USB-HOST 芯片 CH374 的 U 盘文件级子程序库说明

版本：1C

http://wch.cn

1、概述

很多数码产品以及单片机系统都需要存储器，当前，U 盘（含闪盘、USB 闪存盘、USB 移动硬盘等，下同）已经成为很常用的移动存储设备，其价格仅比相同容量的闪存略高，而远比闪存易于采购和易于携带，并且 U 盘的规格通用，具有多种容量可供选用。所以，数码产品以及单片机系统可以直接采用 U 盘作为大容量的移动存储器。

CH374 是 USB 总线的 HOST 主机及 DEVICE 设备双用接口芯片，单片机可以通过 CH374 读写 U 盘中的数据，由于很多产品最终会与使用 WINDOWS 操作系统的个人计算机交换数据，所以为了方便数据交换，U 盘中的数据应该符合 WINDOWS 的文件系统格式。

CH374 提供了 U 盘文件级子程序库，单片机可以直接调用子程序读写 U 盘中的文件数据，硬件上只需要在原单片机系统中增加一个 CH374 芯片，综合成本较低。CH374 的 U 盘文件级子程序库支持常用的 FAT12、FAT16 和 FAT32 文件系统，支持 U 盘最大容量 100GB。

单片机不需要考虑文件系统，只需要了解文件名、文件长度等基础知识。一个 U 盘中可以有多个文件，每个文件都是一组数据的集合，以文件名区分和识别。文件长度是指文件中有效数据的长度，而实际占用的磁盘空间通常大于或者等于文件长度，实际文件数据的存放可能不是连续的，而是通过一组“指针”链接的多个块（也就是分配单元或簇），从而能够根据需要随时增大文件长度以容纳更多数据。目录是为了便于分类管理，管理者可以人为指定将多个文件归档在一起，例如 2004 年的文件归到一个目录中。

2、子程序库分类

子程序库有两种文件路径表示方式，一种是完整全路径，另一种是逐级路径。目前子程序库主要使用“完整全路径”方式，而普及版源程序的方式是“逐级路径”。

子程序库按功能分为两个版本：

简易版，支持 FAT12、FAT16 和 FAT32 文件系统，只支持读文件，不支持新建和写文件。

增强版，支持 FAT12、FAT16 和 FAT32 文件系统，支持文件读写、删除和新建等。

单片机的资源和速度有限，通常在处理 FAT16 文件系统的 U 盘时效率最高，而在处理 FAT32 时效率最低（约低 5%到 20%）。使用 U 盘文件级子程序库实现同样的功能，简易版效率最高。

各子程序库的子程序调用方式完全统一，同一种单片机的示例程序完全通用，只需要在链接时指定不同的子程序库就可以实现不同的功能。不同单片机的 C 语言示例程序基本通用，尤其是 main 主程序基本上适用于所有单片机，只需要修改硬件相关部分，重新编译和链接就可以使用。

3、当前版本

使用 U 盘文件级子程序库，单片机系统需要具备以下硬件资源：不少于 4KB 到 7KB 的程序空间，不少于 620 字节的随机存储器 RAM。对于 MCS-51 单片机，620 字节的 RAM 包括不少于 70 字节的内部 RAM 和 550 字节的外部 RAM，不同的子程序库对内部 RAM 的占用稍有不同。

目前的 U 盘文件级子程序库只支持第一个逻辑盘，支持 8+3 格式的大写字母的短文件名，可以支持中文文件名，通过外加程序可以支持小写字母及长文件名。

目前可以提供以下 U 盘文件级子程序库（具体适用环境请参考各.H 头文件中的说明）：

MCS51 单片机的增强版子程序库，文件名是 CH374HF6.LIB，网站上可以下载该子程序库。子目录是 LIB6，提供多个示例程序 CH374HFT.C。根据 CH374 的 INT#连接方式分为查询方式和中断方式，根据文件数据块读写方式分为扇区模式和字节模式，根据外部 RAM 的数据复制方式分为“单 DPTR 复制”、“双 DPTR 复制”以及“单 DPTR 和 P2+R0 复制”，分别对应 I/O 接口子程序库 C51DPTR1.LIB 和 C51DPTR2.LIB 以及 C51P2R0.LIB。

