

# 嵌入式资源免费下载

## 总线协议:

1. [基于 PCIe 驱动程序的数据传输卡 DMA 传输](#)
2. [基于 PCIe 总线协议的设备驱动开发](#)
3. [CANopen 协议介绍](#)
4. [基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计](#)
5. [FPGA 实现 PCIe 总线 DMA 设计](#)
6. [PCI Express 协议实现与验证](#)
7. [VPX 总线技术及其实现](#)
8. [基于 Xilinx FPGA 的 PCIE 接口实现](#)
9. [基于 PCI 总线的 GPS 授时卡设计](#)
10. [基于 CPCI 标准的 6U 信号处理平台的设计](#)
11. [USB30 电路保护](#)
12. [USB30 协议分析与框架设计](#)
13. [USB 30 中的 CRC 校验原理及实现](#)
14. [基于 CPLD 的 UART 设计](#)
15. [IPMI 在 VPX 系统中的应用与设计](#)
16. [基于 CPCI 总线的 PMC 载板设计](#)
17. [基于 VPX 总线的工件台运动控制系统研究与开发](#)
18. [PCI Express 流控机制的研究与实现](#)
19. [UART16C554 的设计](#)
20. [基于 VPX 的高性能计算机设计](#)
21. [基于 CAN 总线技术的嵌入式网关设计](#)
22. [Visual C 串行通讯控件使用方法与技巧的研究](#)
23. [IEEE1588 精密时钟同步关键技术研究](#)
24. [GPS 信号发生器射频模块的一种实现方案](#)
25. [基于 CPCI 接口的视频采集卡的设计](#)
26. [基于 VPX 的 3U 信号处理平台的设计](#)
27. [基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计](#)
28. [AT89C52 单片机与 ARINC429 航空总线接口设计](#)
29. [基于 CPCI 总线多 DSP 系统的高速主机接口设计](#)
30. [总线协议中的 CRC 及其在 SATA 通信技术中的应用](#)
31. [基于 FPGA 的 SATA 硬盘加解密控制器设计](#)
32. [Modbus 协议在串口通讯中的研究及应用](#)
33. [高可用的磁盘阵列 Cache 的设计和实现](#)
34. [RAID 阵列中高速 Cache 管理的优化](#)

35. [一种新的基于 RAID 的 CACHE 技术研究与实现](#)
36. [基于 PCIE-104 总线的高速数据接口设计](#)
37. [基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计](#)
38. [北斗卫星系统在海洋工程中的应用](#)
39. [北斗卫星系统在远洋船舶上应用的研究](#)
40. [基于 CPCI 总线的红外实时信号处理系统](#)
41. [硬件实现 RAID 与软件实现 RAID 的比较](#)
42. [基于 PCI Express 总线系统的热插拔设计](#)
43. [基于 RAID5 的磁盘阵列 Cache 的研究与实现](#)
44. [基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发](#)
45. [基于磁盘阵列引擎的 RAID5 小写性能优化](#)
46. [基于 IEEE1588 的时钟同步技术研究](#)
47. [基于 Davinci 平台的 SD 卡读写优化](#)
48. [基于 PCI 总线的图像处理及传输系统的设计](#)
49. [串口和以太网通信技术在油液在线监测系统中的应用](#)
50. [USB30 数据传输协议分析及实现](#)
51. [IEEE 1588 协议在工业以太网中的实现](#)
52. [基于 USB30 的设备自定义请求实现方法](#)
53. [IEEE1588 协议在网络测控系统中的应用](#)
54. [USB30 物理层中弹性缓冲的设计与实现](#)
55. [USB30 的高速信息传输瓶颈研究](#)
56. [基于 IPv6 的 UDP 通信的实现](#)
57. [一种基于 IPv6 的流媒体传送方案研究与实现](#)
58. [基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现](#)
59. [RS485CAN 网关设计与实现](#)
60. [MVB 周期信息的实时调度](#)
61. [RS485 和 PROFINET 网关设计](#)
62. [基于 IPv6 的 Socket 通信的实现](#)
63. [MVB 网络重复器的设计](#)
64. [一种新型 MVB 通信板的探究](#)
65. [具有 MVB 接口的输入输出设备的分析](#)
66. [基于 STM32 的 GSM 模块综合应用](#)
67. [基于 ARM7 的 MVB CAN 网关设计](#)
68. [机车车辆的 MVB CAN 总线网关设计](#)
69. [智能变电站冗余网络中 IEEE1588 协议的应用](#)
70. [CAN 总线的浅析 CANopen 协议](#)
71. [基于 CANopen 协议实现多电机系统实时控制](#)
72. [以太网时钟同步协议的研究](#)
73. [基于 CANopen 的列车通信网络实现研究](#)
74. [基于 SJA1000 的 CAN 总线智能控制系统设计](#)
75. [基于 CANopen 的运动控制单元的设计](#)
76. [基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现](#)

