

主要包含4个C文件和和三个头文件

CH376INC.H 与CH376HFT.C为u盘主文件

SPI\_SW.C 为对硬件操作文件 单片机与SPI接口

FILE\_SYS.H与FILE\_SYS.C 为ch376接口函数

分别列出

CH376INC.H文件：

/\* C Define for CH376 \*/

/\* Website: http://wch.cn \*/

/\* Email: tech@wch.cn \*/

/\* Author: W.ch 2008.10 \*/

/\* V1.0 for CH376 \*/

#ifndef \_\_CH376INC\_H\_\_

#define \_\_CH376INC\_H\_\_

#ifdef \_\_cplusplus

extern "C" {

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 常用类型和常量定义 \*/

#ifndef TRUE

#define TRUE 1

#define FALSE 0

#endif

#ifndef NULL

#define NULL 0

#endif

#ifndef UINT8

typedef unsigned char UINT8;

#endif

#ifndef UINT16

typedef unsigned short UINT16;

#endif

#ifndef UINT32

typedef unsigned long UINT32;

#endif

#ifndef PUINT8

typedef unsigned char \*PUINT8;

#endif

#ifndef PUINT16

typedef unsigned short \*PUINT16;

#endif

#ifndef PUINT32

typedef unsigned long \*PUINT32;

#endif

#ifndef UINT8V

typedef unsigned char volatile UINT8V;

#endif

#ifndef PUINT8V

typedef unsigned char volatile \*PUINT8V;

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 硬件特性 \*/

#define CH376\_DAT\_BLOCK\_LEN 0x40 /\* USB单个数据包, 数据块的最大长度, 默认缓冲区的长度 \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 命令代码 \*/

/\* 部分命令兼容CH375芯片, 但是输入数据或者输出数据的可能局部不同) \*/

/\* 一个命令操作顺序包含:

一个命令码(对于串口方式,命令码之前还需要两个同步码),

若干个输入数据(可以是0个),

产生中断通知 或者 若干个输出数据(可以是0个), 二选一, 有中断通知则一定没有输出数据, 有输出数据则一定不产生中断

仅CMD01\_WR\_REQ\_DATA命令例外, 顺序包含: 一个命令码, 一个输出数据, 若干个输入数据

命令码起名规则: CMDxy\_NAME

其中的x和y都是数字, x说明最少输入数据个数(字节数), y说明最少输出数据个数(字节数), y如果是H则说明产生中断通知,

有些命令能够实现0到多个字节的数据块读写, 数据块本身的字节数未包含在上述x或y之内 \*/

/\* 本文件默认会同时提供与CH375芯片命令码兼容的命令码格式（即去掉x和y之后), 如果不需要, 那么可以定义\_NO\_CH375\_COMPATIBLE\_禁止 \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 主要命令(手册一), 常用 \*/

#define CMD01\_GET\_IC\_VER 0x01 /\* 获取芯片及固件版本 \*/

/\* 输出: 版本号( 位7为0, 位6为1, 位5~位0为版本号 ) \*/

/\* CH376返回版本号的值为041H即版本号为01H \*/

#define CMD21\_SET\_BAUDRATE 0x02 /\* 串口方式: 设置串口通讯波特率(上电或者复位后的默认波特率为9600bps,由D4/D5/D6引脚选择) \*/

/\* 输入: 波特率分频系数, 波特率分频常数 \*/

/\* 输出: 操作状态( CMD\_RET\_SUCCESS或CMD\_RET\_ABORT, 其它值说明操作未完成 ) \*/

#define CMD00\_ENTER\_SLEEP 0x03 /\* 进入睡眠状态 \*/

#define CMD00\_RESET\_ALL 0x05 /\* 执行硬件复位 \*/

#define CMD11\_CHECK\_EXIST 0x06 /\* 测试通讯接口和工作状态 \*/

/\* 输入: 任意数据 \*/

/\* 输出: 输入数据的按位取反 \*/

#define CMD20\_CHK\_SUSPEND 0x0B /\* 设备方式: 设置检查USB总线挂起状态的方式 \*/

/\* 输入: 数据10H, 检查方式 \*/

/\* 00H=不检查USB挂起, 04H=以50mS为间隔检查USB挂起, 05H=以10mS为间隔检查USB挂起 \*/

#define CMD20\_SET\_SDO\_INT 0x0B /\* SPI接口方式: 设置SPI的SDO引脚的中断方式 \*/

/\* 输入: 数据16H, 中断方式 \*/

/\* 10H=禁止SDO引脚用于中断输出,在SCS片选无效时三态输出禁止, 90H=SDO引脚在SCS片选无效时兼做中断请求输出 \*/

#define CMD14\_GET\_FILE\_SIZE 0x0C /\* 主机文件模式: 获取当前文件长度 \*/

/\* 输入: 数据68H \*/

/\* 输出: 当前文件长度(总长度32位,低字节在前) \*/

#define CMD50\_SET\_FILE\_SIZE 0x0D /\* 主机文件模式: 设置当前文件长度 \*/

/\* 输入: 数据68H, 当前文件长度(总长度32位,低字节在前) \*/

#define CMD11\_SET\_USB\_MODE 0x15 /\* 设置USB工作模式 \*/

/\* 输入: 模式代码 \*/

/\* 00H=未启用的设备方式, 01H=已启用的设备方式并且使用外部固件模式(串口不支持), 02H=已启用的设备方式并且使用内置固件模式 \*/

/\* 03H=SD卡主机模式/未启用的主机模式,用于管理和存取SD卡中的文件 \*/

/\* 04H=未启用的主机方式, 05H=已启用的主机方式, 06H=已启用的主机方式并且自动产生SOF包, 07H=已启用的主机方式并且复位USB总线 \*/

/\* 输出: 操作状态( CMD\_RET\_SUCCESS或CMD\_RET\_ABORT, 其它值说明操作未完成 ) \*/

#define CMD01\_GET\_STATUS 0x22 /\* 获取中断状态并取消中断请求 \*/

/\* 输出: 中断状态 \*/

#define CMD00\_UNLOCK\_USB 0x23 /\* 设备方式: 释放当前USB缓冲区 \*/

#define CMD01\_RD\_USB\_DATA0 0x27 /\* 从当前USB中断的端点缓冲区或者主机端点的接收缓冲区读取数据块 \*/

/\* 输出: 长度, 数据流 \*/

#define CMD01\_RD\_USB\_DATA 0x28 /\* 设备方式: 从当前USB中断的端点缓冲区读取数据块, 并释放缓冲区, 相当于 CMD01\_RD\_USB\_DATA0 + CMD00\_UNLOCK\_USB \*/

/\* 输出: 长度, 数据流 \*/

#define CMD10\_WR\_USB\_DATA7 0x2B /\* 设备方式: 向USB端点2的发送缓冲区写入数据块 \*/

/\* 输入: 长度, 数据流 \*/

#define CMD10\_WR\_HOST\_DATA 0x2C /\* 向USB主机端点的发送缓冲区写入数据块 \*/

/\* 输入: 长度, 数据流 \*/

#define CMD01\_WR\_REQ\_DATA 0x2D /\* 向内部指定缓冲区写入请求的数据块 \*/

/\* 输出: 长度 \*/

/\* 输入: 数据流 \*/

#define CMD20\_WR\_OFS\_DATA 0x2E /\* 向内部缓冲区指定偏移地址写入数据块 \*/

/\* 输入: 偏移, 长度, 数据流 \*/

#define CMD10\_SET\_FILE\_NAME 0x2F /\* 主机文件模式: 设置将要操作的文件的文件名 \*/

/\* 输入: 以0结束的字符串(含结束符0在内长度不超过14个字符) \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 主要命令(手册一), 常用, 以下命令总是在操作结束时产生中断通知, 并且总是没有输出数据 \*/

#define CMD0H\_DISK\_CONNECT 0x30 /\* 主机文件模式/不支持SD卡: 检查磁盘是否连接 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_MOUNT 0x31 /\* 主机文件模式: 初始化磁盘并测试磁盘是否就绪 \*/

/\* 输出中断 \*/

#define CMD0H\_FILE\_OPEN 0x32 /\* 主机文件模式: 打开文件或者目录(文件夹),或者枚举文件和目录(文件夹) \*/

/\* 输出中断 \*/

#define CMD0H\_FILE\_ENUM\_GO 0x33 /\* 主机文件模式: 继续枚举文件和目录(文件夹) \*/

/\* 输出中断 \*/

#define CMD0H\_FILE\_CREATE 0x34 /\* 主机文件模式: 新建文件,如果文件已经存在那么先删除 \*/

/\* 输出中断 \*/

#define CMD0H\_FILE\_ERASE 0x35 /\* 主机文件模式: 删除文件,如果已经打开则直接删除,否则对于文件会先打开再删除,子目录必须先打开 \*/

/\* 输出中断 \*/

#define CMD1H\_FILE\_CLOSE 0x36 /\* 主机文件模式: 关闭当前已经打开的文件或者目录(文件夹) \*/

/\* 输入: 是否允许更新文件长度 \*/

/\* 00H=禁止更新长度, 01H=允许更新长度 \*/

/\* 输出中断 \*/

#define CMD1H\_DIR\_INFO\_READ 0x37 /\* 主机文件模式: 读取文件的目录信息 \*/

/\* 输入: 指定需要读取的目录信息结构在扇区内的索引号 \*/

/\* 索引号范围为00H~0FH, 索引号0FFH则为当前已经打开的文件 \*/

/\* 输出中断 \*/

#define CMD0H\_DIR\_INFO\_SAVE 0x38 /\* 主机文件模式: 保存文件的目录信息 \*/

/\* 输出中断 \*/

#define CMD4H\_BYTE\_LOCATE 0x39 /\* 主机文件模式: 以字节为单位移动当前文件指针 \*/

/\* 输入: 偏移字节数(总长度32位,低字节在前) \*/

/\* 输出中断 \*/

#define CMD2H\_BYTE\_READ 0x3A /\* 主机文件模式: 以字节为单位从当前位置读取数据块 \*/

/\* 输入: 请求读取的字节数(总长度16位,低字节在前) \*/

/\* 输出中断 \*/

#define CMD0H\_BYTE\_RD\_GO 0x3B /\* 主机文件模式: 继续字节读 \*/

/\* 输出中断 \*/

#define CMD2H\_BYTE\_WRITE 0x3C /\* 主机文件模式: 以字节为单位向当前位置写入数据块 \*/

/\* 输入: 请求写入的字节数(总长度16位,低字节在前) \*/

/\* 输出中断 \*/

#define CMD0H\_BYTE\_WR\_GO 0x3D /\* 主机文件模式: 继续字节写 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_CAPACITY 0x3E /\* 主机文件模式: 查询磁盘物理容量 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_QUERY 0x3F /\* 主机文件模式: 查询磁盘空间信息 \*/

/\* 输出中断 \*/

#define CMD0H\_DIR\_CREATE 0x40 /\* 主机文件模式: 新建目录(文件夹)并打开,如果目录已经存在那么直接打开 \*/

/\* 输出中断 \*/

#define CMD4H\_SEC\_LOCATE 0x4A /\* 主机文件模式: 以扇区为单位移动当前文件指针 \*/

/\* 输入: 偏移扇区数(总长度32位,低字节在前) \*/

/\* 输出中断 \*/

#define CMD1H\_SEC\_READ 0x4B /\* 主机文件模式/不支持SD卡: 以扇区为单位从当前位置读取数据块 \*/

/\* 输入: 请求读取的扇区数 \*/

/\* 输出中断 \*/

#define CMD1H\_SEC\_WRITE 0x4C /\* 主机文件模式/不支持SD卡: 以扇区为单位在当前位置写入数据块 \*/

/\* 输入: 请求写入的扇区数 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_BOC\_CMD 0x50 /\* 主机方式/不支持SD卡: 对USB存储器执行BulkOnly传输协议的命令 \*/

/\* 输出中断 \*/

#define CMD5H\_DISK\_READ 0x54 /\* 主机方式/不支持SD卡: 从USB存储器读物理扇区 \*/

/\* 输入: LBA物理扇区地址(总长度32位, 低字节在前), 扇区数(01H~FFH) \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_RD\_GO 0x55 /\* 主机方式/不支持SD卡: 继续执行USB存储器的物理扇区读操作 \*/

/\* 输出中断 \*/

#define CMD5H\_DISK\_WRITE 0x56 /\* 主机方式/不支持SD卡: 向USB存储器写物理扇区 \*/

/\* 输入: LBA物理扇区地址(总长度32位, 低字节在前), 扇区数(01H~FFH) \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_WR\_GO 0x57 /\* 主机方式/不支持SD卡: 继续执行USB存储器的物理扇区写操作 \*/

/\* 输出中断 \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 辅助命令(手册二), 不太常用或者是为了与CH375和CH372兼容 \*/

#define CMD10\_SET\_USB\_SPEED 0x04 /\* 设置USB总线速度, 在每次CMD11\_SET\_USB\_MODE设置USB工作模式时会自动恢复到12Mbps全速 \*/

/\* 输入: 总线速度代码 \*/

/\* 00H=12Mbps全速FullSpeed（默认值）, 01H=1.5Mbps（仅修改频率）, 02H=1.5Mbps低速LowSpeed \*/

#define CMD11\_GET\_DEV\_RATE 0x0A /\* 主机方式: 获取当前连接的USB设备的数据速率类型 \*/

/\* 输入: 数据07H \*/

/\* 输出: 数据速率类型 \*/

/\* 位4为1则是1.5Mbps低速USB设备, 否则是12Mbps全速USB设备 \*/

#define CMD11\_GET\_TOGGLE 0x0A /\* 获取OUT事务的同步状态 \*/

/\* 输入: 数据1AH \*/

/\* 输出: 同步状态 \*/

/\* 位4为1则OUT事务同步, 否则OUT事务不同步 \*/

#define CMD11\_READ\_VAR8 0x0A /\* 读取指定的8位文件系统变量 \*/

/\* 输入: 变量地址 \*/

/\* 输出: 数据 \*/

/\*#define CMD11\_GET\_MAX\_LUN = CMD11\_READ\_VAR8( VAR\_UDISK\_LUN )\*/ /\* 主机方式: 获取USB存储器最大和当前逻辑单元号 \*/

#define CMD20\_SET\_RETRY 0x0B /\* 主机方式: 设置USB事务操作的重试次数 \*/

/\* 输入: 数据25H, 重试次数 \*/

/\* 位7为0则收到NAK时不重试, 位7为1位6为0则收到NAK时无限重试, 位7为1位6为1则收到NAK时最多重试3秒, 位5~位0为超时后的重试次数 \*/

#define CMD20\_WRITE\_VAR8 0x0B /\* 设置指定的8位文件系统变量 \*/

/\* 输入: 变量地址, 数据 \*/

/\*#define CMD20\_SET\_DISK\_LUN = CMD20\_WRITE\_VAR8( VAR\_UDISK\_LUN )\*/ /\* 主机方式: 设置USB存储器的当前逻辑单元号 \*/

#define CMD14\_READ\_VAR32 0x0C /\* 读取指定的32位文件系统变量 \*/

/\* 输入: 变量地址 \*/

/\* 输出: 数据(总长度32位,低字节在前) \*/

#define CMD50\_WRITE\_VAR32 0x0D /\* 设置指定的32位文件系统变量 \*/

/\* 输入: 变量地址, 数据(总长度32位,低字节在前) \*/

#define CMD01\_DELAY\_100US 0x0F /\* 延时100uS(串口不支持) \*/

/\* 输出: 延时期间输出0,延时结束输出非0 \*/

#define CMD40\_SET\_USB\_ID 0x12 /\* 设备方式: 设置USB厂商VID和产品PID \*/

/\* 输入: 厂商ID低字节, 厂商ID高字节, 产品ID低字节, 产品ID高字节 \*/

#define CMD10\_SET\_USB\_ADDR 0x13 /\* 设置USB地址 \*/

/\* 输入: 地址值 \*/

#define CMD01\_TEST\_CONNECT 0x16 /\* 主机方式/不支持SD卡: 检查USB设备连接状态 \*/

/\* 输出: 状态( USB\_INT\_CONNECT或USB\_INT\_DISCONNECT或USB\_INT\_USB\_READY, 其它值说明操作未完成 ) \*/

#define CMD00\_ABORT\_NAK 0x17 /\* 主机方式: 放弃当前NAK的重试 \*/

#define CMD10\_SET\_ENDP2 0x18 /\* 设备方式(串口不支持): 设置USB端点0的接收器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000-就绪ACK, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD10\_SET\_ENDP3 0x19 /\* 设备方式(串口不支持): 设置USB端点0的发送器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000~1000-就绪ACK, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD10\_SET\_ENDP4 0x1A /\* 设备方式(串口不支持): 设置USB端点1的接收器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000-就绪ACK, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD10\_SET\_ENDP5 0x1B /\* 设备方式(串口不支持): 设置USB端点1的发送器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000~1000-就绪ACK, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD10\_SET\_ENDP6 0x1C /\* 设置USB端点2/主机端点的接收器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000-就绪ACK, 1101-就绪但不返回ACK, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD10\_SET\_ENDP7 0x1D /\* 设置USB端点2/主机端点的发送器 \*/