MCS51 单片机的简易版子程序库，文件名是 CH374HF3.LIB。子目录是 LIB3，可参考 CH374HF6

的例子程序。与 CH374HF6 相同，链接时需选择 I/O 接口子程序库。

MCS51 单片机的增强版子程序库，绝大多数变量使用外部 RAM，基本不用内部 RAM，文件名是 CH374HFA.LIB。子目录是 LIBA，可以参考 CH374HF6 的例子程序。与 CH374HF6 相同，链接时需选择 I/O 接口子程序库。

MCS51 单片机非总线 I/O 的增强版子程序库，绝大多数变量使用外部 RAM，基本不用内部 RAM，文件名是 CH374HFC.LIB。子目录是 LIBC，适用于非标准 MCS51 单片机、或者单片机通过普通 I/O 引脚模拟并口连接 CH374、或者单片机通过 SPI 串口连接 CH374 等应用。

MCS51 单片机非总线 I/O 的简易版子程序库，绝大多数变量使用外部 RAM，基本不用内部 RAM，文件名是 CH374HF4.LIB。子目录是 LIB4，适用于非标准 MCS51 单片机、或者单片机通过普通 I/O 引脚模拟并口连接 CH374、或者单片机通过 SPI 串口连接 CH374 等应用。

MSP430 单片机的增强版子程序库，文件名是 CH374HFF.R43，支持 V3xx 的 IAR 编译环境。子目录是 LIBF，提供示例程序。

32 位 ARM 单片机的增强版子程序库，文件名是 CH374HF9.LIB，支持 ADS 编译环境，ARM 指令、小端数据格式。子目录是 LIB9，提供示例程序。

32 位 ARM 单片机的增强版子程序库，文件名是 CH374HFS.LIB 和 CH374HFI.LIB，均为 ADS 编译环境，前者为小端数据格式，后者为大端数据格式，均支持 ARM/Thumb 相互调用。子目录是 LIBS 和 LIBI。

32 位 ARM 单片机的增强版子程序库，文件名是 CH374HFM.LIB 和 CH374HFU.R79，前者为 Keil 编译环境，后者为 IAR 编译环境，均为小端数据格式、支持 ARM/Thumb 相互调用。子目录是 LIBM 和 LIBU。

AVR 单片机的增强版子程序库，文件名是 CH374HFB.A 和 libCH374HFD.A 以及 CH374HFJ.R90，分别支持 ICC、WinAVR-GCC、IAR 编译环境。子目录是 LIBB 和 LIBD 以及 LIBJ，提供示例程序。

实际的子程序库更多，如 MIPS 内核单片机、Renesas/M16C 系列单片机、MC9S12 系列单片机、MC68 系列单片机、MB95 系列单片机、ST20 系列单片机等，有关单片机型号和编译器版本等信息请查看其中的.H 头文件的说明。

目前，除了 MCS51 单片同的 CH374HF3、CH374HF6、CH374HFA 三个子程序库只支持并口连接之外，其它子程序库都支持并口连接或者 SPI 串口连接。在例子程序中，可以支持硬件总线并口和通用 I/O 模拟并口，可以支持硬件 SPI 串口和通用 I/O 模拟 SPI 串口等。

批量用户可以预定其它单片机或者 DSP 的 U 盘文件级子程序库。

4、一般说明

使用 U 盘文件级子程序库，单片机系统需要具有不少于 620 字节的随机存储器 RAM，其中 512 字节用于磁盘数据缓冲区，除此之外，根据操作方式的不同，可能还需要文件数据缓冲区，通常情况下，RAM 越多读写效率越高。（注：个别大扇区 U 盘可能需要更大的 RAM 空间）