77. [分布式控制系统精确时钟同步技术](#)
78. [基于 IEEE 1588 的时钟同步技术在分布式系统中应用](#)
79. [基于 SJA1000 的 CAN 总线通讯模块的实现](#)
80. [嵌入式设备的精确时钟同步技术的研究与实现](#)
81. [基于 SJA1000 的 CAN 网桥设计](#)
82. [基于 CAN 总线分布式温室监控系统的设计与实现](#)
83. [基于 DSP 的 CANopen 通讯协议的实现](#)
84. [基于 PCI9656 控制芯片的高速网卡 DMA 设计](#)
85. [基于以太网及串口的数据采集模块设计](#)
86. [MVB1 类设备控制器的 FPGA 设计](#)
87. [MVB 接口彩色液晶显示诊断单元的显示应用软件设计](#)
88. [IPv6 新型套接字的网络编程剖析](#)
89. [基于规则的 IPv4 源程序到 IPv6 源程序的移植方法](#)
90. [MVB 网络接口单元的 SOC 解决方案](#)
91. [基于 IPSec 协议的 IPv6 安全研究](#)

## VxWorks:

1. [基于 VxWorks 的多任务程序设计](#)
2. [基于 VxWorks 的数据采集存储装置设计](#)
3. [Flash 文件系统分析及其在 VxWorks 中的实现](#)
4. [VxWorks 多任务编程中的异常研究](#)
5. [VxWorks 应用技巧两例](#)
6. [一种基于 VxWorks 的飞行仿真实时管理系统](#)
7. [在 VxWorks 系统中使用 TrueType 字库](#)
8. [基于 FreeType 的 VxWorks 中文显示方案](#)
9. [基于 Tilcon 的 VxWorks 简单动画开发](#)
10. [基于 Tilcon 的某武器显控系统界面设计](#)
11. [基于 Tilcon 的综合导航信息处理装置界面设计](#)
12. [VxWorks 的内存配置和管理](#)
13. [基于 VxWorks 系统的 PCI 配置与应用](#)
14. [基于 MPC8270 的 VxWorks BSP 的移植](#)
15. [Bootrom 功能改进经验谈](#)
16. [基于 VxWorks 嵌入式系统的中文平台研究与实现](#)
17. [VxBus 的 A429 接口驱动](#)
18. [基于 VxBus 和 MPC8569E 千兆网驱动开发和实现](#)
19. [一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法](#)
20. [基于 VxBus 的设备驱动开发](#)