/\* 输入: 工作方式 \*/

/\* 位7为1则位6为同步触发位, 否则同步触发位不变 \*/

/\* 位3~位0为事务响应方式: 0000-就绪ACK, 1101-就绪但无须应答, 1110-正忙NAK, 1111-错误STALL \*/

#define CMD00\_DIRTY\_BUFFER 0x25 /\* 主机文件模式: 清除内部的磁盘和文件缓冲区 \*/

#define CMD10\_WR\_USB\_DATA3 0x29 /\* 设备方式(串口不支持): 向USB端点0的发送缓冲区写入数据块 \*/

/\* 输入: 长度, 数据流 \*/

#define CMD10\_WR\_USB\_DATA5 0x2A /\* 设备方式(串口不支持): 向USB端点1的发送缓冲区写入数据块 \*/

/\* 输入: 长度, 数据流 \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 辅助命令(手册二), 不太常用或者是为了与CH375和CH372兼容, 以下命令总是在操作结束时产生中断通知, 并且总是没有输出数据 \*/

#define CMD1H\_CLR\_STALL 0x41 /\* 主机方式: 控制传输-清除端点错误 \*/

/\* 输入: 端点号 \*/

/\* 输出中断 \*/

#define CMD1H\_SET\_ADDRESS 0x45 /\* 主机方式: 控制传输-设置USB地址 \*/

/\* 输入: 地址值 \*/

/\* 输出中断 \*/

#define CMD1H\_GET\_DESCR 0x46 /\* 主机方式: 控制传输-获取描述符 \*/

/\* 输入: 描述符类型 \*/

/\* 输出中断 \*/

#define CMD1H\_SET\_CONFIG 0x49 /\* 主机方式: 控制传输-设置USB配置 \*/

/\* 输入: 配置值 \*/

/\* 输出中断 \*/

#define CMD0H\_AUTO\_SETUP 0x4D /\* 主机方式/不支持SD卡: 自动配置USB设备 \*/

/\* 输出中断 \*/

#define CMD2H\_ISSUE\_TKN\_X 0x4E /\* 主机方式: 发出同步令牌, 执行事务, 该命令可代替 CMD10\_SET\_ENDP6/CMD10\_SET\_ENDP7 + CMD1H\_ISSUE\_TOKEN \*/

/\* 输入: 同步标志, 事务属性 \*/

/\* 同步标志的位7为主机端点IN的同步触发位, 位6为主机端点OUT的同步触发位, 位5~位0必须为0 \*/

/\* 事务属性的低4位是令牌, 高4位是端点号 \*/

/\* 输出中断 \*/

#define CMD1H\_ISSUE\_TOKEN 0x4F /\* 主机方式: 发出令牌, 执行事务, 建议用CMD2H\_ISSUE\_TKN\_X命令 \*/

/\* 输入: 事务属性 \*/

/\* 低4位是令牌, 高4位是端点号 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_INIT 0x51 /\* 主机方式/不支持SD卡: 初始化USB存储器 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_RESET 0x52 /\* 主机方式/不支持SD卡: 控制传输-复位USB存储器 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_SIZE 0x53 /\* 主机方式/不支持SD卡: 获取USB存储器的容量 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_INQUIRY 0x58 /\* 主机方式/不支持SD卡: 查询USB存储器特性 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_READY 0x59 /\* 主机方式/不支持SD卡: 检查USB存储器就绪 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_R\_SENSE 0x5A /\* 主机方式/不支持SD卡: 检查USB存储器错误 \*/

/\* 输出中断 \*/

#define CMD0H\_RD\_DISK\_SEC 0x5B /\* 主机文件模式: 从磁盘读取一个扇区的数据到内部缓冲区 \*/

/\* 输出中断 \*/

#define CMD0H\_WR\_DISK\_SEC 0x5C /\* 主机文件模式: 将内部缓冲区的一个扇区的数据写入磁盘 \*/

/\* 输出中断 \*/

#define CMD0H\_DISK\_MAX\_LUN 0x5D /\* 主机方式: 控制传输-获取USB存储器最大逻辑单元号 \*/

/\* 输出中断 \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 以下定义只是为了兼容CH375的INCLUDE文件中的命令名称格式 \*/

#ifndef \_NO\_CH375\_COMPATIBLE\_

#define CMD\_GET\_IC\_VER CMD01\_GET\_IC\_VER

#define CMD\_SET\_BAUDRATE CMD21\_SET\_BAUDRATE

#define CMD\_ENTER\_SLEEP CMD00\_ENTER\_SLEEP

#define CMD\_RESET\_ALL CMD00\_RESET\_ALL

#define CMD\_CHECK\_EXIST CMD11\_CHECK\_EXIST

#define CMD\_CHK\_SUSPEND CMD20\_CHK\_SUSPEND

#define CMD\_SET\_SDO\_INT CMD20\_SET\_SDO\_INT

#define CMD\_GET\_FILE\_SIZE CMD14\_GET\_FILE\_SIZE

#define CMD\_SET\_FILE\_SIZE CMD50\_SET\_FILE\_SIZE

#define CMD\_SET\_USB\_MODE CMD11\_SET\_USB\_MODE

#define CMD\_GET\_STATUS CMD01\_GET\_STATUS

#define CMD\_UNLOCK\_USB CMD00\_UNLOCK\_USB

#define CMD\_RD\_USB\_DATA0 CMD01\_RD\_USB\_DATA0

#define CMD\_RD\_USB\_DATA CMD01\_RD\_USB\_DATA

#define CMD\_WR\_USB\_DATA7 CMD10\_WR\_USB\_DATA7

#define CMD\_WR\_HOST\_DATA CMD10\_WR\_HOST\_DATA

#define CMD\_WR\_REQ\_DATA CMD01\_WR\_REQ\_DATA

#define CMD\_WR\_OFS\_DATA CMD20\_WR\_OFS\_DATA

#define CMD\_SET\_FILE\_NAME CMD10\_SET\_FILE\_NAME

#define CMD\_DISK\_CONNECT CMD0H\_DISK\_CONNECT

#define CMD\_DISK\_MOUNT CMD0H\_DISK\_MOUNT

#define CMD\_FILE\_OPEN CMD0H\_FILE\_OPEN

#define CMD\_FILE\_ENUM\_GO CMD0H\_FILE\_ENUM\_GO

#define CMD\_FILE\_CREATE CMD0H\_FILE\_CREATE

#define CMD\_FILE\_ERASE CMD0H\_FILE\_ERASE

#define CMD\_FILE\_CLOSE CMD1H\_FILE\_CLOSE

#define CMD\_DIR\_INFO\_READ CMD1H\_DIR\_INFO\_READ

#define CMD\_DIR\_INFO\_SAVE CMD0H\_DIR\_INFO\_SAVE

#define CMD\_BYTE\_LOCATE CMD4H\_BYTE\_LOCATE

#define CMD\_BYTE\_READ CMD2H\_BYTE\_READ

#define CMD\_BYTE\_RD\_GO CMD0H\_BYTE\_RD\_GO

#define CMD\_BYTE\_WRITE CMD2H\_BYTE\_WRITE

#define CMD\_BYTE\_WR\_GO CMD0H\_BYTE\_WR\_GO

#define CMD\_DISK\_CAPACITY CMD0H\_DISK\_CAPACITY

#define CMD\_DISK\_QUERY CMD0H\_DISK\_QUERY

#define CMD\_DIR\_CREATE CMD0H\_DIR\_CREATE

#define CMD\_SEC\_LOCATE CMD4H\_SEC\_LOCATE

#define CMD\_SEC\_READ CMD1H\_SEC\_READ

#define CMD\_SEC\_WRITE CMD1H\_SEC\_WRITE

#define CMD\_DISK\_BOC\_CMD CMD0H\_DISK\_BOC\_CMD

#define CMD\_DISK\_READ CMD5H\_DISK\_READ

#define CMD\_DISK\_RD\_GO CMD0H\_DISK\_RD\_GO

#define CMD\_DISK\_WRITE CMD5H\_DISK\_WRITE

#define CMD\_DISK\_WR\_GO CMD0H\_DISK\_WR\_GO

#define CMD\_SET\_USB\_SPEED CMD10\_SET\_USB\_SPEED

#define CMD\_GET\_DEV\_RATE CMD11\_GET\_DEV\_RATE

#define CMD\_GET\_TOGGLE CMD11\_GET\_TOGGLE

#define CMD\_READ\_VAR8 CMD11\_READ\_VAR8

#define CMD\_SET\_RETRY CMD20\_SET\_RETRY

#define CMD\_WRITE\_VAR8 CMD20\_WRITE\_VAR8

#define CMD\_READ\_VAR32 CMD14\_READ\_VAR32

#define CMD\_WRITE\_VAR32 CMD50\_WRITE\_VAR32

#define CMD\_DELAY\_100US CMD01\_DELAY\_100US

#define CMD\_SET\_USB\_ID CMD40\_SET\_USB\_ID

#define CMD\_SET\_USB\_ADDR CMD10\_SET\_USB\_ADDR

#define CMD\_TEST\_CONNECT CMD01\_TEST\_CONNECT

#define CMD\_ABORT\_NAK CMD00\_ABORT\_NAK

#define CMD\_SET\_ENDP2 CMD10\_SET\_ENDP2

#define CMD\_SET\_ENDP3 CMD10\_SET\_ENDP3

#define CMD\_SET\_ENDP4 CMD10\_SET\_ENDP4

#define CMD\_SET\_ENDP5 CMD10\_SET\_ENDP5

#define CMD\_SET\_ENDP6 CMD10\_SET\_ENDP6

#define CMD\_SET\_ENDP7 CMD10\_SET\_ENDP7

#define CMD\_DIRTY\_BUFFER CMD00\_DIRTY\_BUFFER

#define CMD\_WR\_USB\_DATA3 CMD10\_WR\_USB\_DATA3

#define CMD\_WR\_USB\_DATA5 CMD10\_WR\_USB\_DATA5

#define CMD\_CLR\_STALL CMD1H\_CLR\_STALL

#define CMD\_SET\_ADDRESS CMD1H\_SET\_ADDRESS

#define CMD\_GET\_DESCR CMD1H\_GET\_DESCR

#define CMD\_SET\_CONFIG CMD1H\_SET\_CONFIG

#define CMD\_AUTO\_SETUP CMD0H\_AUTO\_SETUP

#define CMD\_ISSUE\_TKN\_X CMD2H\_ISSUE\_TKN\_X

#define CMD\_ISSUE\_TOKEN CMD1H\_ISSUE\_TOKEN

#define CMD\_DISK\_INIT CMD0H\_DISK\_INIT

#define CMD\_DISK\_RESET CMD0H\_DISK\_RESET

#define CMD\_DISK\_SIZE CMD0H\_DISK\_SIZE

#define CMD\_DISK\_INQUIRY CMD0H\_DISK\_INQUIRY

#define CMD\_DISK\_READY CMD0H\_DISK\_READY

#define CMD\_DISK\_R\_SENSE CMD0H\_DISK\_R\_SENSE

#define CMD\_RD\_DISK\_SEC CMD0H\_RD\_DISK\_SEC

#define CMD\_WR\_DISK\_SEC CMD0H\_WR\_DISK\_SEC

#define CMD\_DISK\_MAX\_LUN CMD0H\_DISK\_MAX\_LUN

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 并口方式, 状态端口(读命令端口)的位定义 \*/

#ifndef PARA\_STATE\_INTB

#define PARA\_STATE\_INTB 0x80 /\* 并口方式状态端口的位7: 中断标志,低有效 \*/

#define PARA\_STATE\_BUSY 0x10 /\* 并口方式状态端口的位4: 忙标志,高有效 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 串口方式, 操作命令前的引导同步码 \*/

#ifndef SER\_CMD\_TIMEOUT

#define SER\_CMD\_TIMEOUT 32 /\* 串口命令超时时间, 单位为mS, 同步码之间及同步码与命令码之间的间隔应该尽量短, 超时后的处理方式为丢弃 \*/

#define SER\_SYNC\_CODE1 0x57 /\* 启动操作的第1个串口同步码 \*/

#define SER\_SYNC\_CODE2 0xAB /\* 启动操作的第2个串口同步码 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 操作状态 \*/

#ifndef CMD\_RET\_SUCCESS

#define CMD\_RET\_SUCCESS 0x51 /\* 命令操作成功 \*/

#define CMD\_RET\_ABORT 0x5F /\* 命令操作失败 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* USB中断状态 \*/

#ifndef USB\_INT\_EP0\_SETUP

/\* 以下状态代码为特殊事件中断, 如果通过CMD20\_CHK\_SUSPEND启用USB总线挂起检查, 那么必须处理USB总线挂起和睡眠唤醒的中断状态 \*/

#define USB\_INT\_USB\_SUSPEND 0x05 /\* USB总线挂起事件 \*/

#define USB\_INT\_WAKE\_UP 0x06 /\* 从睡眠中被唤醒事件 \*/

/\* 以下状态代码0XH用于USB设备方式 \*/

/\* 内置固件模式下只需要处理: USB\_INT\_EP1\_OUT, USB\_INT\_EP1\_IN, USB\_INT\_EP2\_OUT, USB\_INT\_EP2\_IN \*/

/\* 位7-位4为0000 \*/

/\* 位3-位2指示当前事务, 00=OUT, 10=IN, 11=SETUP \*/

/\* 位1-位0指示当前端点, 00=端点0, 01=端点1, 10=端点2, 11=USB总线复位 \*/

#define USB\_INT\_EP0\_SETUP 0x0C /\* USB端点0的SETUP \*/

#define USB\_INT\_EP0\_OUT 0x00 /\* USB端点0的OUT \*/

#define USB\_INT\_EP0\_IN 0x08 /\* USB端点0的IN \*/

#define USB\_INT\_EP1\_OUT 0x01 /\* USB端点1的OUT \*/

#define USB\_INT\_EP1\_IN 0x09 /\* USB端点1的IN \*/

#define USB\_INT\_EP2\_OUT 0x02 /\* USB端点2的OUT \*/

#define USB\_INT\_EP2\_IN 0x0A /\* USB端点2的IN \*/

/\* USB\_INT\_BUS\_RESET 0x0000XX11B \*/ /\* USB总线复位 \*/

#define USB\_INT\_BUS\_RESET1 0x03 /\* USB总线复位 \*/

#define USB\_INT\_BUS\_RESET2 0x07 /\* USB总线复位 \*/

#define USB\_INT\_BUS\_RESET3 0x0B /\* USB总线复位 \*/

#define USB\_INT\_BUS\_RESET4 0x0F /\* USB总线复位 \*/

#endif

/\* 以下状态代码2XH-3XH用于USB主机方式的通讯失败代码 \*/

/\* 位7-位6为00 \*/

/\* 位5为1 \*/

/\* 位4指示当前接收的数据包是否同步 \*/

/\* 位3-位0指示导致通讯失败时USB设备的应答: 0010=ACK, 1010=NAK, 1110=STALL, 0011=DATA0, 1011=DATA1, XX00=超时 \*/

/\* USB\_INT\_RET\_ACK 0x001X0010B \*/ /\* 错误:对于IN事务返回ACK \*/

/\* USB\_INT\_RET\_NAK 0x001X1010B \*/ /\* 错误:返回NAK \*/

/\* USB\_INT\_RET\_STALL 0x001X1110B \*/ /\* 错误:返回STALL \*/

/\* USB\_INT\_RET\_DATA0 0x001X0011B \*/ /\* 错误:对于OUT/SETUP事务返回DATA0 \*/

/\* USB\_INT\_RET\_DATA1 0x001X1011B \*/ /\* 错误:对于OUT/SETUP事务返回DATA1 \*/

/\* USB\_INT\_RET\_TOUT 0x001XXX00B \*/ /\* 错误:返回超时 \*/

/\* USB\_INT\_RET\_TOGX 0x0010X011B \*/ /\* 错误:对于IN事务返回数据不同步 \*/

/\* USB\_INT\_RET\_PID 0x001XXXXXB \*/ /\* 错误:未定义 \*/

/\* 以下状态代码1XH用于USB主机方式的操作状态代码 \*/

#ifndef USB\_INT\_SUCCESS

#define USB\_INT\_SUCCESS 0x14 /\* USB事务或者传输操作成功 \*/

#define USB\_INT\_CONNECT 0x15 /\* 检测到USB设备连接事件, 可能是新连接或者断开后重新连接 \*/

#define USB\_INT\_DISCONNECT 0x16 /\* 检测到USB设备断开事件 \*/

#define USB\_INT\_BUF\_OVER 0x17 /\* USB传输的数据有误或者数据太多缓冲区溢出 \*/

#define USB\_INT\_USB\_READY 0x18 /\* USB设备已经被初始化(已经分配USB地址) \*/

#define USB\_INT\_DISK\_READ 0x1D /\* USB存储器请求数据读出 \*/

#define USB\_INT\_DISK\_WRITE 0x1E /\* USB存储器请求数据写入 \*/

#define USB\_INT\_DISK\_ERR 0x1F /\* USB存储器操作失败 \*/

#endif

/\* 以下状态代码用于主机文件模式下的文件系统错误码 \*/

#ifndef ERR\_DISK\_DISCON

#define ERR\_DISK\_DISCON 0x82 /\* 磁盘尚未连接,可能磁盘已经断开 \*/

#define ERR\_LARGE\_SECTOR 0x84 /\* 磁盘的扇区太大,只支持每扇区512字节 \*/

#define ERR\_TYPE\_ERROR 0x92 /\* 磁盘分区类型不支持,只支持FAT12/FAT16/BigDOS/FAT32,需要由磁盘管理工具重新分区 \*/