为了使用子程序库，应该在源程序中包含子程序库的头文件 CH374HF?.H，该头文件可以为 CH374 子程序库分配必要的 I/O 及内存资源，并产生必要的与硬件有关的目标代码，如果该文件是被工程项目的多个源程序包含作为头文件，那么应该只允许一个头文件分配资源和产生代码，除此之外的头文件应该被事先定义 CH374HF_NO_CODE 常量，从而禁止该头文件产生重复的目标代码。例如：

```
#define     CH374HF_NO_CODE     1    // 禁止分配资源或产生代码
#include        CH374HF?.H
```

4.1. 存取模式

子程序库对 U 盘文件的读写方式分为两种：扇区模式和字节模式。

扇区模式下，以扇区（每扇区通常是 512 字节，实际值在 CH374vSectorSize 变量中）为基本单位对 U 盘文件进行读写，所以读写速度较快，但是通常情况下需要额外的文件数据缓冲区（如果与磁盘数据缓冲区合用则效率不高），额外的文件数据缓冲区必须是扇区长度 CH374vSectorSize 的整数倍，所以适用于 RAM 多、数据量大、频繁读写数据的单片机系统。

字节模式下，以字节为基本单位对 U 盘文件进行读写，读写速度较慢，但是不需要额外的文件数据缓冲区（实际上是与磁盘数据缓冲区合用），使用方便，适用于 RAM 少、数据量小或者数据零碎、不经常读写数据的单片机系统。如果频繁地向 U 盘写入零碎的数据，可能会缩短 U 盘中闪存的使用寿命（因为闪存只能进行有限次擦写）。

查看子程序库的全局变量“磁盘及文件状态 CH374DiskStatus”可以获取当前的文件模式：为 DISK_OPEN_FILE 则代表扇区模式，为 DISK_OPEN_FILE_B 则代表字节模式。

每次新建或者打开一个文件后，默认为扇区模式，支持以扇区为单位的文件操作子程序 CH374FileRead 和 CH374FileWrite 及 CH374FileLocate。当执行一次以字节为单位的操作命令后将自动进入字节模式（只有关闭文件后再重新打开才能恢复扇区模式），支持以字节为单位的操作子程序 CH374ByteRead 和 CH374ByteWrite 及 CH374ByteLocate。对于已打开的同一个文件，不能混用两种模式的操作子程序。

4.2. 文件长度

在 FAT 文件系统中，磁盘容量以簇为基本单位进行分配，而簇的大小总是扇区的倍数，所以文件的占用空间总是簇的倍数，也是扇区的倍数。虽然文件占用的空间是簇或者扇区 CH374vSectorSize 的倍数（CH374vSectorSize 通常是 512），但是在实际应用中，保存在文件中的有效数据的长度却不一定是 CH374vSectorSize 的倍数，所以 FAT 文件系统在 FDT 文件目录表中专门记录了当前文件中有效数据的长度，也就是通常所说的文件长度，文件长度总是小于或者等于文件占用的空间。

在对文件写入数据后，如果是覆盖了原数据，那么文件长度可能不发生变化，当超过原文件长度后，变为追加数据，那么文件长度应该发生变化（增大）。如果向文件追加数据后，没有修改 FDT 中的文件长度，那么 FAT 文件系统会认为超过文件长度的数据是无效的，正常情况下，计算机无法读出超过文件长度的数据，虽然数据实际存在。

写数据子程序 CH374FileWrite 和 CH374ByteWrite 用于将数据写到文件所占用的磁盘空间中，如果是追加数据并且超过原文件长度后，子程序会自动更新全局变量“文件长度 CH374vFileSize”。