21. [基于 VxBus 的驱动程序架构分析](#)
22. [基于 VxBus 的高速数据采集卡驱动程序开发](#)
23. [Vxworks 下的冗余 CAN 通讯模块设计](#)
24. [WindML 工业平台下开发 S1d13506 驱动及显示功能的实现](#)
25. [WindML 中 Mesa 的应用](#)
26. [VxWorks 下图形用户界面开发中双缓冲技术应用](#)
27. [VxWorks 上的一种 GUI 系统的设计与实现](#)
28. [VxWorks 环境下 socket 的实现](#)
29. [VxWorks 的 WindML 图形界面程序的框架分析](#)
30. [VxWorks 实时操作系统及其在 PC104 下以太网编程的应用](#)
31. [实时操作系统任务调度策略的研究与设计](#)
32. [军事指挥系统中 VxWorks 下汉字显示技术](#)
33. [基于 VxWorks 实时控制系统中文交互界面开发平台](#)
34. [基于 VxWorks 操作系统的 WindML 图形操控界面实现方法](#)
35. [基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发](#)
36. [VxWorks 下的多串口卡设计](#)
37. [VxWorks 内存管理机制的研究](#)
38. [T9 输入法在 Tilcon 下的实现](#)
39. [基于 VxWorks 的 WindML 图形界面开发方法](#)
40. [基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现](#)
41. [基于 VxWorks 的通信服务器实时多任务软件设计](#)
42. [基于 VXWORKS 的 RS485MVB 网关的设计与实现](#)
43. [实时操作系统 VxWorks 在微机保护中的应用](#)
44. [基于 VxWorks 的多任务程序设计及通信管理](#)
45. [基于 Tilcon 的 VxWorks 图形界面开发技术](#)
46. [嵌入式图形系统 Tilcon 及应用研究](#)
47. [基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现](#)
48. [基于嵌入式的 Tilcon 用户图形界面设计与开发](#)
49. [基于 Tilcon 的交互式多页面的设计](#)
50. [基于 Tilcon 的嵌入式系统人机界面开发技术](#)
51. [基于 Tilcon 的指控系统多任务人机交互软件设计](#)
52. [基于 Tilcon 航海标绘台界面设计](#)
53. [基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发](#)
54. [VxWorks 环境下内存文件系统的应用](#)
55. [VxWorks 下的多重定时器设计](#)
56. [Freescale 的 MPC8641D 的 VxWorks BSP](#)
57. [VxWorks 实验五\[时间片轮转调度\]](#)
58. [解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题](#)
59. [基于 VxWorks 系统的 MiniGUI 图形界面开发](#)
60. [VxWorks BSP 开发中的 PCI 配置方法](#)
61. [VxWorks 在 S3C2410 上的 BSP 设计](#)
62. [VxWorks 操作系统中 PCI 总线驱动程序的设计与实现](#)



63. [VxWorks 概述](#)
64. [基于 AT91RM9200 的 VxWorks END 网络驱动开发](#)
65. [基于 EBD9200 的 VxWorks BSP 设计和实现](#)
66. [基于 VxWorks 的 BSP 技术分析](#)
67. [ARM LPC2210 的 VxWorks BSP 源码](#)
68. [基于 LPC2210 的 VxWorks BSP 移植](#)
69. [基于 VxWorks 平台的 SCTP 协议软件设计实现](#)
70. [VxWorks 快速启动的实现方法\[上电到应用程序 1 秒\]](#)

## Linux:

1. [Linux 程序设计第三版及源代码](#)
2. [NAND FLASH 文件系统的设计与实现](#)
3. [多通道串行通信设备的 Linux 驱动程序实现](#)
4. [Zsh 开发指南-数组](#)
5. [常用 GDB 命令中文速览](#)
6. [嵌入式 C 进阶之道](#)
7. [Linux 串口编程实例](#)
8. [基于 Yocto Project 的嵌入式应用设计](#)
9. [Android 应用的反编译](#)
10. [基于 Android 行为的加密应用系统研究](#)
11. [嵌入式 Linux 系统移植步步通](#)
12. [嵌入式 C++ 语言精华文章集锦](#)
13. [基于 Linux 的高性能服务器端的设计与研究](#)
14. [S3C6410 移植 Android 内核](#)
15. [Android 开发指南中文版](#)
16. [图解 Linux 操作系统架构设计与实现原理（第二版）](#)
17. [如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核](#)
18. [Android 简单 mp3 播放器源码](#)
19. [嵌入式 Linux 系统实时性的研究](#)
20. [Android 嵌入式系统架构及内核浅析](#)
21. [基于嵌入式 Linux 操作系统内核实时性的改进方法研究](#)
22. [Linux TCP IP 协议详解](#)
23. [Linux 桌面环境下内存去重技术的研究与实现](#)
24. [掌握 Android 7.0 新增特性 Quick Settings](#)
25. [Android 应用逆向分析方法研究](#)
26. [Android 操作系统的课程教学](#)
27. [Android 智能手机操作系统的研究](#)
28. [Android 英文朗读功能的实现](#)