#define ERR\_BPB\_ERROR 0xA1 /\* 磁盘尚未格式化,或者参数错误,需要由WINDOWS采用默认参数重新格式化 \*/

#define ERR\_DISK\_FULL 0xB1 /\* 磁盘文件太满,剩余空间太少或者已经没有,需要磁盘整理 \*/

#define ERR\_FDT\_OVER 0xB2 /\* 目录(文件夹)内文件太多,没有空闲的目录项,FAT12/FAT16根目录下的文件数应该少于512个,需要磁盘整理 \*/

#define ERR\_FILE\_CLOSE 0xB4 /\* 文件已经关闭,如果需要使用,应该重新打开文件 \*/

#define ERR\_OPEN\_DIR 0x41 /\* 指定路径的目录(文件夹)被打开 \*/

#define ERR\_MISS\_FILE 0x42 /\* 指定路径的文件没有找到,可能是文件名称错误 \*/

#define ERR\_FOUND\_NAME 0x43 /\* 搜索到相匹配的文件名,或者是要求打开目录(文件夹)而实际结果却打开了文件 \*/

/\* 以下文件系统错误码用于文件系统子程序 \*/

#define ERR\_MISS\_DIR 0xB3 /\* 指定路径的某个子目录(文件夹)没有找到,可能是目录名称错误 \*/

#define ERR\_LONG\_BUF\_OVER 0x48 /\* 长文件缓冲区溢出 \*/

#define ERR\_LONG\_NAME\_ERR 0x49 /\* 短文件名没有对应的长文件名或者长文件名错误 \*/

#define ERR\_NAME\_EXIST 0x4A /\* 同名的短文件已经存在,建议重新生成另外一个短文件名 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 以下状态代码用于主机文件模式下的磁盘及文件状态, VAR\_DISK\_STATUS \*/

#ifndef DEF\_DISK\_UNKNOWN

#define DEF\_DISK\_UNKNOWN 0x00 /\* 尚未初始化,未知状态 \*/

#define DEF\_DISK\_DISCONN 0x01 /\* 磁盘没有连接或者已经断开 \*/

#define DEF\_DISK\_CONNECT 0x02 /\* 磁盘已经连接,但是尚未初始化或者无法识别该磁盘 \*/

#define DEF\_DISK\_MOUNTED 0x03 /\* 磁盘已经初始化成功,但是尚未分析文件系统或者文件系统不支持 \*/

#define DEF\_DISK\_READY 0x10 /\* 已经分析磁盘的文件系统并且能够支持 \*/

#define DEF\_DISK\_OPEN\_ROOT 0x12 /\* 已经打开根目录,使用后必须关闭,注意FAT12/FAT16根目录是固定长度 \*/

#define DEF\_DISK\_OPEN\_DIR 0x13 /\* 已经打开子目录(文件夹) \*/

#define DEF\_DISK\_OPEN\_FILE 0x14 /\* 已经打开文件 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 文件系统常用定义 \*/

#ifndef DEF\_SECTOR\_SIZE

#define DEF\_SECTOR\_SIZE 512 /\* U盘或者SD卡默认的物理扇区的大小 \*/

#endif

#ifndef DEF\_WILDCARD\_CHAR

#define DEF\_WILDCARD\_CHAR 0x2A /\* 路径名的通配符 '\*' \*/

#define DEF\_SEPAR\_CHAR1 0x5C /\* 路径名的分隔符 '\' \*/

#define DEF\_SEPAR\_CHAR2 0x2F /\* 路径名的分隔符 '/' \*/

#define DEF\_FILE\_YEAR 2004 /\* 默认文件日期: 2004年 \*/

#define DEF\_FILE\_MONTH 1 /\* 默认文件日期: 1月 \*/

#define DEF\_FILE\_DATE 1 /\* 默认文件日期: 1日 \*/

#endif

#ifndef ATTR\_DIRECTORY

/\* FAT数据区中文件目录信息 \*/

typedef struct \_FAT\_DIR\_INFO {

UINT8 DIR\_Name[11]; /\* 00H,文件名,共11字节,不足处填空格 \*/

UINT8 DIR\_Attr; /\* 0BH,文件属性,参考后面的说明 \*/

UINT8 DIR\_NTRes; /\* 0CH \*/

UINT8 DIR\_CrtTimeTenth; /\* 0DH,文件创建的时间,以0.1秒单位计数 \*/

UINT16 DIR\_CrtTime; /\* 0EH,文件创建的时间 \*/

UINT16 DIR\_CrtDate; /\* 10H,文件创建的日期 \*/

UINT16 DIR\_LstAccDate; /\* 12H,最近一次存取操作的日期 \*/

UINT16 DIR\_FstClusHI; /\* 14H \*/

UINT16 DIR\_WrtTime; /\* 16H,文件修改时间,参考前面的宏MAKE\_FILE\_TIME \*/

UINT16 DIR\_WrtDate; /\* 18H,文件修改日期,参考前面的宏MAKE\_FILE\_DATE \*/

UINT16 DIR\_FstClusLO; /\* 1AH \*/

UINT32 DIR\_FileSize; /\* 1CH,文件长度 \*/

} FAT\_DIR\_INFO, \*P\_FAT\_DIR\_INFO; /\* 20H \*/

/\* 文件属性 \*/

#define ATTR\_READ\_ONLY 0x01 /\* 文件为只读属性 \*/

#define ATTR\_HIDDEN 0x02 /\* 文件为隐含属性 \*/

#define ATTR\_SYSTEM 0x04 /\* 文件为系统属性 \*/

#define ATTR\_VOLUME\_ID 0x08 /\* 卷标 \*/

#define ATTR\_DIRECTORY 0x10 /\* 子目录(文件夹) \*/

#define ATTR\_ARCHIVE 0x20 /\* 文件为存档属性 \*/

#define ATTR\_LONG\_NAME ( ATTR\_READ\_ONLY | ATTR\_HIDDEN | ATTR\_SYSTEM | ATTR\_VOLUME\_ID ) /\* 长文件名属性 \*/

#define ATTR\_LONG\_NAME\_MASK ( ATTR\_LONG\_NAME | ATTR\_DIRECTORY | ATTR\_ARCHIVE )

/\* 文件属性 UINT8 \*/

/\* bit0 bit1 bit2 bit3 bit4 bit5 bit6 bit7 \*/

/\* 只 隐 系 卷 目 存 未定义 \*/

/\* 读 藏 统 标 录 档 \*/

/\* 文件时间 UINT16 \*/

/\* Time = (Hour<<11) + (Minute<<5) + (Second>>1) \*/

#define MAKE\_FILE\_TIME( h, m, s ) ( (h<<11) + (m<<5) + (s>>1) ) /\* 生成指定时分秒的文件时间数据 \*/

/\* 文件日期 UINT16 \*/

/\* Date = ((Year-1980)<<9) + (Month<<5) + Day \*/

#define MAKE\_FILE\_DATE( y, m, d ) ( ((y-1980)<<9) + (m<<5) + d ) /\* 生成指定年月日的文件日期数据 \*/

#define LONE\_NAME\_MAX\_CHAR (255\*2) /\* 长文件名最多字符数/字节数 \*/

#define LONG\_NAME\_PER\_DIR (13\*2) /\* 长文件名在每个文件目录信息结构中的字符数/字节数 \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* SCSI命令和数据输入输出结构 \*/

#ifndef SPC\_CMD\_INQUIRY

/\* SCSI命令码 \*/

#define SPC\_CMD\_INQUIRY 0x12

#define SPC\_CMD\_READ\_CAPACITY 0x25

#define SPC\_CMD\_READ10 0x28

#define SPC\_CMD\_WRITE10 0x2A

#define SPC\_CMD\_TEST\_READY 0x00

#define SPC\_CMD\_REQUEST\_SENSE 0x03

#define SPC\_CMD\_MODESENSE6 0x1A

#define SPC\_CMD\_MODESENSE10 0x5A

#define SPC\_CMD\_START\_STOP 0x1B

/\* BulkOnly协议的命令块 \*/

typedef struct \_BULK\_ONLY\_CBW {

UINT32 CBW\_Sig;

UINT32 CBW\_Tag;

UINT8 CBW\_DataLen0; /\* 08H,输入: 数据传输长度,对于输入数据其有效值是0到48,对于输出数据其有效值为0到33 \*/

UINT8 CBW\_DataLen1;

UINT16 CBW\_DataLen2;

UINT8 CBW\_Flag; /\* 0CH,输入: 传输方向等标志,位7为1则输入数据,位为0则输出数据或者没有数据 \*/

UINT8 CBW\_LUN;

UINT8 CBW\_CB\_Len; /\* 0EH,输入: 命令块的长度,有效值是1到16 \*/

UINT8 CBW\_CB\_Buf[16]; /\* 0FH,输入: 命令块,该缓冲区最多为16个字节 \*/

} BULK\_ONLY\_CBW, \*P\_BULK\_ONLY\_CBW; /\* BulkOnly协议的命令块, 输入CBW结构 \*/

/\* INQUIRY命令的返回数据 \*/

typedef struct \_INQUIRY\_DATA {

UINT8 DeviceType; /\* 00H, 设备类型 \*/

UINT8 RemovableMedia; /\* 01H, 位7为1说明是移动存储 \*/

UINT8 Versions; /\* 02H, 协议版本 \*/

UINT8 DataFormatAndEtc; /\* 03H, 指定返回数据格式 \*/

UINT8 AdditionalLength; /\* 04H, 后续数据的长度 \*/

UINT8 Reserved1;

UINT8 Reserved2;

UINT8 MiscFlag; /\* 07H, 一些控制标志 \*/

UINT8 VendorIdStr[8]; /\* 08H, 厂商信息 \*/

UINT8 ProductIdStr[16]; /\* 10H, 产品信息 \*/

UINT8 ProductRevStr[4]; /\* 20H, 产品版本 \*/

} INQUIRY\_DATA, \*P\_INQUIRY\_DATA; /\* 24H \*/

/\* REQUEST SENSE命令的返回数据 \*/

typedef struct \_SENSE\_DATA {

UINT8 ErrorCode; /\* 00H, 错误代码及有效位 \*/

UINT8 SegmentNumber;

UINT8 SenseKeyAndEtc; /\* 02H, 主键码 \*/

UINT8 Information0;

UINT8 Information1;

UINT8 Information2;

UINT8 Information3;

UINT8 AdditSenseLen; /\* 07H, 后续数据的长度 \*/

UINT8 CmdSpecInfo[4];

UINT8 AdditSenseCode; /\* 0CH, 附加键码 \*/

UINT8 AddSenCodeQual; /\* 0DH, 详细的附加键码 \*/

UINT8 FieldReplaUnit;

UINT8 SenseKeySpec[3];

} SENSE\_DATA, \*P\_SENSE\_DATA; /\* 12H \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 主机文件模式下的数据输入和输出结构 \*/

#ifndef MAX\_FILE\_NAME\_LEN

#define MAX\_FILE\_NAME\_LEN (13+1) /\* 文件名最大长度,最大长度是1个根目录符+8个主文件名+1个小数点+3个类型名+结束符=14 \*/

/\* 命令的输入数据和输出数据 \*/

typedef union \_CH376\_CMD\_DATA {

struct {

UINT8 mBuffer[ MAX\_FILE\_NAME\_LEN ];

} Default;

INQUIRY\_DATA DiskMountInq; /\* 返回: INQUIRY命令的返回数据 \*/

/\* CMD0H\_DISK\_MOUNT: 初始化磁盘并测试磁盘是否就绪,首次执行时 \*/

FAT\_DIR\_INFO OpenDirInfo; /\* 返回: 枚举到的文件目录信息 \*/

/\* CMD0H\_FILE\_OPEN: 枚举文件和目录(文件夹) \*/

FAT\_DIR\_INFO EnumDirInfo; /\* 返回: 枚举到的文件目录信息 \*/

/\* CMD0H\_FILE\_ENUM\_GO: 继续枚举文件和目录(文件夹) \*/

struct {

UINT8 mUpdateFileSz; /\* 输入参数: 是否允许更新文件长度, 0则禁止更新长度 \*/

} FileCLose; /\* CMD1H\_FILE\_CLOSE: 关闭当前已经打开的文件 \*/

struct {

UINT8 mDirInfoIndex; /\* 输入参数: 指定需要读取的目录信息结构在扇区内的索引号, 0FFH则为当前已经打开的文件 \*/

} DirInfoRead; /\* CMD1H\_DIR\_INFO\_READ: 读取文件的目录信息 \*/

union {

UINT32 mByteOffset; /\* 输入参数: 偏移字节数,以字节为单位的偏移量(总长度32位,低字节在前) \*/

UINT32 mSectorLba; /\* 返回: 当前文件指针对应的绝对线性扇区号,0FFFFFFFFH则已到文件尾(总长度32位,低字节在前) \*/

} ByteLocate; /\* CMD4H\_BYTE\_LOCATE: 以字节为单位移动当前文件指针 \*/

struct {

UINT16 mByteCount; /\* 输入参数: 请求读取的字节数(总长度16位,低字节在前) \*/

} ByteRead; /\* CMD2H\_BYTE\_READ: 以字节为单位从当前位置读取数据块 \*/

struct {

UINT16 mByteCount; /\* 输入参数: 请求写入的字节数(总长度16位,低字节在前) \*/

} ByteWrite; /\* CMD2H\_BYTE\_WRITE: 以字节为单位向当前位置写入数据块 \*/

union {

UINT32 mSectorOffset; /\* 输入参数: 偏移扇区数,以扇区为单位的偏移量(总长度32位,低字节在前) \*/

UINT32 mSectorLba; /\* 返回: 当前文件指针对应的绝对线性扇区号,0FFFFFFFFH则已到文件尾(总长度32位,低字节在前) \*/

} SectorLocate; /\* CMD4H\_SEC\_LOCATE: 以扇区为单位移动当前文件指针 \*/

struct {

UINT8 mSectorCount; /\* 输入参数: 请求读取的扇区数 \*/

/\* 返回: 允许读取的扇区数 \*/

UINT8 mReserved1;

UINT8 mReserved2;

UINT8 mReserved3;

UINT32 mStartSector; /\* 返回: 允许读取的扇区块的起始绝对线性扇区号(总长度32位,低字节在前) \*/

} SectorRead; /\* CMD1H\_SEC\_READ: 以扇区为单位从当前位置读取数据块 \*/

struct {

UINT8 mSectorCount; /\* 输入参数: 请求写入的扇区数 \*/

/\* 返回: 允许写入的扇区数 \*/

UINT8 mReserved1;

UINT8 mReserved2;

UINT8 mReserved3;

UINT32 mStartSector; /\* 返回: 允许写入的扇区块的起始绝对线性扇区号(总长度32位,低字节在前) \*/

} SectorWrite; /\* CMD1H\_SEC\_WRITE: 以扇区为单位在当前位置写入数据块 \*/

struct {

UINT32 mDiskSizeSec; /\* 返回: 整个物理磁盘的总扇区数(总长度32位,低字节在前) \*/

} DiskCapacity; /\* CMD0H\_DISK\_CAPACITY: 查询磁盘物理容量 \*/

struct {

UINT32 mTotalSector; /\* 返回: 当前逻辑盘的总扇区数(总长度32位,低字节在前) \*/

UINT32 mFreeSector; /\* 返回: 当前逻辑盘的剩余扇区数(总长度32位,低字节在前) \*/

UINT8 mDiskFat; /\* 返回: 当前逻辑盘的FAT类型,1-FAT12,2-FAT16,3-FAT32 \*/

} DiskQuery; /\* CMD\_DiskQuery, 查询磁盘信息 \*/

BULK\_ONLY\_CBW DiskBocCbw; /\* 输入参数: CBW命令结构 \*/

/\* CMD0H\_DISK\_BOC\_CMD: 对USB存储器执行BulkOnly传输协议的命令 \*/

struct {

UINT8 mMaxLogicUnit; /\* 返回: USB存储器的最大逻辑单元号 \*/

} DiskMaxLun; /\* CMD0H\_DISK\_MAX\_LUN: 控制传输-获取USB存储器最大逻辑单元号 \*/

INQUIRY\_DATA DiskInitInq; /\* 返回: INQUIRY命令的返回数据 \*/

/\* CMD0H\_DISK\_INIT: 初始化USB存储器 \*/

INQUIRY\_DATA DiskInqData; /\* 返回: INQUIRY命令的返回数据 \*/

/\* CMD0H\_DISK\_INQUIRY: 查询USB存储器特性 \*/

SENSE\_DATA ReqSenseData; /\* 返回: REQUEST SENSE命令的返回数据 \*/

/\* CMD0H\_DISK\_R\_SENSE: 检查USB存储器错误 \*/

struct {

UINT32 mDiskSizeSec; /\* 返回: 整个物理磁盘的总扇区数(总长度32位,高字节在前) \*/

} DiskSize; /\* CMD0H\_DISK\_SIZE: 获取USB存储器的容量 \*/

struct {

UINT32 mStartSector; /\* 输入参数: LBA扇区地址(总长度32位,低字节在前) \*/

UINT8 mSectorCount; /\* 输入参数: 请求读取的扇区数 \*/

} DiskRead; /\* CMD5H\_DISK\_READ: 从USB存储器读数据块(以扇区为单位) \*/

struct {

UINT32 mStartSector; /\* 输入参数: LBA扇区地址(总长度32位,低字节在前) \*/

UINT8 mSectorCount; /\* 输入参数: 请求写入的扇区数 \*/

} DiskWrite; /\* CMD5H\_DISK\_WRITE: 向USB存储器写数据块(以扇区为单位) \*/

} CH376\_CMD\_DATA, \*P\_CH376\_CMD\_DATA;

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 主机文件模式下的文件系统变量的地址 \*/

#ifndef VAR\_FILE\_SIZE

/\* 8位/单字节变量 \*/

#define VAR\_SYS\_BASE\_INFO 0x20 /\* 当前系统的基本信息 \*/

/\* 位6用于指示USB存储设备的子类别SubClass-Code, 位6为0则说明子类别为6, 位6为1则说明子类别是非6的其它值 \*/

/\* 位5用于指示USB设备方式下的USB配置状态和USB主机方式下的USB设备连接状态 \*/

/\* USB设备方式下, 位5为1则USB配置完成, 位5位0则尚未配置 \*/

/\* USB主机方式下, 位5为1则USB端口存在USB设备, 位5位0则USB端口没有USB设备 \*/

/\* 位4用于指示USB设备方式下的缓冲区锁定状态, 位4为1则说明USB缓冲区处于锁定状态, 位6为1则说明已经释放 \*/

/\* 其它位, 保留,请勿修改 \*/

#define VAR\_RETRY\_TIMES 0x25 /\* USB事务操作的重试次数 \*/

/\* 位7为0则收到NAK时不重试, 位7为1位6为0则收到NAK时无限重试(可以用CMD\_ABORT\_NAK命令放弃重试), 位7为1位6为1则收到NAK时最多重试3秒 \*/

/\* 位5~位0为超时后的重试次数 \*/

#define VAR\_FILE\_BIT\_FLAG 0x26 /\* 主机文件模式下的位标志 \*/

/\* 位1和位0, 逻辑盘的FAT文件系统标志, 00-FAT12, 01-FAT16, 10-FAT32, 11-非法 \*/

/\* 位2, 当前缓冲区中的FAT表数据是否被修改标志, 0-未修改, 1-已修改 \*/

/\* 位3, 文件长度需要修改标志, 当前文件被追加数据, 0-未追加无需修改, 1-已追加需要修改 \*/

/\* 其它位, 保留,请勿修改 \*/

#define VAR\_DISK\_STATUS 0x2B /\* 主机文件模式下的磁盘及文件状态 \*/

#define VAR\_SD\_BIT\_FLAG 0x30 /\* 主机文件模式下SD卡的位标志 \*/

/\* 位0, SD卡版本, 0-只支持SD第一版,1-支持SD第二版 \*/

/\* 位1, 自动识别, 0-SD卡, 1-MMC卡 \*/

/\* 位2, 自动识别, 0-标准容量SD卡, 1-大容量SD卡(HC-SD) \*/

/\* 位4, ACMD41命令超时 \*/

/\* 位5, CMD1命令超时 \*/

/\* 位6, CMD58命令超时 \*/

/\* 其它位, 保留,请勿修改 \*/

#define VAR\_UDISK\_TOGGLE 0x31 /\* USB存储设备的BULK-IN/BULK-OUT端点的同步标志 \*/

/\* 位7, Bulk-In端点的同步标志 \*/

/\* 位6, Bulk-In端点的同步标志 \*/

/\* 位5~位0, 必须为0 \*/

#define VAR\_UDISK\_LUN 0x34 /\* USB存储设备的逻辑单元号 \*/

/\* 位7~位4, USB存储设备的当前逻辑单元号,CH376初始化USB存储设备后,默认是访问0#逻辑单元 \*/

/\* 位3~位0, USB存储设备的最大逻辑单元号,加1后等于逻辑单元数 \*/

#define VAR\_SEC\_PER\_CLUS 0x38 /\* 逻辑盘的每簇扇区数 \*/

#define VAR\_FILE\_DIR\_INDEX 0x3B /\* 当前文件目录信息在扇区内的索引号 \*/

#define VAR\_CLUS\_SEC\_OFS 0x3C /\* 当前文件指针在簇内的扇区偏移,为0xFF则指向文件末尾,簇结束 \*/

/\* 32位/4字节变量 \*/

#define VAR\_DISK\_ROOT 0x44 /\* 对于FAT16盘为根目录占用扇区数,对于FAT32盘为根目录起始簇号(总长度32位,低字节在前) \*/

#define VAR\_DSK\_TOTAL\_CLUS 0x48 /\* 逻辑盘的总簇数(总长度32位,低字节在前) \*/

#define VAR\_DSK\_START\_LBA 0x4C /\* 逻辑盘的起始绝对扇区号LBA(总长度32位,低字节在前) \*/

#define VAR\_DSK\_DAT\_START 0x50 /\* 逻辑盘的数据区域的起始LBA(总长度32位,低字节在前) \*/

#define VAR\_LBA\_BUFFER 0x54 /\* 当前磁盘数据缓冲区的数据对应的LBA(总长度32位,低字节在前) \*/

#define VAR\_LBA\_CURRENT 0x58 /\* 当前读写的磁盘起始LBA地址(总长度32位,低字节在前) \*/

#define VAR\_FAT\_DIR\_LBA 0x5C /\* 当前文件目录信息所在的扇区LBA地址(总长度32位,低字节在前) \*/

#define VAR\_START\_CLUSTER 0x60 /\* 当前文件或者目录(文件夹)的起始簇号(总长度32位,低字节在前) \*/

#define VAR\_CURRENT\_CLUST 0x64 /\* 当前文件的当前簇号(总长度32位,低字节在前) \*/

#define VAR\_FILE\_SIZE 0x68 /\* 当前文件的长度(总长度32位,低字节在前) \*/

#define VAR\_CURRENT\_OFFSET 0x6C /\* 当前文件指针,当前读写位置的字节偏移(总长度32位,低字节在前) \*/

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* 常用USB定义 \*/

/\* USB的包标识PID, 主机方式可能用到 \*/

#ifndef DEF\_USB\_PID\_SETUP

#define DEF\_USB\_PID\_NULL 0x00 /\* 保留PID, 未定义 \*/

#define DEF\_USB\_PID\_SOF 0x05

#define DEF\_USB\_PID\_SETUP 0x0D

#define DEF\_USB\_PID\_IN 0x09

#define DEF\_USB\_PID\_OUT 0x01

#define DEF\_USB\_PID\_ACK 0x02

#define DEF\_USB\_PID\_NAK 0x0A

#define DEF\_USB\_PID\_STALL 0x0E

#define DEF\_USB\_PID\_DATA0 0x03

#define DEF\_USB\_PID\_DATA1 0x0B

#define DEF\_USB\_PID\_PRE 0x0C

#endif

/\* USB请求类型, 外置固件模式可能用到 \*/

#ifndef DEF\_USB\_REQ\_TYPE

#define DEF\_USB\_REQ\_READ 0x80 /\* 控制读操作 \*/

#define DEF\_USB\_REQ\_WRITE 0x00 /\* 控制写操作 \*/

#define DEF\_USB\_REQ\_TYPE 0x60 /\* 控制请求类型 \*/

#define DEF\_USB\_REQ\_STAND 0x00 /\* 标准请求 \*/

#define DEF\_USB\_REQ\_CLASS 0x20 /\* 设备类请求 \*/

#define DEF\_USB\_REQ\_VENDOR 0x40 /\* 厂商请求 \*/

#define DEF\_USB\_REQ\_RESERVE 0x60 /\* 保留请求 \*/

#endif

/\* USB标准设备请求, RequestType的位6位5=00(Standard), 外置固件模式可能用到 \*/

#ifndef DEF\_USB\_GET\_DESCR

#define DEF\_USB\_CLR\_FEATURE 0x01

#define DEF\_USB\_SET\_FEATURE 0x03

#define DEF\_USB\_GET\_STATUS 0x00

#define DEF\_USB\_SET\_ADDRESS 0x05

#define DEF\_USB\_GET\_DESCR 0x06

#define DEF\_USB\_SET\_DESCR 0x07

#define DEF\_USB\_GET\_CONFIG 0x08

#define DEF\_USB\_SET\_CONFIG 0x09

#define DEF\_USB\_GET\_INTERF 0x0A

#define DEF\_USB\_SET\_INTERF 0x0B

#define DEF\_USB\_SYNC\_FRAME 0x0C

#endif

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#ifdef \_\_cplusplus

}

#endif

#endif

CH376HFT.C文件

/\* 2008.10.18

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Copyright (C) W.ch 1999-2009 \*\*

\*\* Web: http://www.winchiphead.com \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* USB Host File Interface for CH376 \*\*

\*\* TC2.0@PC, KC7.0@MCS51 \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* CH376 主机文件系统接口 \*/

/\* MCS-51单片机C语言的U盘文件读写示例程序 \*/

/\* 本程序演示字节读写,文件枚举,文件复制, 用于将U盘中的/C51/CH376HFT.C文件中的小写字母转成大写字母后, 写到新建的文件NEWFILE.TXT中,

如果找不到原文件CH376HFT.C, 那么该程序将显示C51子目录下所有以CH376开头的文件名, 并新建NEWFILE.TXT文件并写入提示信息,

如果找不到C51子目录, 那么该程序将显示根目录下的所有文件名, 并新建NEWFILE.TXT文件并写入提示信息 \*/

/\* C51 CH376HFT.C \*/

/\* LX51 CH376HFT.OBJ \*/

/\* OHX51 CH376HFT \*/

#include <reg52.h>

#include <stdio.h>

#include <string.h>

#define EN\_DISK\_QUERY 1 /\* 启用磁盘查询 \*/

#include "HAL.H"

#include "HAL\_BASE.C"

#include "DEBUG.H"

#include "DEBUG.C"

#include "SPI\_SW.C" /\* 软件模拟SPI方式 \*/

#include "FILE\_SYS.H"

#include "FILE\_SYS.C"

UINT8 idata buf[64];

void main(void)

{

UINT8 i, s;

UINT16 len;

P\_FAT\_DIR\_INFO pDir;

UINT8 xdata SrcName[64];

UINT8 xdata TarName[64];

mDelaymS( 100 ); /\* 延时100毫秒 \*/

mInitSTDIO( ); /\* 为了让计算机通过串口监控演示过程 \*/

printf( "系统启动---Start\n" );

s = mInitCH376Host( ); /\* 初始化CH376 \*/

mStopIfError( s );

/\* 其它电路初始化 \*/

while ( 1 )

{

printf( "等待U盘或者SD卡插入---Wait Udisk/SD\n" );

while ( CH376DiskConnect( ) != USB\_INT\_SUCCESS )

{

/\* 检查U盘是否连接,等待U盘插入,对于SD卡,可以由单片机直接查询SD卡座的插拔状态引脚 \*/

mDelaymS( 100 );

}

mDelaymS( 200 ); /\* 延时,可选操作,有的USB存储器需要几十毫秒的延时 \*/

/\* 对于检测到USB设备的,最多等待100\*50mS,主要针对有些MP3太慢,对于检测到USB设备并且连接DISK\_MOUNTED的,最多等待5\*50mS,主要针对DiskReady不过的 \*/

for ( i = 0; i < 100; i ++ )

{

/\* 最长等待时间,100\*50mS \*/

mDelaymS( 50 );

printf( "准备好了? ---- Ready ?\n" );

s = CH376DiskMount( ); /\* 初始化磁盘并测试磁盘是否就绪 \*/

if ( s == USB\_INT\_SUCCESS )

break; /\* 准备好 \*/

else

if ( s == ERR\_DISK\_DISCON )

break; /\* 检测到断开,重新检测并计时 \*/

if ( CH376GetDiskStatus( ) >= DEF\_DISK\_MOUNTED && i >= 5 )

break; /\* 有的U盘总是返回未准备好,不过可以忽略,只要其建立连接MOUNTED且尝试5\*50mS \*/

}

if ( s == ERR\_DISK\_DISCON )

{

/\* 检测到断开,重新检测并计时 \*/

printf( "设备已断开---Device gone\n" );

continue;

}

if ( CH376GetDiskStatus( ) < DEF\_DISK\_MOUNTED )

{

/\* 未知USB设备,例如USB键盘、打印机等 \*/

printf( "Unknown device\n" );

goto UnknownUsbDevice;

}

i = CH376ReadBlock( buf ); /\* 如果需要,可以读取数据块CH376\_CMD\_DATA.DiskMountInq,返回长度 \*/

if ( i == sizeof( INQUIRY\_DATA ) )

{

/\* U盘的厂商和产品信息 \*/

buf[ i ] = 0;

printf( "U盘厂商---UdiskInfo: %s\n", ((P\_INQUIRY\_DATA)buf) -> VendorIdStr );

}

/\* 读取文件 \*/

strcpy( SrcName, "\\C51\\CH376HFT.C" ); /\* 源文件名,多级目录下的文件名和路径名必须复制到RAM中再处理,而根目录或者当前目录下的文件名可以在RAM或者ROM中 \*/

strcpy( TarName, "\\NEWFILE.TXT" ); /\* 目标文件名 \*/

printf( "打开文件 --- Open\n" );

s = CH376FileOpenPath( SrcName ); /\* 打开文件,该文件在C51子目录下 \*/

if ( s == ERR\_MISS\_DIR || s == ERR\_MISS\_FILE )

{

/\* 没有找到目录或者没有找到文件 \*/

/\* 列出文件,完整枚举可以参考EXAM13全盘枚举 \*/

if ( s == ERR\_MISS\_DIR )

strcpy( buf, "\\\*" ); /\* C51子目录不存在则列出根目录下的文件 \*/

else

strcpy( buf, "\\C51\\CH376\*" ); /\* CH376HFT.C文件不存在则列出\C51子目录下的以CH376开头的文件 \*/

printf( "列举所有文件---List file %s\n", buf );

s = CH376FileOpenPath( buf ); /\* 枚举多级目录下的文件或者目录,输入缓冲区必须在RAM中 \*/

while ( s == USB\_INT\_DISK\_READ )

{

/\* 枚举到匹配的文件 \*/

CH376ReadBlock( buf ); /\* 读取枚举到的文件的FAT\_DIR\_INFO结构,返回长度总是sizeof( FAT\_DIR\_INFO ) \*/

pDir = (P\_FAT\_DIR\_INFO)buf; /\* 当前文件目录信息 \*/

if ( pDir -> DIR\_Name[0] != '.' )

{

/\* 不是本级或者上级目录名则继续,否则必须丢弃不处理 \*/

if ( pDir -> DIR\_Name[0] == 0x05 )

pDir -> DIR\_Name[0] = 0xE5; /\* 特殊字符替换 \*/

pDir -> DIR\_Attr = 0; /\* 强制文件名字符串结束以便打印输出 \*/

/\* 打印名称,原始8+3格式,未整理成含小数点分隔符 \*/

printf( "\*\*\* EnumName: %s\n", pDir -> DIR\_Name );

}

xWriteCH376Cmd( CMD0H\_FILE\_ENUM\_GO ); /\* 继续枚举文件和目录 \*/

xEndCH376Cmd( );

s = Wait376Interrupt( );

}

if ( s != ERR\_MISS\_FILE )

mStopIfError( s ); /\* 操作出错 \*/

/\*--- 新建一个文件 ---\*/

printf( "新文件被创建---Create\n" );

s = CH376FileCreatePath( TarName ); /\* 新建多级目录下的文件,支持多级目录路径,输入缓冲区必须在RAM中 \*/

mStopIfError( s );

printf( "新文件被写入---Write\n" );

strcpy( buf, "找不到/C51/CH376HFT.C文件,欢迎使用一鸣电子模块!\xd\n" );

s = CH376ByteWrite( buf, strlen(buf), NULL ); /\* 以字节为单位向当前位置写入数据块 \*/

mStopIfError( s );

printf( "新文件被关闭---Close\n" );

s = CH376FileClose( TRUE ); /\* 关闭文件,对于字节读写建议自动更新文件长度 \*/

mStopIfError( s );

printf("\n");

strcpy( TarName, "\\YI.TXT" ); /\* 目标文件名 \*/

/\*--- 新建另外一个文件；文件名：yimingtest.txt ---\*/

printf( "创建另外一个文件---Create\n" );

s = CH376FileCreatePath( TarName );

mStopIfError( s );

printf( "新文件被写入----Write\n" );

strcpy( buf, "恭喜发财了!\xd\n" );

s = CH376ByteWrite( buf, strlen(buf) , NULL);

mStopIfError ( s );

printf( "文件被关闭 --- Close\n" );

s = CH376FileClose(TRUE);

mStopIfError( s );

/\*--- 追加文件数据的例子 ---\*/

printf( "\n" );

strcpy( SrcName, "\\YI.TXT" );

s = CH376FileOpenPath( SrcName );

printf( "Error: %02X\n", (UINT16)s );

if( s == USB\_INT\_SUCCESS )

{

//成功打开文件

strcpy( buf, "是不是要请哥们吃饭了呢?\xd\n" );

CH376ByteLocate(0xFFFFFFFF);

s = CH376ByteWrite( buf, strlen(buf) , NULL);

mStopIfError ( s );

printf( "文件被关闭 --- Close\n" );

s = CH376FileClose(TRUE);

mStopIfError( s );

}

else

mStopIfError( s );

/\*--- 读取文件数据的例子 ---\*/

printf( "\n" );

printf( "读取文件数据的例子:\n" );

strcpy( SrcName, "\\YI.TXT" );

s = CH376FileOpenPath( SrcName );

printf( "Error: %02X\n", (UINT16)s );

if( s == USB\_INT\_SUCCESS )

{

len = CH376GetFileSize();

CH376ByteLocate( 0 );

CH376ByteRead(buf,len,NULL);

printf( "%s \n", buf);

mStopIfError ( s );

printf( "文件被关闭 --- Close\n" );

s = CH376FileClose(TRUE);

mStopIfError( s );

}

else

mStopIfError( s );