29. [基于 Yocto 订制嵌入式 Linux 发行版](#)
30. [基于嵌入式 Linux 的网络设备驱动设计与实现](#)
31. [如何高效学习嵌入式](#)
32. [基于 Android 平台的 GPS 定位系统的设计与实现](#)
33. [LINUX ARM 下的 USB 驱动开发](#)
34. [Linux 下基于 I2C 协议的 RTC 驱动开发](#)
35. [嵌入式下 Linux 系统设备驱动程序的开发](#)
36. [基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现](#)
37. [Linux 系统中进程调度策略](#)
38. [嵌入式 Linux 实时性方法](#)
39. [基于实时 Linux 计算机联锁系统实时性分析与改进](#)
40. [基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究](#)
41. [Android 手机应用开发之音乐资源播放器](#)
42. [Linux 下以太网的 IPv6 隧道技术的实现](#)
43. [Research and design of mobile learning platform based on Android](#)
44. [基于 linux 和 Qt 的串口通信调试器调的设计及应用](#)
45. [在 Linux 平台上基于 QT 的动态图像采集系统的设计](#)
46. [基于 Android 平台的医护查房系统的研究与设计](#)
47. [基于 Android 平台的软件自动化监控工具的设计开发](#)
48. [基于 Android 的视频软硬解码及渲染的对比研究与实现](#)
49. [基于 Android 移动设备的加速度传感器技术研究](#)
50. [基于 Android 系统振动测试仪研究](#)
51. [基于缓存竞争优化的 Linux 进程调度策略](#)
52. [Linux 基于 W83697 和 W83977 的 UART 串口驱动开发文档](#)
53. [基于 AT91RM9200 的嵌入式 Linux 系统的移植与实现](#)
54. [路由信息协议在 Linux 平台上的实现](#)
55. [Linux 下 IPv6 高级路由器的实现](#)

## Windows CE:

1. [Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计](#)
2. [Windows CE 的 CAN 总线驱动程序设计](#)
3. [基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究](#)
4. [基于 Windows CE.NET 平台的串行通信实现](#)
5. [基于 Windows CE.NET 下的 GPRS 模块的研究与开发](#)
6. [win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码](#)
7. [Windows 下的 USB 设备驱动程序开发](#)
8. [WinCE 的大容量程控数据传输解决方案设计](#)
9. [WinCE6.0 安装开发详解](#)

- [10. DOS 下仿 Windows 的自带计算器程序 C 源码](#)
- [11. G726 局域网语音通话程序和源代码](#)
- [12. WinCE 主板加载第三方驱动程序的方法](#)
- [13. WinCE 下的注册表编辑程序和源代码](#)
- [14. WinCE 串口通信源代码](#)
- [15. WINCE 的 SD 卡程序\[可实现读写的源码\]](#)
- [16. 基于 WinCE 的 BootLoader 研究](#)
- [17. Windows CE 环境下无线网卡的自动安装](#)
- [18. 基于 Windows CE 的可视电话的研究与实现](#)
- [19. 基于 WinCE 的嵌入式图像采集系统设计](#)
- [20. 基于 ARM 与 WinCE 的掌纹鉴别系统](#)
- [21. DCOM 协议在网络冗余环境下的应用](#)
- [22. Windows XP Embedded 在变电站通信管理机中的应用](#)
- [23. XPE 在多功能显控台上的开发与应用](#)
- [24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现](#)
- [25. 虚拟仪器的 Windows XP Embedded 操作系统开发](#)
- [26. 基于 EVC 的嵌入式导航电子地图设计](#)
- [27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现](#)
- [28. 基于 XPE 的数字残币兑换工具开发](#)
- [29. Windows CENET 下 ADC 驱动开发设计](#)
- [30. Windows CE 下 USB 设备流驱动开发与设计](#)
- [31. Windows 驱动程序设计](#)
- [32. 基于 Windows CE 的 GPS 应用](#)
- [33. 基于 Windows CE 下大像素图像分块显示算法的研究](#)
- [34. 基于 Windows CE 的数控软件开发与实现](#)

## PowerPC:

- [1. Freescale MPC8536 开发板原理图](#)
- [2. 基于 MPC8548E 的固件设计](#)
- [3. 基于 MPC8548E 的嵌入式数据处理系统设计](#)
- [4. 基于 PowerPC 嵌入式网络通信平台的实现](#)
- [5. PowerPC 在车辆显控系统中的应用](#)
- [6. 基于 PowerPC 的单板计算机的设计](#)
- [7. 用 PowerPC860 实现 FPGA 配置](#)
- [8. 基于 MPC8247 嵌入式电力交换系统的设计与实现](#)
- [9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发](#)
- [10. 基于 MPC8313E 嵌入式系统 UBoot 的移植](#)