}

else

{

/\* 找到文件或者出错 \*/

mStopIfError( s );

}

UnknownUsbDevice:

printf( "设备拔出 --- Take out\n" );

while ( CH376DiskConnect( ) == USB\_INT\_SUCCESS )

{

/\* 检查U盘是否连接,等待U盘拔出 \*/

mDelaymS( 100 );

}

mDelaymS( 200 );

}

}

SPI\_SW.C 文件

/\* CH376芯片 软件模拟SPI串行连接的硬件抽象层 V1.0 \*/

/\* 提供I/O接口子程序 \*/

#include "HAL.H"

/\* 本例中的硬件连接方式如下(实际应用电路可以参照修改下述定义及子程序) \*/

/\* 单片机的引脚 CH376芯片的引脚

P1.4 SCS

P1.5 SDI

P1.6 SDO

P1.7 SCK \*/

sbit P24 = P2^4;

sbit P25 = P2^5;

sbit P26 = P2^6;

sbit P27 = P2^7;

#define CH376\_SPI\_SCS P24 /\* 假定CH376的SCS引脚 \*/

#define CH376\_SPI\_SDI P25 /\* 假定CH376的SDI引脚 \*/

#define CH376\_SPI\_SDO P26 /\* 假定CH376的SDO引脚 \*/

#define CH376\_SPI\_SCK P27 /\* 假定CH376的SCK引脚 \*/

#define CH376\_INT\_WIRE INT0 /\* 假定CH376的INT#引脚,如果未连接那么也可以通过查询兼做中断输出的SDO引脚状态实现 \*/

void CH376\_PORT\_INIT( void ) /\* 由于使用软件模拟SPI读写时序,所以进行初始化 \*/

{

/\* 如果是硬件SPI接口,那么可使用mode3(CPOL=1&CPHA=1)或mode0(CPOL=0&CPHA=0),CH376在时钟上升沿采样输入,下降沿输出,数据位是高位在前 \*/

CH376\_SPI\_SCS = 1; /\* 禁止SPI片选 \*/

CH376\_SPI\_SCK = 1; /\* 默认为高电平,SPI模式3,也可以用SPI模式0,但模拟程序可能需稍做修改 \*/

/\* 对于双向I/O引脚模拟SPI接口,那么必须在此设置SPI\_SCS,SPI\_SCK,SPI\_SDI为输出方向,SPI\_SDO为输入方向 \*/

}

void mDelay0\_5uS( void ) /\* 至少延时0.5uS,根据单片机主频调整 \*/

{

}

void Spi376OutByte( UINT8 d ) /\* SPI输出8个位数据 \*/

{

/\* 如果是硬件SPI接口,应该是先将数据写入SPI数据寄存器,然后查询SPI状态寄存器以等待SPI字节传输完成 \*/

UINT8 i;

for ( i = 0; i < 8; i ++ )

{

CH376\_SPI\_SCK = 0;

if ( d & 0x80 )

CH376\_SPI\_SDI = 1;

else

CH376\_SPI\_SDI = 0;

d <<= 1; /\* 数据位是高位在前 \*/

CH376\_SPI\_SCK = 1; /\* CH376在时钟上升沿采样输入 \*/

}

}

UINT8 Spi376InByte( void ) /\* SPI输入8个位数据 \*/

{

/\* 如果是硬件SPI接口,应该是先查询SPI状态寄存器以等待SPI字节传输完成,然后从SPI数据寄存器读出数据 \*/

UINT8 i, d;

d = 0;

for ( i = 0; i < 8; i ++ )

{

CH376\_SPI\_SCK = 0; /\* CH376在时钟下降沿输出 \*/

d <<= 1; /\* 数据位是高位在前 \*/

if ( CH376\_SPI\_SDO )

d ++;

CH376\_SPI\_SCK = 1;

}

return( d );

}

#define xEndCH376Cmd( ) { CH376\_SPI\_SCS = 1; } /\* SPI片选无效,结束CH376命令,仅用于SPI接口方式 \*/

void xWriteCH376Cmd( UINT8 mCmd ) /\* 向CH376写命令 \*/

{

CH376\_SPI\_SCS = 1; /\* 防止之前未通过xEndCH376Cmd禁止SPI片选 \*/

mDelay0\_5uS( );

/\* 对于双向I/O引脚模拟SPI接口,那么必须确保已经设置SPI\_SCS,SPI\_SCK,SPI\_SDI为输出方向,SPI\_SDO为输入方向 \*/

CH376\_SPI\_SCS = 0; /\* SPI片选有效 \*/

Spi376OutByte( mCmd ); /\* 发出命令码 \*/

/\* 延时1.5uS确保读写周期大于1.5uS,或者用上面一行的状态查询代替 \*/

mDelay0\_5uS( );

mDelay0\_5uS( );

mDelay0\_5uS( );

}

#ifdef FOR\_LOW\_SPEED\_MCU /\* 不需要延时 \*/

#define xWriteCH376Data( d ) { Spi376OutByte( d ); } /\* 向CH376写数据 \*/

#define xReadCH376Data( ) ( Spi376InByte( ) ) /\* 从CH376读数据 \*/

#else

void xWriteCH376Data( UINT8 mData ) /\* 向CH376写数据 \*/

{

Spi376OutByte( mData );

}

UINT8 xReadCH376Data( void ) /\* 从CH376读数据 \*/

{

return( Spi376InByte( ) );

}

#endif

/\* 查询CH376中断(INT#低电平) \*/

UINT8 Query376Interrupt( void )

{

/\* 如果连接了CH376的中断引脚则直接查询中断引脚 \*/

return( CH376\_INT\_WIRE ? FALSE : TRUE );

}

UINT8 mInitCH376Host( void ) /\* 初始化CH376 \*/

{

UINT8 res;

CH376\_PORT\_INIT( ); /\* 接口硬件初始化 \*/

xWriteCH376Cmd( CMD11\_CHECK\_EXIST ); /\* 测试单片机与CH376之间的通讯接口 \*/

xWriteCH376Data( 0x65 ); //0110 0101

res = xReadCH376Data( );

xEndCH376Cmd( );

if ( res != 0x9A ) //1001 1010

return( ERR\_USB\_UNKNOWN ); /\* 通讯接口不正常,可能原因有:接口连接异常,其它设备影响(片选不唯一),串口波特率,一直在复位,晶振不工作 \*/

xWriteCH376Cmd( CMD11\_SET\_USB\_MODE ); /\* 设备USB工作模式 \*/

xWriteCH376Data( 0x06 );

mDelayuS( 20 );

/\*--- 设置CH376工作模式之后，读取其结果状态 ---\*/

res = xReadCH376Data( );

xEndCH376Cmd( );

if ( res == CMD\_RET\_SUCCESS )

return( USB\_INT\_SUCCESS );

else

return( ERR\_USB\_UNKNOWN ); /\* 设置模式错误 \*/

}

FILE\_SYS.H文件

/\* CH376芯片 文件系统层 V1.0 \*/

/\* 提供文件系统常用子程序,提供命令打包 \*/

/\* 不使用的子程序可以注释掉,从而节约单片机的程序ROM空间和数据RAM空间 \*/

/\* 这里的子程序是通过括号中的变量传递参数,如果参数较多,为了节约RAM,也可以参考CH375子程序库改成通过同一全局变量/联合结构CH376\_CMD\_DATA传递 \*/

/\* name 参数是指短文件名, 可以包括根目录符, 但不含有路径分隔符, 总长度不超过1+8+1+3+1字节 \*/

/\* PathName 参数是指全路径的短文件名, 包括根目录符、多级子目录及路径分隔符、文件名/目录名 \*/

/\* LongName 参数是指长文件名, 以UNICODE小端顺序编码, 以两个0字节结束, 使用长文件名子程序必须先定义全局缓冲区GlobalBuf, 长度不小于64字节, 可以与其它子程序共用 \*/

/\* 定义 NO\_DEFAULT\_CH376\_INT 用于禁止默认的Wait376Interrupt子程序,禁止后,应用程序必须自行定义一个同名子程序 \*/

/\* 定义 DEF\_INT\_TIMEOUT 用于设置默认的Wait376Interrupt子程序中的等待中断的超时时间/循环计数值, 0则不检查超时而一直等待 \*/

/\* 定义 EN\_DIR\_CREATE 用于提供新建多级子目录的子程序,默认是不提供 \*/

/\* 定义 EN\_DISK\_QUERY 用于提供磁盘容量查询和剩余空间查询的子程序,默认是不提供 \*/

/\* 定义 EN\_SECTOR\_ACCESS 用于提供以扇区为单位读写文件的子程序,默认是不提供 \*/

/\* 定义 EN\_LONG\_NAME 用于提供支持长文件名的子程序,默认是不提供 \*/

#ifndef \_\_CH376\_FS\_H\_\_

#define \_\_CH376\_FS\_H\_\_

#include "HAL.H"

#define STRUCT\_OFFSET( s, m ) ( (UINT8)( & ((s \*)0) -> m ) ) /\* 定义获取结构成员相对偏移地址的宏 \*/

#ifdef EN\_LONG\_NAME

#ifndef LONG\_NAME\_BUF\_LEN

#define LONG\_NAME\_BUF\_LEN ( LONG\_NAME\_PER\_DIR \* 20 ) /\* 自行定义的长文件名缓冲区长度,最小值为LONG\_NAME\_PER\_DIR\*1 \*/

#endif

#endif

UINT8 CH376ReadBlock( PUINT8 buf ); /\* 从当前主机端点的接收缓冲区读取数据块,返回长度 \*/

UINT8 CH376WriteReqBlock( PUINT8 buf ); /\* 向内部指定缓冲区写入请求的数据块,返回长度 \*/

void CH376WriteHostBlock( PUINT8 buf, UINT8 len ); /\* 向USB主机端点的发送缓冲区写入数据块 \*/

void CH376WriteOfsBlock( PUINT8 buf, UINT8 ofs, UINT8 len ); /\* 向内部缓冲区指定偏移地址写入数据块 \*/

void CH376SetFileName( PUINT8 name ); /\* 设置将要操作的文件的文件名 \*/

UINT32 CH376Read32bitDat( void ); /\* 从CH376芯片读取32位的数据并结束命令 \*/

UINT8 CH376ReadVar8( UINT8 var ); /\* 读CH376芯片内部的8位变量 \*/

void CH376WriteVar8( UINT8 var, UINT8 dat ); /\* 写CH376芯片内部的8位变量 \*/

UINT32 CH376ReadVar32( UINT8 var ); /\* 读CH376芯片内部的32位变量 \*/

void CH376WriteVar32( UINT8 var, UINT32 dat ); /\* 写CH376芯片内部的32位变量 \*/

void CH376EndDirInfo( void ); /\* 在调用CH376DirInfoRead获取FAT\_DIR\_INFO结构之后应该通知CH376结束 \*/

UINT32 CH376GetFileSize( void ); /\* 读取当前文件长度 \*/

UINT8 CH376GetDiskStatus( void ); /\* 获取磁盘和文件系统的工作状态 \*/

UINT8 CH376GetIntStatus( void ); /\* 获取中断状态并取消中断请求 \*/

#ifndef NO\_DEFAULT\_CH376\_INT

UINT8 Wait376Interrupt( void ); /\* 等待CH376中断(INT#低电平)，返回中断状态码, 超时则返回ERR\_USB\_UNKNOWN \*/

#endif

UINT8 CH376SendCmdWaitInt( UINT8 mCmd ); /\* 发出命令码后,等待中断 \*/

UINT8 CH376SendCmdDatWaitInt( UINT8 mCmd, UINT8 mDat ); /\* 发出命令码和一字节数据后,等待中断 \*/

UINT8 CH376DiskReqSense( void ); /\* 检查USB存储器错误 \*/

UINT8 CH376DiskConnect( void ); /\* 检查U盘是否连接,不支持SD卡 \*/

UINT8 CH376DiskMount( void ); /\* 初始化磁盘并测试磁盘是否就绪 \*/

UINT8 CH376FileOpen( PUINT8 name ); /\* 在根目录或者当前目录下打开文件或者目录(文件夹) \*/

UINT8 CH376FileCreate( PUINT8 name ); /\* 在根目录或者当前目录下新建文件,如果文件已经存在那么先删除 \*/

UINT8 CH376DirCreate( PUINT8 name ); /\* 在根目录下新建目录(文件夹)并打开,如果目录已经存在那么直接打开 \*/

UINT8 CH376SeparatePath( PUINT8 path ); /\* 从路径中分离出最后一级文件名或者目录(文件夹)名,返回最后一级文件名或者目录名的字节偏移 \*/

UINT8 CH376FileOpenDir( PUINT8 PathName, UINT8 StopName ); /\* 打开多级目录下的文件或者目录的上级目录,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

/\* StopName 指向最后一级文件名或者目录名 \*/

UINT8 CH376FileOpenPath( PUINT8 PathName ); /\* 打开多级目录下的文件或者目录(文件夹),支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

UINT8 CH376FileCreatePath( PUINT8 PathName ); /\* 新建多级目录下的文件,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

#ifdef EN\_DIR\_CREATE

UINT8 CH376DirCreatePath( PUINT8 PathName ); /\* 新建多级目录下的目录(文件夹)并打开,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

#endif

UINT8 CH376FileErase( PUINT8 PathName ); /\* 删除文件,如果已经打开则直接删除,否则对于文件会先打开再删除,支持多级目录路径 \*/

UINT8 CH376FileClose( UINT8 UpdateSz ); /\* 关闭当前已经打开的文件或者目录(文件夹) \*/

UINT8 CH376DirInfoRead( void ); /\* 读取当前文件的目录信息 \*/

UINT8 CH376DirInfoSave( void ); /\* 保存文件的目录信息 \*/

UINT8 CH376ByteLocate( UINT32 offset ); /\* 以字节为单位移动当前文件指针 \*/

UINT8 CH376ByteRead( PUINT8 buf, UINT16 ReqCount, PUINT16 RealCount ); /\* 以字节为单位从当前位置读取数据块 \*/

UINT8 CH376ByteWrite( PUINT8 buf, UINT16 ReqCount, PUINT16 RealCount ); /\* 以字节为单位向当前位置写入数据块 \*/

#ifdef EN\_DISK\_QUERY

UINT8 CH376DiskCapacity( PUINT32 DiskCap ); /\* 查询磁盘物理容量,扇区数 \*/

UINT8 CH376DiskQuery( PUINT32 DiskFre ); /\* 查询磁盘剩余空间信息,扇区数 \*/

#endif

UINT8 CH376SecLocate( UINT32 offset ); /\* 以扇区为单位移动当前文件指针 \*/

#ifdef EN\_SECTOR\_ACCESS

UINT8 CH376DiskReadSec( PUINT8 buf, UINT32 iLbaStart, UINT8 iSectorCount ); /\* 从U盘读取多个扇区的数据块到缓冲区,不支持SD卡 \*/

UINT8 CH376DiskWriteSec( PUINT8 buf, UINT32 iLbaStart, UINT8 iSectorCount ); /\* 将缓冲区中的多个扇区的数据块写入U盘,不支持SD卡 \*/

UINT8 CH376SecRead( PUINT8 buf, UINT8 ReqCount, PUINT8 RealCount ); /\* 以扇区为单位从当前位置读取数据块,不支持SD卡 \*/

UINT8 CH376SecWrite( PUINT8 buf, UINT8 ReqCount, PUINT8 RealCount ); /\* 以扇区为单位在当前位置写入数据块,不支持SD卡 \*/

#endif

#ifdef EN\_LONG\_NAME

UINT8 CH376LongNameWrite( PUINT8 buf, UINT16 ReqCount ); /\* 长文件名专用的字节写子程序 \*/

UINT8 CH376CheckNameSum( PUINT8 DirName ); /\* 计算长文件名的短文件名检验和,输入为无小数点分隔符的固定11字节格式 \*/

UINT8 CH376LocateInUpDir( PUINT8 PathName ); /\* 在上级目录(文件夹)中移动文件指针到当前文件目录信息所在的扇区 \*/

/\* 另外,顺便将当前文件目录信息所在的扇区的前一个扇区的LBA地址写入CH376内部VAR\_FAT\_DIR\_LBA变量(为了方便收集长文件名时向前搜索,否则要多移动一次) \*/

/\* 使用了全局缓冲区GlobalBuf的前12个字节 \*/

UINT8 CH376GetLongName( PUINT8 PathName, PUINT8 LongName ); /\* 由短文件名或者目录(文件夹)名获得相应的长文件名 \*/

/\* 需要输入短文件名的完整路径PathName,需要提供缓冲区接收长文件名LongName(以UNICODE小端编码,以双0结束) \*/

/\* 使用了全局缓冲区GlobalBuf的前34个字节,sizeof(GlobalBuf)>=sizeof(FAT\_DIR\_INFO)+2 \*/

UINT8 CH376CreateLongName( PUINT8 PathName, PUINT8 LongName ); /\* 新建具有长文件名的文件,关闭文件后返回,LongName输入路径必须在RAM中 \*/

/\* 需要输入短文件名的完整路径PathName(请事先参考FAT规范由长文件名自行产生),需要输入以UNICODE小端编码的以双0结束的长文件名LongName \*/

/\* 使用了全局缓冲区GlobalBuf的前64个字节,sizeof(GlobalBuf)>=sizeof(FAT\_DIR\_INFO)\*2 \*/

#endif

#endif

FILE\_SYS.C文件

/\* CH376芯片 文件系统层 V1.1 \*/

/\* 提供文件系统常用子程序,提供命令打包 \*/

/\* 不使用的子程序可以注释掉,从而节约单片机的程序ROM空间和数据RAM空间 \*/

/\* 这里的子程序是通过括号中的变量传递参数,如果参数较多,为了节约RAM,也可以参考CH375子程序库改成通过同一全局变量/联合结构CH376\_CMD\_DATA传递 \*/

/\* name 参数是指短文件名, 可以包括根目录符, 但不含有路径分隔符, 总长度不超过1+8+1+3+1字节 \*/

/\* PathName 参数是指全路径的短文件名, 包括根目录符、多级子目录及路径分隔符、文件名/目录名 \*/

/\* LongName 参数是指长文件名, 以UNICODE小端顺序编码, 以两个0字节结束, 使用长文件名子程序必须先定义全局缓冲区GlobalBuf, 长度不小于64字节, 可以与其它子程序共用 \*/

/\* 定义 NO\_DEFAULT\_CH376\_INT 用于禁止默认的Wait376Interrupt子程序,禁止后,应用程序必须自行定义一个同名子程序 \*/

/\* 定义 DEF\_INT\_TIMEOUT 用于设置默认的Wait376Interrupt子程序中的等待中断的超时时间/循环计数值, 0则不检查超时而一直等待 \*/

/\* 定义 EN\_DIR\_CREATE 用于提供新建多级子目录的子程序,默认是不提供 \*/

/\* 定义 EN\_DISK\_QUERY 用于提供磁盘容量查询和剩余空间查询的子程序,默认是不提供 \*/

/\* 定义 EN\_SECTOR\_ACCESS 用于提供以扇区为单位读写文件的子程序,默认是不提供 \*/

/\* 定义 EN\_LONG\_NAME 用于提供支持长文件名的子程序,默认是不提供 \*/

#include "FILE\_SYS.H"

UINT8 CH376ReadBlock( PUINT8 buf ) /\* 从当前主机端点的接收缓冲区读取数据块,返回长度 \*/

{

UINT8 s, l;

xWriteCH376Cmd( CMD01\_RD\_USB\_DATA0 );

s = l = xReadCH376Data( ); /\* 长度 \*/

if ( l )

{

do

{

\*buf = xReadCH376Data( );

buf ++;

}while ( -- l );

}

xEndCH376Cmd( );

return( s );

}

UINT8 CH376WriteReqBlock( PUINT8 buf ) /\* 向内部指定缓冲区写入请求的数据块,返回长度 \*/

{

UINT8 s, l;

xWriteCH376Cmd( CMD01\_WR\_REQ\_DATA );

s = l = xReadCH376Data( ); /\* 长度 \*/

if ( l ) {

do {

xWriteCH376Data( \*buf );

buf ++;

} while ( -- l );

}

xEndCH376Cmd( );

return( s );

}

void CH376WriteHostBlock( PUINT8 buf, UINT8 len ) /\* 向USB主机端点的发送缓冲区写入数据块 \*/

{

xWriteCH376Cmd( CMD10\_WR\_HOST\_DATA );

xWriteCH376Data( len ); /\* 长度 \*/

if ( len ) {

do {

xWriteCH376Data( \*buf );

buf ++;

} while ( -- len );

}

xEndCH376Cmd( );

}

void CH376WriteOfsBlock( PUINT8 buf, UINT8 ofs, UINT8 len ) /\* 向内部缓冲区指定偏移地址写入数据块 \*/

{

xWriteCH376Cmd( CMD20\_WR\_OFS\_DATA );

xWriteCH376Data( ofs ); /\* 偏移地址 \*/

xWriteCH376Data( len ); /\* 长度 \*/

if ( len ) {

do {

xWriteCH376Data( \*buf );

buf ++;

} while ( -- len );

}

xEndCH376Cmd( );

}

void CH376SetFileName( PUINT8 name ) /\* 设置将要操作的文件的文件名 \*/

{

UINT8 c;

xWriteCH376Cmd( CMD10\_SET\_FILE\_NAME );

c = \*name;

xWriteCH376Data( c );

while ( c )

{

name ++;

c = \*name;

if ( c == DEF\_SEPAR\_CHAR1 || c == DEF\_SEPAR\_CHAR2 )

c = 0; /\* 强行将文件名截止 \*/

xWriteCH376Data( c );

}

xEndCH376Cmd( );

}

UINT32 CH376Read32bitDat( void ) /\* 从CH376芯片读取32位的数据并结束命令 \*/

{

UINT8 c0, c1, c2, c3;

c0 = xReadCH376Data( );

c1 = xReadCH376Data( );

c2 = xReadCH376Data( );

c3 = xReadCH376Data( );

xEndCH376Cmd( );

return( c0 | (UINT16)c1 << 8 | (UINT32)c2 << 16 | (UINT32)c3 << 24 );

}

UINT8 CH376ReadVar8( UINT8 var ) /\* 读CH376芯片内部的8位变量 \*/

{

UINT8 c0;

xWriteCH376Cmd( CMD11\_READ\_VAR8 );

xWriteCH376Data( var );

c0 = xReadCH376Data( );

xEndCH376Cmd( );

return( c0 );

}

void CH376WriteVar8( UINT8 var, UINT8 dat ) /\* 写CH376芯片内部的8位变量 \*/

{

xWriteCH376Cmd( CMD20\_WRITE\_VAR8 );

xWriteCH376Data( var );

xWriteCH376Data( dat );

xEndCH376Cmd( );

}

UINT32 CH376ReadVar32( UINT8 var ) /\* 读CH376芯片内部的32位变量 \*/

{

xWriteCH376Cmd( CMD14\_READ\_VAR32 );

xWriteCH376Data( var );

return( CH376Read32bitDat( ) ); /\* 从CH376芯片读取32位的数据并结束命令 \*/

}

void CH376WriteVar32( UINT8 var, UINT32 dat ) /\* 写CH376芯片内部的32位变量 \*/

{

xWriteCH376Cmd( CMD50\_WRITE\_VAR32 );

xWriteCH376Data( var );

xWriteCH376Data( (UINT8)dat );

xWriteCH376Data( (UINT8)( (UINT16)dat >> 8 ) );

xWriteCH376Data( (UINT8)( dat >> 16 ) );

xWriteCH376Data( (UINT8)( dat >> 24 ) );

xEndCH376Cmd( );

}

void CH376EndDirInfo( void ) /\* 在调用CH376DirInfoRead获取FAT\_DIR\_INFO结构之后应该通知CH376结束 \*/

{

CH376WriteVar8( 0x0D, 0x00 );

}

UINT32 CH376GetFileSize( void ) /\* 读取当前文件长度 \*/

{

return( CH376ReadVar32( VAR\_FILE\_SIZE ) );

}

UINT8 CH376GetDiskStatus( void ) /\* 获取磁盘和文件系统的工作状态 \*/

{

return( CH376ReadVar8( VAR\_DISK\_STATUS ) );

}

UINT8 CH376GetIntStatus( void ) /\* 获取中断状态并取消中断请求 \*/

{

UINT8 s;

xWriteCH376Cmd( CMD01\_GET\_STATUS );

s = xReadCH376Data( );

xEndCH376Cmd( );

return( s );

}

UINT8 Wait376Interrupt( void ) /\* 等待CH376中断(INT#低电平)，返回中断状态码, 超时则返回ERR\_USB\_UNKNOWN \*/

{

UINT32 i;

for ( i = 0; i < 5000000; i ++ )

{

/\* 计数防止超时,默认的超时时间,与单片机主频有关 \*/

if ( Query376Interrupt( ) )

return( CH376GetIntStatus( ) ); /\* 检测到中断 \*/

/\* 在等待CH376中断的过程中,可以做些需要及时处理的其它事情 \*/

}

return( ERR\_USB\_UNKNOWN ); /\* 不应该发生的情况 \*/

}

UINT8 CH376SendCmdWaitInt( UINT8 mCmd ) /\* 发出命令码后,等待中断 \*/

{

xWriteCH376Cmd( mCmd );

xEndCH376Cmd( );

return( Wait376Interrupt( ) );

}

UINT8 CH376SendCmdDatWaitInt( UINT8 mCmd, UINT8 mDat ) /\* 发出命令码和一字节数据后,等待中断 \*/

{

xWriteCH376Cmd( mCmd );

xWriteCH376Data( mDat );

xEndCH376Cmd( );

return( Wait376Interrupt( ) );

}

UINT8 CH376DiskReqSense( void ) /\* 检查USB存储器错误 \*/

{

UINT8 s;

mDelaymS( 5 );

s = CH376SendCmdWaitInt( CMD0H\_DISK\_R\_SENSE );

mDelaymS( 5 );

return( s );

}

UINT8 CH376DiskConnect( void ) /\* 检查U盘是否连接,不支持SD卡 \*/

{

if( Query376Interrupt( ) ) /\*--- 就是简单的查询中断口的电平状态 ---\*/

CH376GetIntStatus( ); /\* 检测到中断 \*/

return( CH376SendCmdWaitInt( CMD0H\_DISK\_CONNECT ) );

}

UINT8 CH376DiskMount( void ) /\* 初始化磁盘并测试磁盘是否就绪 \*/

{

return( CH376SendCmdWaitInt( CMD0H\_DISK\_MOUNT ) );

}

UINT8 CH376FileOpen( PUINT8 name ) /\* 在根目录或者当前目录下打开文件或者目录(文件夹) \*/

{

CH376SetFileName( name ); /\* 设置将要操作的文件的文件名 \*/

if ( name[0] == DEF\_SEPAR\_CHAR1 || name[0] == DEF\_SEPAR\_CHAR2 )

CH376WriteVar32( VAR\_CURRENT\_CLUST, 0 );

return( CH376SendCmdWaitInt( CMD0H\_FILE\_OPEN ) );

}

UINT8 CH376FileCreate( PUINT8 name ) /\* 在根目录或者当前目录下新建文件,如果文件已经存在那么先删除 \*/

{

if ( name ) CH376SetFileName( name ); /\* 设置将要操作的文件的文件名 \*/

return( CH376SendCmdWaitInt( CMD0H\_FILE\_CREATE ) );

}

UINT8 CH376DirCreate( PUINT8 name ) /\* 在根目录下新建目录(文件夹)并打开,如果目录已经存在那么直接打开 \*/

{

CH376SetFileName( name ); /\* 设置将要操作的文件的文件名 \*/

if ( name[0] == DEF\_SEPAR\_CHAR1 || name[0] == DEF\_SEPAR\_CHAR2 ) CH376WriteVar32( VAR\_CURRENT\_CLUST, 0 );

return( CH376SendCmdWaitInt( CMD0H\_DIR\_CREATE ) );

}

UINT8 CH376SeparatePath( PUINT8 path ) /\* 从路径中分离出最后一级文件名或者目录(文件夹)名,返回最后一级文件名或者目录名的字节偏移 \*/

{

PUINT8 pName;

for ( pName = path; \*pName != 0; ++ pName ); /\* 到文件名字符串结束位置 \*/

while ( \*pName != DEF\_SEPAR\_CHAR1 && \*pName != DEF\_SEPAR\_CHAR2 && pName != path ) pName --; /\* 搜索倒数第一个路径分隔符 \*/

if ( pName != path ) pName ++; /\* 找到了路径分隔符,则修改指向目标文件的最后一级文件名,跳过前面的多级目录名及路径分隔符 \*/

return( pName - path );

}

/\* 打开多级目录下的文件或者目录的上级目录,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

UINT8 CH376FileOpenDir( PUINT8 PathName, UINT8 StopName )

/\* StopName 指向最后一级文件名或者目录名 \*/

{

UINT8 i, s;

s = 0;

i = 1; /\* 跳过有可能的根目录符 \*/

while ( 1 )

{

while ( PathName[i] != DEF\_SEPAR\_CHAR1 && PathName[i] != DEF\_SEPAR\_CHAR2 && PathName[i] != 0 )

++ i; /\* 搜索下一个路径分隔符或者路径结束符 \*/

if ( PathName[i] )

i ++; /\* 找到了路径分隔符,修改指向目标文件的最后一级文件名 \*/

else

i = 0; /\* 路径结束 \*/

s = CH376FileOpen( &PathName[s] ); /\* 打开文件或者目录 \*/

if ( i && i != StopName )

{

/\* 路径尚未结束 \*/

if ( s != ERR\_OPEN\_DIR )

{

/\* 因为是逐级打开,尚未到路径结束,所以,如果不是成功打开了目录,那么说明有问题 \*/

if ( s == USB\_INT\_SUCCESS )

return( ERR\_FOUND\_NAME ); /\* 中间路径必须是目录名,如果是文件名则出错 \*/

else

if ( s == ERR\_MISS\_FILE )

return( ERR\_MISS\_DIR ); /\* 中间路径的某个子目录没有找到,可能是目录名称错误 \*/

else

return( s ); /\* 操作出错 \*/

}

s = i; /\* 从下一级目录开始继续 \*/

}

else

return( s ); /\* 路径结束,USB\_INT\_SUCCESS为成功打开文件,ERR\_OPEN\_DIR为成功打开目录(文件夹),其它为操作出错 \*/

}

}

UINT8 CH376FileOpenPath( PUINT8 PathName ) /\* 打开多级目录下的文件或者目录(文件夹),支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

{

return( CH376FileOpenDir( PathName, 0xFF ) );

}

UINT8 CH376FileCreatePath( PUINT8 PathName ) /\* 新建多级目录下的文件,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

{

UINT8 s;

UINT8 Name;

Name = CH376SeparatePath( PathName ); /\* 从路径中分离出最后一级文件名,返回最后一级文件名的偏移 \*/

if ( Name ) { /\* 是多级目录 \*/

s = CH376FileOpenDir( PathName, Name ); /\* 打开多级目录下的最后一级目录,即打开新建文件的上级目录 \*/

if ( s != ERR\_OPEN\_DIR ) { /\* 因为是打开上级目录,所以,如果不是成功打开了目录,那么说明有问题 \*/

if ( s == USB\_INT\_SUCCESS ) return( ERR\_FOUND\_NAME ); /\* 中间路径必须是目录名,如果是文件名则出错 \*/

else if ( s == ERR\_MISS\_FILE ) return( ERR\_MISS\_DIR ); /\* 中间路径的某个子目录没有找到,可能是目录名称错误 \*/

else return( s ); /\* 操作出错 \*/

}

}

return( CH376FileCreate( &PathName[Name] ) ); /\* 在根目录或者当前目录下新建文件 \*/

}

#ifdef EN\_DIR\_CREATE

UINT8 CH376DirCreatePath( PUINT8 PathName ) /\* 新建多级目录下的目录(文件夹)并打开,支持多级目录路径,支持路径分隔符,路径长度不超过255个字符 \*/

{

UINT8 s;

UINT8 Name;

UINT8 ClustBuf[4];

Name = CH376SeparatePath( PathName ); /\* 从路径中分离出最后一级目录名,返回最后一级文件名的偏移 \*/

if ( Name ) { /\* 是多级目录 \*/

s = CH376FileOpenDir( PathName, Name ); /\* 打开多级目录下的最后一级目录,即打开新建目录的上级目录 \*/

if ( s != ERR\_OPEN\_DIR ) { /\* 因为是打开上级目录,所以,如果不是成功打开了目录,那么说明有问题 \*/

if ( s == USB\_INT\_SUCCESS ) return( ERR\_FOUND\_NAME ); /\* 中间路径必须是目录名,如果是文件名则出错 \*/

else if ( s == ERR\_MISS\_FILE ) return( ERR\_MISS\_DIR ); /\* 中间路径的某个子目录没有找到,可能是目录名称错误 \*/

else return( s ); /\* 操作出错 \*/

}

xWriteCH376Cmd( CMD14\_READ\_VAR32 );

xWriteCH376Data( VAR\_START\_CLUSTER ); /\* 上级目录的起始簇号 \*/

for ( s = 0; s != 4; s ++ ) ClustBuf[ s ] = xReadCH376Data( );

xEndCH376Cmd( );

s = CH376DirCreate( &PathName[Name] ); /\* 在当前目录下新建目录 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

s = CH376ByteLocate( sizeof(FAT\_DIR\_INFO) + STRUCT\_OFFSET( FAT\_DIR\_INFO, DIR\_FstClusHI ) ); /\* 移动文件指针 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

s = CH376ByteWrite( &ClustBuf[2], 2, NULL ); /\* 写入上级目录的起始簇号的高16位 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

s = CH376ByteLocate( sizeof(FAT\_DIR\_INFO) + STRUCT\_OFFSET( FAT\_DIR\_INFO, DIR\_FstClusLO ) ); /\* 移动文件指针 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

s = CH376ByteWrite( ClustBuf, 2, NULL ); /\* 写入上级目录的起始簇号的低16位 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

s = CH376ByteLocate( 0 ); /\* 移动文件指针,恢复到目录头位置 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

}

else { /\* 不是多级目录 \*/

if ( PathName[0] == DEF\_SEPAR\_CHAR1 || PathName[0] == DEF\_SEPAR\_CHAR2 ) return( CH376DirCreate( PathName ) ); /\* 在根目录下新建目录 \*/

else return( ERR\_MISS\_DIR ); /\* 必须提供完整路径才能实现在当前目录下新建目录 \*/

}

}

#endif

UINT8 CH376FileErase( PUINT8 PathName ) /\* 删除文件,如果已经打开则直接删除,否则对于文件会先打开再删除,支持多级目录路径 \*/