11. [基于 PowerPC 处理器 SMP 系统的 UBoot 移植](#)
12. [基于 PowerPC 双核处理器嵌入式系统 UBoot 移植](#)
13. [基于 PowerPC 的雷达通用处理机设计](#)
14. [PowerPC 平台引导加载程序的移植](#)
15. [基于 PowerPC 嵌入式内核的多串口通信扩展设计](#)
16. [基于 PowerPC 的多网口系统抗干扰设计](#)
17. [基于 MPC860T 与 VxWorks 的图形界面设计](#)
18. [基于 MPC8260 处理器的 PPMC 系统](#)
19. [基于 PowerPC 的控制器研究与设计](#)
20. [基于 PowerPC 的模拟量输入接口扩展](#)
21. [基于 PowerPC 的车载通信系统设计](#)
22. [基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法](#)
23. [基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究](#)
24. [基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计](#)
25. [基于 PowerPC603e 通用处理模块的设计与实现](#)
26. [嵌入式微机 MPC555 驻留片内监控器的开发与实现](#)
27. [基于 PowerPC 和 DSP 的电能质量在线监测装置的研制](#)
28. [基于 PowerPC 架构多核处理器嵌入式系统硬件设计](#)
29. [基于 PowerPC 的多屏系统设计](#)
30. [基于 PowerPC 的嵌入式 SMP 系统设计](#)
31. [基于 MPC850 的多功能通信管理器](#)
32. [基于 MPC8640D 处理系统的技术研究](#)
33. [基于双核 MPC8641D 处理器的计算机模块设计](#)

## ARM:

1. [基于 DiskOnChip 2000 的驱动程序设计及应用](#)
2. [基于 ARM 体系的 PC-104 总线设计](#)
3. [基于 ARM 的嵌入式系统中断处理机制研究](#)
4. [设计 ARM 的中断处理](#)
5. [基于 ARM 的数据采集系统并行总线的驱动设计](#)
6. [S3C2410 下的 TFT LCD 驱动源码](#)
7. [STM32 SD 卡移植 FATFS 文件系统源码](#)
8. [STM32 ADC 多通道源码](#)
9. [ARM Linux 在 EP7312 上的移植](#)
10. [ARM 经典 300 问](#)
11. [基于 S5PV210 的频谱监测设备嵌入式系统设计与实现](#)
12. [Uboot 中 start.S 源码的指令级的详尽解析](#)

13. [基于 ARM9 的嵌入式 Zigbee 网关设计与实现](#)
14. [基于 S3C6410 处理器的嵌入式 Linux 系统移植](#)
15. [CortexA8 平台的  \$\mu\$ C-OS II 及 LwIP 协议栈的移植与实现](#)
16. [基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计](#)
17. [ARM S3C2440 Linux ADC 驱动](#)
18. [ARM S3C2440 Linux 触摸屏驱动](#)
19. [Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计](#)
20. [Nand Flash 启动模式下的 Uboot 移植](#)
21. [基于 ARM 处理器的 UART 设计](#)
22. [ARM CortexM3 处理器故障的分析与处理](#)
23. [ARM 微处理器启动和调试浅析](#)
24. [基于 ARM 系统下映像文件的执行与中断运行机制的实现](#)
25. [中断调用方式的 ARM 二次开发接口设计](#)
26. [ARM11 嵌入式系统 Linux 下 LCD 的驱动设计](#)
27. [Uboot 在 S3C2440 上的移植](#)
28. [基于 ARM11 的嵌入式无线视频终端的设计](#)
29. [基于 S3C6410 的 Uboot 分析与移植](#)
30. [基于 ARM 嵌入式系统的高保真无损音乐播放器设计](#)
31. [UBoot 在 Mini6410 上的移植](#)
32. [基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现](#)
33. [基于 ARM11 的电源完整性分析](#)
34. [基于 ARM S3C6410 的 uboot 分析与移植](#)
35. [基于 S5PC100 移动视频监控终端的设计与实现](#)
36. [UBoot 在 AT91RM9200 上的移植简析](#)
37. [基于工控级 AT91RM9200 开发板的 UBoot 移植分析](#)

## Hardware:

1. [DSP 电源的典型设计](#)
2. [高频脉冲电源设计](#)
3. [电源的综合保护设计](#)
4. [任意波形电源的设计](#)
5. [高速 PCB 信号完整性分析及应用](#)
6. [DM642 高速图像采集系统的电磁干扰设计](#)
7. [使用 COMExpress Nano 工控板实现 IP 调度设备](#)
8. [基于 COM Express 架构的数据记录仪的设计与实现](#)
9. [基于 COM Express 的信号系统逻辑运算单元设计](#)
10. [基于 COM Express 的回波预处理模块设计](#)
11. [基于 X86 平台的简单多任务内核的分析与实现](#)