{

UINT8 s;

if ( PathName ) { /\* 文件尚未打开 \*/

for ( s = 1; PathName[s] != DEF\_SEPAR\_CHAR1 && PathName[s] != DEF\_SEPAR\_CHAR2 && PathName[s] != 0; ++ s ); /\* 搜索下一个路径分隔符或者路径结束符 \*/

if ( PathName[s] ) { /\* 有路径分隔符,是多级目录下的文件或者目录 \*/

s = CH376FileOpenPath( PathName ); /\* 打开多级目录下的文件或者目录 \*/

if ( s != USB\_INT\_SUCCESS && s != ERR\_OPEN\_DIR ) return( s ); /\* 操作出错 \*/

}

else CH376SetFileName( PathName ); /\* 没有路径分隔符,是根目录或者当前目录下的文件或者目录,设置将要操作的文件的文件名 \*/

}

return( CH376SendCmdWaitInt( CMD0H\_FILE\_ERASE ) );

}

UINT8 CH376FileClose( UINT8 UpdateSz ) /\* 关闭当前已经打开的文件或者目录(文件夹) \*/

{

return( CH376SendCmdDatWaitInt( CMD1H\_FILE\_CLOSE, UpdateSz ) );

}

UINT8 CH376DirInfoRead( void ) /\* 读取当前文件的目录信息 \*/

{

return( CH376SendCmdDatWaitInt( CMD1H\_DIR\_INFO\_READ, 0xFF ) );

}

UINT8 CH376DirInfoSave( void ) /\* 保存文件的目录信息 \*/

{

return( CH376SendCmdWaitInt( CMD0H\_DIR\_INFO\_SAVE ) );

}

UINT8 CH376ByteLocate( UINT32 offset ) /\* 以字节为单位移动当前文件指针 \*/

{

xWriteCH376Cmd( CMD4H\_BYTE\_LOCATE );

xWriteCH376Data( (UINT8)offset );

xWriteCH376Data( (UINT8)((UINT16)offset>>8) );

xWriteCH376Data( (UINT8)(offset>>16) );

xWriteCH376Data( (UINT8)(offset>>24) );

xEndCH376Cmd( );

return( Wait376Interrupt( ) );

}

UINT8 CH376ByteRead( PUINT8 buf, UINT16 ReqCount, PUINT16 RealCount ) /\* 以字节为单位从当前位置读取数据块 \*/

{

UINT8 s;

xWriteCH376Cmd( CMD2H\_BYTE\_READ );

xWriteCH376Data( (UINT8)ReqCount );

xWriteCH376Data( (UINT8)(ReqCount>>8) );

xEndCH376Cmd( );

if ( RealCount ) \*RealCount = 0;

while ( 1 ) {

s = Wait376Interrupt( );

if ( s == USB\_INT\_DISK\_READ ) {

s = CH376ReadBlock( buf ); /\* 从当前主机端点的接收缓冲区读取数据块,返回长度 \*/

xWriteCH376Cmd( CMD0H\_BYTE\_RD\_GO );

xEndCH376Cmd( );

buf += s;

if ( RealCount ) \*RealCount += s;

}

/\* else if ( s == USB\_INT\_SUCCESS ) return( s );\*/ /\* 结束 \*/

else return( s ); /\* 错误 \*/

}

}

UINT8 CH376ByteWrite( PUINT8 buf, UINT16 ReqCount, PUINT16 RealCount ) /\* 以字节为单位向当前位置写入数据块 \*/

{

UINT8 s;

xWriteCH376Cmd( CMD2H\_BYTE\_WRITE );

xWriteCH376Data( (UINT8)ReqCount );

xWriteCH376Data( (UINT8)(ReqCount>>8) );

xEndCH376Cmd( );

if ( RealCount ) \*RealCount = 0;

while ( 1 )

{

s = Wait376Interrupt( );

if ( s == USB\_INT\_DISK\_WRITE )

{

s = CH376WriteReqBlock( buf ); /\* 向内部指定缓冲区写入请求的数据块,返回长度 \*/

xWriteCH376Cmd( CMD0H\_BYTE\_WR\_GO );

xEndCH376Cmd( );

buf += s;

if ( RealCount )

\*RealCount += s;

}

else

return( s ); /\* 错误 \*/

}

}

#ifdef EN\_DISK\_QUERY

UINT8 CH376DiskCapacity( PUINT32 DiskCap ) /\* 查询磁盘物理容量,扇区数 \*/

{

UINT8 s;

s = CH376SendCmdWaitInt( CMD0H\_DISK\_CAPACITY );

if ( s == USB\_INT\_SUCCESS ) { /\* 参考CH376INC.H文件中CH376\_CMD\_DATA结构的DiskCapacity \*/

xWriteCH376Cmd( CMD01\_RD\_USB\_DATA0 );

xReadCH376Data( ); /\* 长度总是sizeof(CH376\_CMD\_DATA.DiskCapacity) \*/

\*DiskCap = CH376Read32bitDat( ); /\* CH376\_CMD\_DATA.DiskCapacity.mDiskSizeSec,从CH376芯片读取32位的数据并结束命令 \*/

}

else \*DiskCap = 0;

return( s );

}

UINT8 CH376DiskQuery( PUINT32 DiskFre ) /\* 查询磁盘剩余空间信息,扇区数 \*/

{

UINT8 s;

UINT8 c0, c1, c2, c3;

s = CH376SendCmdWaitInt( CMD0H\_DISK\_QUERY );

if ( s == USB\_INT\_SUCCESS ) { /\* 参考CH376INC.H文件中CH376\_CMD\_DATA结构的DiskQuery \*/

xWriteCH376Cmd( CMD01\_RD\_USB\_DATA0 );

xReadCH376Data( ); /\* 长度总是sizeof(CH376\_CMD\_DATA.DiskQuery) \*/

xReadCH376Data( ); /\* CH376\_CMD\_DATA.DiskQuery.mTotalSector \*/

xReadCH376Data( );

xReadCH376Data( );

xReadCH376Data( );

c0 = xReadCH376Data( ); /\* CH376\_CMD\_DATA.DiskQuery.mFreeSector \*/

c1 = xReadCH376Data( );

c2 = xReadCH376Data( );

c3 = xReadCH376Data( );

\*DiskFre = c0 | (UINT16)c1 << 8 | (UINT32)c2 << 16 | (UINT32)c3 << 24;

xReadCH376Data( ); /\* CH376\_CMD\_DATA.DiskQuery.mDiskFat \*/

xEndCH376Cmd( );

}

else \*DiskFre = 0;

return( s );

}

#endif

UINT8 CH376SecLocate( UINT32 offset ) /\* 以扇区为单位移动当前文件指针 \*/

{

xWriteCH376Cmd( CMD4H\_SEC\_LOCATE );

xWriteCH376Data( (UINT8)offset );

xWriteCH376Data( (UINT8)((UINT16)offset>>8) );

xWriteCH376Data( (UINT8)(offset>>16) );

xWriteCH376Data( 0 ); /\* 超出最大文件尺寸 \*/

xEndCH376Cmd( );

return( Wait376Interrupt( ) );

}

#ifdef EN\_SECTOR\_ACCESS

UINT8 CH376DiskReadSec( PUINT8 buf, UINT32 iLbaStart, UINT8 iSectorCount ) /\* 从U盘读取多个扇区的数据块到缓冲区,不支持SD卡 \*/

/\* iLbaStart 是准备读取的线性起始扇区号, iSectorCount 是准备读取的扇区数 \*/

{

UINT8 s, err;

UINT16 mBlockCount;

for ( err = 0; err != 3; ++ err ) { /\* 出错重试 \*/

xWriteCH376Cmd( CMD5H\_DISK\_READ ); /\* 从USB存储器读扇区 \*/

xWriteCH376Data( (UINT8)iLbaStart ); /\* LBA的最低8位 \*/

xWriteCH376Data( (UINT8)( (UINT16)iLbaStart >> 8 ) );

xWriteCH376Data( (UINT8)( iLbaStart >> 16 ) );

xWriteCH376Data( (UINT8)( iLbaStart >> 24 ) ); /\* LBA的最高8位 \*/

xWriteCH376Data( iSectorCount ); /\* 扇区数 \*/

xEndCH376Cmd( );

for ( mBlockCount = iSectorCount \* DEF\_SECTOR\_SIZE / CH376\_DAT\_BLOCK\_LEN; mBlockCount != 0; -- mBlockCount ) { /\* 数据块计数 \*/

s = Wait376Interrupt( ); /\* 等待中断并获取状态 \*/

if ( s == USB\_INT\_DISK\_READ ) { /\* USB存储器读数据块,请求数据读出 \*/

s = CH376ReadBlock( buf ); /\* 从当前主机端点的接收缓冲区读取数据块,返回长度 \*/

xWriteCH376Cmd( CMD0H\_DISK\_RD\_GO ); /\* 继续执行USB存储器的读操作 \*/

xEndCH376Cmd( );

buf += s;

}

else break; /\* 返回错误状态 \*/

}

if ( mBlockCount == 0 ) {

s = Wait376Interrupt( ); /\* 等待中断并获取状态 \*/

if ( s == USB\_INT\_SUCCESS ) return( USB\_INT\_SUCCESS ); /\* 操作成功 \*/

}

if ( s == USB\_INT\_DISCONNECT ) return( s ); /\* U盘被移除 \*/

CH376DiskReqSense( ); /\* 检查USB存储器错误 \*/

}

return( s ); /\* 操作失败 \*/

}

UINT8 CH376DiskWriteSec( PUINT8 buf, UINT32 iLbaStart, UINT8 iSectorCount ) /\* 将缓冲区中的多个扇区的数据块写入U盘,不支持SD卡 \*/

/\* iLbaStart 是写入的线起始性扇区号, iSectorCount 是写入的扇区数 \*/

{

UINT8 s, err;

UINT16 mBlockCount;

for ( err = 0; err != 3; ++ err ) { /\* 出错重试 \*/

xWriteCH376Cmd( CMD5H\_DISK\_WRITE ); /\* 向USB存储器写扇区 \*/

xWriteCH376Data( (UINT8)iLbaStart ); /\* LBA的最低8位 \*/

xWriteCH376Data( (UINT8)( (UINT16)iLbaStart >> 8 ) );

xWriteCH376Data( (UINT8)( iLbaStart >> 16 ) );

xWriteCH376Data( (UINT8)( iLbaStart >> 24 ) ); /\* LBA的最高8位 \*/

xWriteCH376Data( iSectorCount ); /\* 扇区数 \*/

xEndCH376Cmd( );

for ( mBlockCount = iSectorCount \* DEF\_SECTOR\_SIZE / CH376\_DAT\_BLOCK\_LEN; mBlockCount != 0; -- mBlockCount ) { /\* 数据块计数 \*/

s = Wait376Interrupt( ); /\* 等待中断并获取状态 \*/

if ( s == USB\_INT\_DISK\_WRITE ) { /\* USB存储器写数据块,请求数据写入 \*/

CH376WriteHostBlock( buf, CH376\_DAT\_BLOCK\_LEN ); /\* 向USB主机端点的发送缓冲区写入数据块 \*/

xWriteCH376Cmd( CMD0H\_DISK\_WR\_GO ); /\* 继续执行USB存储器的写操作 \*/

xEndCH376Cmd( );

buf += CH376\_DAT\_BLOCK\_LEN;

}

else break; /\* 返回错误状态 \*/

}

if ( mBlockCount == 0 ) {

s = Wait376Interrupt( ); /\* 等待中断并获取状态 \*/

if ( s == USB\_INT\_SUCCESS ) return( USB\_INT\_SUCCESS ); /\* 操作成功 \*/

}

if ( s == USB\_INT\_DISCONNECT ) return( s ); /\* U盘被移除 \*/

CH376DiskReqSense( ); /\* 检查USB存储器错误 \*/

}

return( s ); /\* 操作失败 \*/

}

UINT8 CH376SecRead( PUINT8 buf, UINT8 ReqCount, PUINT8 RealCount ) /\* 以扇区为单位从当前位置读取数据块,不支持SD卡 \*/

{

UINT8 s;

UINT8 cnt;

UINT32 StaSec;

UINT32 fsz, fofs;

if ( RealCount ) \*RealCount = 0;

do {

xWriteCH376Cmd( CMD01\_GET\_IC\_VER );

cnt = xReadCH376Data( );

if ( cnt == 0x41 ) {

xWriteCH376Cmd( CMD14\_READ\_VAR32 );

xWriteCH376Data( VAR\_FILE\_SIZE );

xReadCH376Data( );

fsz = xReadCH376Data( );

fsz |= (UINT16)(xReadCH376Data( )) << 8;

cnt = xReadCH376Data( );

fsz |= (UINT32)cnt << 16;

xWriteCH376Cmd( CMD14\_READ\_VAR32 );

xWriteCH376Data( VAR\_CURRENT\_OFFSET );

xReadCH376Data( );

fofs = xReadCH376Data( );

fofs |= (UINT16)(xReadCH376Data( )) << 8;

fofs |= (UINT32)(xReadCH376Data( )) << 16;

if ( fsz >= fofs + 510 ) CH376WriteVar8( VAR\_FILE\_SIZE + 3, 0xFF );

else cnt = 0xFF;

}

else cnt = 0xFF;

xWriteCH376Cmd( CMD1H\_SEC\_READ );

xWriteCH376Data( ReqCount );

xEndCH376Cmd( );

s = Wait376Interrupt( );

if ( cnt != 0xFF ) CH376WriteVar8( VAR\_FILE\_SIZE + 3, cnt );

if ( s != USB\_INT\_SUCCESS ) return( s );

xWriteCH376Cmd( CMD01\_RD\_USB\_DATA0 );

xReadCH376Data( ); /\* 长度总是sizeof(CH376\_CMD\_DATA.SectorRead) \*/

cnt = xReadCH376Data( ); /\* CH376\_CMD\_DATA.SectorRead.mSectorCount \*/

xReadCH376Data( );

xReadCH376Data( );

xReadCH376Data( );

StaSec = CH376Read32bitDat( ); /\* CH376\_CMD\_DATA.SectorRead.mStartSector,从CH376芯片读取32位的数据并结束命令 \*/

if ( cnt == 0 ) break;

s = CH376DiskReadSec( buf, StaSec, cnt ); /\* 从U盘读取多个扇区的数据块到缓冲区 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

buf += cnt \* DEF\_SECTOR\_SIZE;

if ( RealCount ) \*RealCount += cnt;

ReqCount -= cnt;

} while ( ReqCount );

return( s );

}

UINT8 CH376SecWrite( PUINT8 buf, UINT8 ReqCount, PUINT8 RealCount ) /\* 以扇区为单位在当前位置写入数据块,不支持SD卡 \*/

{

UINT8 s;

UINT8 cnt;

UINT32 StaSec;

if ( RealCount ) \*RealCount = 0;

do {

xWriteCH376Cmd( CMD1H\_SEC\_WRITE );

xWriteCH376Data( ReqCount );

xEndCH376Cmd( );

s = Wait376Interrupt( );

if ( s != USB\_INT\_SUCCESS ) return( s );

xWriteCH376Cmd( CMD01\_RD\_USB\_DATA0 );

xReadCH376Data( ); /\* 长度总是sizeof(CH376\_CMD\_DATA.SectorWrite) \*/

cnt = xReadCH376Data( ); /\* CH376\_CMD\_DATA.SectorWrite.mSectorCount \*/

xReadCH376Data( );

xReadCH376Data( );

xReadCH376Data( );

StaSec = CH376Read32bitDat( ); /\* CH376\_CMD\_DATA.SectorWrite.mStartSector,从CH376芯片读取32位的数据并结束命令 \*/

if ( cnt == 0 ) break;

s = CH376DiskWriteSec( buf, StaSec, cnt ); /\* 将缓冲区中的多个扇区的数据块写入U盘 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

buf += cnt \* DEF\_SECTOR\_SIZE;

if ( RealCount ) \*RealCount += cnt;

ReqCount -= cnt;

} while ( ReqCount );

return( s );

}

#endif

#ifdef EN\_LONG\_NAME

UINT8 CH376LongNameWrite( PUINT8 buf, UINT16 ReqCount ) /\* 长文件名专用的字节写子程序 \*/

{

UINT8 s, c;

c = CH376ReadVar8( VAR\_DISK\_STATUS );

if ( c == DEF\_DISK\_OPEN\_ROOT ) CH376WriteVar8( VAR\_DISK\_STATUS, DEF\_DISK\_OPEN\_DIR );

xWriteCH376Cmd( CMD2H\_BYTE\_WRITE );

xWriteCH376Data( (UINT8)ReqCount );

xWriteCH376Data( (UINT8)(ReqCount>>8) );

xEndCH376Cmd( );

while ( 1 ) {

s = Wait376Interrupt( );

if ( s == USB\_INT\_DISK\_WRITE ) {

if ( buf ) buf += CH376WriteReqBlock( buf ); /\* 向内部指定缓冲区写入请求的数据块,返回长度 \*/

else {

xWriteCH376Cmd( CMD01\_WR\_REQ\_DATA ); /\* 向内部指定缓冲区写入请求的数据块 \*/

s = xReadCH376Data( ); /\* 长度 \*/

while ( s -- ) xWriteCH376Data( 0 ); /\* 填充0 \*/

}

xWriteCH376Cmd( CMD0H\_BYTE\_WR\_GO );

xEndCH376Cmd( );

}

/\* else if ( s == USB\_INT\_SUCCESS ) return( s );\*/ /\* 结束 \*/

else {

if ( c == DEF\_DISK\_OPEN\_ROOT ) CH376WriteVar8( VAR\_DISK\_STATUS, c );

return( s ); /\* 错误 \*/

}

}

}

UINT8 CH376CheckNameSum( PUINT8 DirName ) /\* 计算长文件名的短文件名检验和,输入为无小数点分隔符的固定11字节格式 \*/

{

UINT8 NameLen;

UINT8 CheckSum;

CheckSum = 0;

for ( NameLen = 0; NameLen != 11; NameLen ++ ) CheckSum = ( CheckSum & 1 ? 0x80 : 0x00 ) + ( CheckSum >> 1 ) + \*DirName++;

return( CheckSum );

}

UINT8 CH376LocateInUpDir( PUINT8 PathName ) /\* 在上级目录(文件夹)中移动文件指针到当前文件目录信息所在的扇区 \*/

/\* 另外,顺便将当前文件目录信息所在的扇区的前一个扇区的LBA地址写入CH376内部VAR\_FAT\_DIR\_LBA变量(为了方便收集长文件名时向前搜索,否则要多移动一次) \*/

/\* 使用了全局缓冲区GlobalBuf的前12个字节 \*/

{

UINT8 s;

xWriteCH376Cmd( CMD14\_READ\_VAR32 );

xWriteCH376Data( VAR\_FAT\_DIR\_LBA ); /\* 当前文件目录信息所在的扇区LBA地址 \*/

for ( s = 4; s != 8; s ++ ) GlobalBuf[ s ] = xReadCH376Data( ); /\* 临时保存于全局缓冲区中,节约RAM \*/

xEndCH376Cmd( );

s = CH376SeparatePath( PathName ); /\* 从路径中分离出最后一级文件名或者目录名,返回最后一级文件名或者目录名的偏移 \*/

if ( s ) s = CH376FileOpenDir( PathName, s ); /\* 是多级目录,打开多级目录下的最后一级目录,即打开文件的上级目录 \*/

else s = CH376FileOpen( "/" ); /\* 根目录下的文件,则打开根目录 \*/

if ( s != ERR\_OPEN\_DIR ) return( s );

\*(PUINT32)(&GlobalBuf[0]) = 0; /\* 目录扇区偏移扇区数,保存在全局缓冲区中,节约RAM \*/

while ( 1 ) { /\* 不断移动文件指针,直到与当前文件目录信息所在的扇区LBA地址匹配 \*/

s = CH376SecLocate( \*(PUINT32)(&GlobalBuf[0]) ); /\* 以扇区为单位在上级目录中移动文件指针 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

CH376ReadBlock( &GlobalBuf[8] ); /\* 从内存缓冲区读取CH376\_CMD\_DATA.SectorLocate.mSectorLba数据块,返回长度总是sizeof(CH376\_CMD\_DATA.SectorLocate) \*/

if ( \*(PUINT32)(&GlobalBuf[8]) == \*(PUINT32)(&GlobalBuf[4]) ) return( USB\_INT\_SUCCESS ); /\* 已到当前文件目录信息扇区 \*/

xWriteCH376Cmd( CMD50\_WRITE\_VAR32 );

xWriteCH376Data( VAR\_FAT\_DIR\_LBA ); /\* 得到前一个扇区,设置为新的文件目录信息扇区LBA地址 \*/

for ( s = 8; s != 12; s ++ ) xWriteCH376Data( GlobalBuf[ s ] );

xEndCH376Cmd( );

++ \*(PUINT32)(&GlobalBuf[0]);

}

}

UINT8 CH376GetLongName( PUINT8 PathName, PUINT8 LongName ) /\* 由短文件名或者目录(文件夹)名获得相应的长文件名 \*/

/\* 需要输入短文件名的完整路径PathName,需要提供缓冲区接收长文件名LongName(以UNICODE小端编码,以双0结束) \*/

/\* 使用了全局缓冲区GlobalBuf的前34个字节,sizeof(GlobalBuf)>=sizeof(FAT\_DIR\_INFO)+2 \*/

{

UINT8 s;

UINT16 NameCount; /\* 长文件名字节计数 \*/

s = CH376FileOpenPath( PathName ); /\* 打开多级目录下的文件或者目录 \*/

if ( s != USB\_INT\_SUCCESS && s != ERR\_OPEN\_DIR ) return( s );

s = CH376DirInfoRead( ); /\* 读取当前文件的目录信息FAT\_DIR\_INFO,将相关数据调到内存中 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

CH376ReadBlock( GlobalBuf ); /\* 从内存缓冲区读取FAT\_DIR\_INFO数据块,返回长度总是sizeof(FAT\_DIR\_INFO) \*/

CH376EndDirInfo( ); /\* 获取完FAT\_DIR\_INFO结构 \*/

GlobalBuf[32] = CH376CheckNameSum( GlobalBuf ); /\* 计算长文件名的短文件名检验和,保存在全局缓冲区中,节约RAM \*/

GlobalBuf[33] = CH376ReadVar8( VAR\_FILE\_DIR\_INDEX ); /\* 当前文件目录信息在扇区内的索引号,保存在全局缓冲区中,节约RAM \*/

NameCount = 0;

while ( 1 ) {

if ( GlobalBuf[33] == 0 ) { /\* 当前的文件目录信息扇区处理结束,转到前一个扇区 \*/

s = CH376LocateInUpDir( PathName ); /\* 在上级目录中移动文件指针到当前文件目录信息所在的扇区 \*/

if ( s != USB\_INT\_SUCCESS ) break;

if ( CH376ReadVar32( VAR\_CURRENT\_OFFSET ) == 0 ) { /\* 当前已经处于目录扇区的开始,无法获取长文件名 \*/

s = ERR\_LONG\_NAME\_ERR;

break;

}

GlobalBuf[33] = DEF\_SECTOR\_SIZE / sizeof( FAT\_DIR\_INFO ); /\* 指向前一个扇区的最后一个文件目录信息 \*/

}

GlobalBuf[33] --; /\* 从后向前搜索文件目录信息 \*/

s = CH376SendCmdDatWaitInt( CMD1H\_DIR\_INFO\_READ, GlobalBuf[33] ); /\* 读取指定的目录信息FAT\_DIR\_INFO,将相关数据调到内存中 \*/

if ( s != USB\_INT\_SUCCESS ) break;

CH376ReadBlock( GlobalBuf ); /\* 从内存缓冲区读取FAT\_DIR\_INFO数据块,返回长度总是sizeof(FAT\_DIR\_INFO) \*/

CH376EndDirInfo( ); /\* 获取完FAT\_DIR\_INFO结构 \*/

if ( ( GlobalBuf[11] & ATTR\_LONG\_NAME\_MASK ) != ATTR\_LONG\_NAME || GlobalBuf[13] != GlobalBuf[32] ) { /\* 类型错误或者校验和错误 \*/

s = ERR\_LONG\_NAME\_ERR;

break; /\* 没有直接返回是因为如果是打开了根目录那么必须要关闭后才能返回 \*/

}

for ( s = 1; s < sizeof( FAT\_DIR\_INFO ); s += 2 ) { /\* 收集长文件名,长文件名的字符在磁盘上UNICODE用小端方式存放 \*/

if ( s == 1 + 5 \* 2 ) s = 14; /\* 从长文件名的第一组1-5个字符跳到第二组6-11个字符 \*/

else if ( s == 14 + 6 \* 2 ) s = 28; /\* 从长文件名的第二组6-11个字符跳到第三组12-13个字符 \*/

LongName[ NameCount++ ] = GlobalBuf[ s ];

LongName[ NameCount++ ] = GlobalBuf[ s + 1 ];

if ( GlobalBuf[ s ] == 0 && GlobalBuf[ s + 1 ] == 0 ) break; /\* 长文件名结束 \*/

if ( NameCount >= LONG\_NAME\_BUF\_LEN ) { /\* 长文件名缓冲区溢出 \*/

s = ERR\_LONG\_BUF\_OVER;

goto CH376GetLongNameE;

}

}

if ( GlobalBuf[0] & 0x40 ) { /\* 长文件名目录信息块结束 \*/

if ( s >= sizeof( FAT\_DIR\_INFO ) ) \*(PUINT16)( &LongName[ NameCount ] ) = 0x0000; /\* 尚未收集到长文件名的结束符,则强制结束 \*/

s = USB\_INT\_SUCCESS; /\* 成功完成长文件名收集完成 \*/

break;

}

}

CH376GetLongNameE:

CH376FileClose( FALSE ); /\* 对于根目录则必须要关闭 \*/

return( s );

}

UINT8 CH376CreateLongName( PUINT8 PathName, PUINT8 LongName ) /\* 新建具有长文件名的文件,关闭文件后返回,LongName输入路径必须在RAM中 \*/

/\* 需要输入短文件名的完整路径PathName(请事先参考FAT规范由长文件名自行产生),需要输入以UNICODE小端编码的以双0结束的长文件名LongName \*/

/\* 使用了全局缓冲区GlobalBuf的前64个字节,sizeof(GlobalBuf)>=sizeof(FAT\_DIR\_INFO)\*2 \*/

{

UINT8 s, i;

UINT8 DirBlockCnt; /\* 长文件名占用文件目录结构的个数 \*/

UINT16 count; /\* 临时变量,用于计数,用于字节读文件方式下实际读取的字节数 \*/

UINT16 NameCount; /\* 长文件名字节计数 \*/

UINT32 NewFileLoc; /\* 当前文件目录信息在上级目录中的起始位置,偏移地址 \*/

for ( count = 0; count < LONG\_NAME\_BUF\_LEN; count += 2 ) if ( \*(PUINT16)(&LongName[count]) == 0 ) break; /\* 到结束位置 \*/

if ( count == 0 || count >= LONG\_NAME\_BUF\_LEN || count > LONE\_NAME\_MAX\_CHAR ) return( ERR\_LONG\_NAME\_ERR ); /\* 长文件名无效 \*/

DirBlockCnt = count / LONG\_NAME\_PER\_DIR; /\* 长文件名占用文件目录结构的个数 \*/

i = count - DirBlockCnt \* LONG\_NAME\_PER\_DIR;

if ( i ) { /\* 有零头 \*/

if ( ++ DirBlockCnt \* LONG\_NAME\_PER\_DIR > LONG\_NAME\_BUF\_LEN ) return( ERR\_LONG\_BUF\_OVER ); /\* 缓冲区溢出 \*/

count += 2; /\* 加上0结束符后的长度 \*/

i += 2;

if ( i < LONG\_NAME\_PER\_DIR ) { /\* 最末的文件目录结构不满 \*/

while ( i++ < LONG\_NAME\_PER\_DIR ) LongName[count++] = 0xFF; /\* 把剩余数据填为0xFF \*/

}

}

s = CH376FileOpenPath( PathName ); /\* 打开多级目录下的文件 \*/

if ( s == USB\_INT\_SUCCESS ) { /\* 短文件名存在则返回错误 \*/

s = ERR\_NAME\_EXIST;

goto CH376CreateLongNameE;

}

if ( s != ERR\_MISS\_FILE ) return( s );

s = CH376FileCreatePath( PathName ); /\* 新建多级目录下的文件 \*/

if ( s != USB\_INT\_SUCCESS ) return( s );

i = CH376ReadVar8( VAR\_FILE\_DIR\_INDEX ); /\* 临时用于保存当前文件目录信息在扇区内的索引号 \*/

s = CH376LocateInUpDir( PathName ); /\* 在上级目录中移动文件指针到当前文件目录信息所在的扇区 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE; /\* 没有直接返回是因为如果是打开了根目录那么必须要关闭后才能返回 \*/

NewFileLoc = CH376ReadVar32( VAR\_CURRENT\_OFFSET ) + i \* sizeof(FAT\_DIR\_INFO); /\* 计算当前文件目录信息在上级目录中的起始位置,偏移地址 \*/

s = CH376ByteLocate( NewFileLoc ); /\* 在上级目录中移动文件指针到当前文件目录信息的位置 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

s = CH376ByteRead( &GlobalBuf[ sizeof(FAT\_DIR\_INFO) ], sizeof(FAT\_DIR\_INFO), NULL ); /\* 以字节为单位读取数据,获得当前文件的目录信息FAT\_DIR\_INFO \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

for ( i = DirBlockCnt; i != 0; -- i ) { /\* 搜索空闲的文件目录结构用于存放长文件名 \*/

s = CH376ByteRead( GlobalBuf, sizeof(FAT\_DIR\_INFO), &count ); /\* 以字节为单位读取数据,获得下一个文件目录信息FAT\_DIR\_INFO \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

if ( count == 0 ) break; /\* 无法读出数据,上级目录结束了 \*/

if ( GlobalBuf[0] && GlobalBuf[0] != 0xE5 ) { /\* 后面有正在使用的文件目录结构,由于长文件名必须连接存放,所以空间不够,必须放弃当前位置并向后转移 \*/

s = CH376ByteLocate( NewFileLoc ); /\* 在上级目录中移动文件指针到当前文件目录信息的位置 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

GlobalBuf[ 0 ] = 0xE5; /\* 文件删除标志 \*/

for ( s = 1; s != sizeof(FAT\_DIR\_INFO); s ++ ) GlobalBuf[ s ] = GlobalBuf[ sizeof(FAT\_DIR\_INFO) + s ];

s = CH376LongNameWrite( GlobalBuf, sizeof(FAT\_DIR\_INFO) ); /\* 写入一个文件目录结构,用于删除之前新建的文件,实际上稍后会将之转移到目录的最末位置 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

do { /\* 向后搜索空闲的文件目录结构 \*/

s = CH376ByteRead( GlobalBuf, sizeof(FAT\_DIR\_INFO), &count ); /\* 以字节为单位读取数据,获得下一个文件目录信息FAT\_DIR\_INFO \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

} while ( count && GlobalBuf[0] ); /\* 如果仍然是正在使用的文件目录结构则继续向后搜索,直到上级目录结束或者有尚未使用过的文件目录结构 \*/

NewFileLoc = CH376ReadVar32( VAR\_CURRENT\_OFFSET ); /\* 用上级目录的当前文件指针作为当前文件目录信息在上级目录中的起始位置 \*/

i = DirBlockCnt + 1; /\* 需要的空闲的文件目录结构的个数,包括短文件名本身一个和长文件名 \*/

if ( count == 0 ) break; /\* 无法读出数据,上级目录结束了 \*/

NewFileLoc -= sizeof(FAT\_DIR\_INFO); /\* 倒回到刚才搜索到的空闲的文件目录结构的起始位置 \*/

}

}

if ( i ) { /\* 空闲的文件目录结构不足以存放长文件名,原因是上级目录结束了,下面增加上级目录的长度 \*/

s = CH376ReadVar8( VAR\_SEC\_PER\_CLUS ); /\* 每簇扇区数 \*/

if ( s == 128 ) { /\* FAT12/FAT16的根目录,容量是固定的,无法增加文件目录结构 \*/

s = ERR\_FDT\_OVER; /\* FAT12/FAT16根目录下的文件数应该少于512个,需要磁盘整理 \*/

goto CH376CreateLongNameE;

}

count = s \* DEF\_SECTOR\_SIZE; /\* 每簇字节数 \*/

if ( count < i \* sizeof(FAT\_DIR\_INFO) ) count <<= 1; /\* 一簇不够则增加一簇,这种情况只会发生于每簇为512字节的情况下 \*/

s = CH376LongNameWrite( NULL, count ); /\* 以字节为单位向当前位置写入全0数据块,清空新增加的文件目录簇 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

}

s = CH376ByteLocate( NewFileLoc ); /\* 在上级目录中移动文件指针到当前文件目录信息的位置 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

GlobalBuf[11] = ATTR\_LONG\_NAME;

GlobalBuf[12] = 0x00;

GlobalBuf[13] = CH376CheckNameSum( &GlobalBuf[ sizeof(FAT\_DIR\_INFO) ] ); /\* 计算长文件名的短文件名检验和 \*/

GlobalBuf[26] = 0x00;

GlobalBuf[27] = 0x00;

for ( s = 0; DirBlockCnt != 0; ) { /\* 长文件名占用的文件目录结构计数 \*/

GlobalBuf[0] = s ? DirBlockCnt : DirBlockCnt | 0x40; /\* 首次要置长文件名入口标志 \*/

DirBlockCnt --;

NameCount = DirBlockCnt \* LONG\_NAME\_PER\_DIR;

for ( s = 1; s < sizeof( FAT\_DIR\_INFO ); s += 2 ) { /\* 输出长文件名,长文件名的字符在磁盘上UNICODE用小端方式存放 \*/

if ( s == 1 + 5 \* 2 ) s = 14; /\* 从长文件名的第一组1-5个字符跳到第二组6-11个字符 \*/

else if ( s == 14 + 6 \* 2 ) s = 28; /\* 从长文件名的第二组6-11个字符跳到第三组12-13个字符 \*/

GlobalBuf[ s ] = LongName[ NameCount++ ];

GlobalBuf[ s + 1 ] = LongName[ NameCount++ ];

}

s = CH376LongNameWrite( GlobalBuf, sizeof(FAT\_DIR\_INFO) ); /\* 以字节为单位写入一个文件目录结构,长文件名 \*/

if ( s != USB\_INT\_SUCCESS ) goto CH376CreateLongNameE;

}

s = CH376LongNameWrite( &GlobalBuf[ sizeof(FAT\_DIR\_INFO) ], sizeof(FAT\_DIR\_INFO) ); /\* 以字节为单位写入一个文件目录结构,这是转移来的之前新建的文件的目录信息 \*/

CH376CreateLongNameE:

CH376FileClose( FALSE ); /\* 对于根目录则必须要关闭 \*/

return( s );

}

#endif