12. [基于 UEFI Shell 的 PreOS Application 的开发与研究](#)
13. [基于 UEFI 固件的恶意代码防范技术研究](#)
14. [MIPS 架构计算机平台的支持固件研究](#)
15. [基于 UEFI 固件的攻击验证技术研究](#)
16. [基于 UEFI 的 Application 和 Driver 的分析与开发](#)
17. [基于 UEFI 的可信 BIOS 研究与实现](#)
18. [基于 UEFI 的国产计算机平台 BIOS 研究](#)
19. [基于 UEFI 的安全模块设计分析](#)
20. [基于 FPGA Nios II 的等精度频率计设计](#)
21. [基于 FPGA 的 SOPC 设计](#)
22. [基于 SOPC 基本信号产生器的设计与实现](#)
23. [基于龙芯平台的 PMON 研究与开发](#)
24. [基于 X86 平台的嵌入式 BIOS 可配置设计](#)
25. [基于龙芯 2F 架构的 PMON 分析与优化](#)
26. [CPU 与 GPU 之间接口电路的设计与实现](#)
27. [基于龙芯 1A 平台的 PMON 源码编译和启动分析](#)
28. [基于 PC104 工控机的嵌入式直流监控装置的设计](#)
29. [GPGPU 技术研究与实现](#)
30. [GPU 实现的高速 FIR 数字滤波算法](#)
31. [一种基于 CPU/GPU 异构计算的混合编程模型](#)
32. [面向 OpenCL 模型的 GPU 性能优化](#)
33. [基于 GPU 的 FDTD 算法](#)
34. [基于 GPU 的瑕疵检测](#)
35. [基于 GPU 通用计算的分析与研究](#)
36. [面向 OpenCL 架构的 GPGPU 量化性能模型](#)
37. [基于 OpenCL 的图像积分图算法优化研究](#)
38. [基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究](#)
39. [基于 OpenCL 的异构系统并行编程](#)
40. [嵌入式系统中热备份双机切换技术研究](#)
41. [EFI-Tiano 环境下的 AES 算法应用模型](#)

## Programming:

1. [计算机软件基础数据结构 - 算法](#)
2. [高级数据结构对算法的优化](#)
3. [零基础学算法](#)
4. [Linux 环境下基于 TCP 的 Socket 编程浅析](#)
5. [Linux 环境下基于 UDP 的 socket 编程浅析](#)

6. [基于 Socket 的网络编程技术及其实现](#)
7. [数据结构考题 - 第 1 章 绪论](#)
8. [数据结构考题 - 第 2 章 线性表](#)
9. [数据结构考题 - 第 2 章 线性表 - 答案](#)
10. [基于小波变换与偏微分方程的图像分解及边缘检测](#)
11. [基于图像能量的布匹瑕疵检测方法](#)
12. [基于 OpenCL 的拉普拉斯图像增强算法优化研究](#)
13. [异构平台上基于 OpenCL 的 FFT 实现与优化](#)
14. [数据结构考题 - 第 4 章 串](#)
15. [数据结构考题 - 第 4 章 串答案](#)
16. [用 IPv6 编程接口实现有连接通信的方法](#)

## FPGA / CPLD:

1. [一种基于并行处理器的快速车道线检测系统及 FPGA 实现](#)
2. [基于 FPGA 和 DSP 的 DBF 实现](#)
3. [高速浮点运算单元的 FPGA 实现](#)
4. [DLMS 算法的脉动阵结构设计及 FPGA 实现](#)
5. [一种基于 FPGA 的 3DES 加密算法实现](#)
6. [可编程 FIR 滤波器的 FPGA 实现](#)
7. [基于 FPGA 的 AES 加密算法的高速实现](#)
8. [基于 FPGA 的精确时钟同步方法](#)
9. [应用分布式算法在 FPGA 平台实现 FIR 低通滤波器](#)
10. [流水线技术在用 FPGA 实现高速 DSP 运算中的应用](#)
11. [基于 FPGA 的 CAN 总线通信节点设计](#)
12. [基于 FPGA 的高速时钟数据恢复电路的实现](#)
13. [基于 FPGA 的高阶高速 FIR 滤波器设计与实现](#)
14. [基于 FPGA 高效实现 FIR 滤波器的研究](#)
- 15.