如果数据量少或者数据不连续，那么可以在每次追加数据后立即更新 FDT 中的文件长度，但是，如果数据量大并且需要连续写入数据，立即更新 FDT 会降低效率，并且频繁修改 FDT 也会缩短 U 盘中闪存的使用寿命（因为闪存只能进行有限次擦写），所以在这种情况下，应该在连续写入多组数据后再更新一次 FDT 中的文件长度，或者一直等到关闭文件时再更新文件长度，CH374FileClose 总是用“全局变量 CH374vFileSize”更新 FDT 中的文件长度。

在字节模式下，子程序库能够自动处理任意的文件长度，在 CH374FileClose 关闭文件时，总是用“文件长度 CH374vFileSize”自动更新 FDT 中的文件长度，确保文件长度真实反应文件中有效数据的长度。在字节模式下修改文件长度有 3 种方法：

方法 1：调用 CH374ByteWrite，写入 0 字节数据，强制更新文件长度

方法 2：调用 CH374FileClose，关闭文件，总是自动更新文件长度

方法 3：调用 CH374FileModify，指定修改文件长度

在扇区模式下，子程序库只能自动处理扇区大小 CH374vSectorSize 的倍数的文件长度，在关闭文件时，可以选择自动更新文件长度或者不更新，如果文件中有效数据的长度是 CH374vSectorSize 的倍数，那么可以指定自动更新，如果有效数据的长度不是 CH374vSectorSize 的倍数，那么可以由 CH374FileModify 修改文件长度。在扇区模式下修改文件长度有 3 种方法：

方法 1：调用 CH374FileWrite，写入 0 扇区数据，强制更新文件长度，如果希望文件长度不是 CH374vSectorSize 的倍数，那么应该事先人为修改 CH374vFileSize

方法 2：调用 CH374FileClose，关闭文件，指定自动更新文件长度，如果希望文件长度不是 CH374vSectorSize 的倍数，那么应该事先人为修改 CH374vFileSize

方法 3：调用 CH374FileModify，指定修改文件长度

虽然子程序库最大支持 1GB 的文件，但是为了提高效率，建议单个文件的长度不要超过 100MB，通常在几 KB 到几 MB 范围是比较正常的，而 U 盘总容量限制为 2048GB。

5、操作步骤

5.1. 子程序库提供的子程序总表

子程序分类	子程序简称	用途和概述（详细说明见后）
常用的基本子程序	CH374Init	初始化 CH374 芯片,可以用 CH374LibInit 代替
	CH374DiskConnect	查询 U 盘是否连接，支持 USB-HUB
	CH374DiskReady	查询 U 盘是否准备就绪,通常在就绪后才能读写
	CH374FileOpen	打开指定名称的文件或者目录、搜索枚举文件
	CH374FileEnumer	搜索枚举指定目录下的文件,返回文件名
	CH374FileCreate	新建文件并打开,如果文件已存在则先删除再新建
	CH374FileClose	关闭当前文件
扇区模式读写文件	CH374FileLocate	以扇区为单位移动当前文件指针
	CH374FileReadX	以扇区为单位从当前文件读取数据
	CH374FileWriteX	以扇区为单位向当前文件写入数据
字节模式读写文件	CH374ByteLocate	以字节为单位移动当前文件指针，进入字节模式
	CH374ByteRead	以字节为单位从当前文件读取数据，进入字节模式
	CH374ByteWrite	以字节为单位向当前文件写入数据，进入字节模式
查询修改文件信息	CH374FileQuery	查询当前文件的信息：长度,日期,时间等
	CH374FileModify	查询或修改当前文件的信息：长度,日期,时间等
偶尔用到	CH374DirtyBuffer	清除磁盘缓冲区，如果临时用过 DISK_BASE_BUF
	CH374DiskSize	查询磁盘总容量：U 盘/闪盘/移动硬盘的物理容量
	CH374DiskQuery	查询磁盘信息：U 盘总容量,U 盘剩余容量等
	CH374FileErase	删除文件并关闭
备用	CH374GetVer	获取当前子程序库的版本号
	CH374Reset	复位 CH374 芯片
	CH374SaveVariable	备份/保存/恢复库的变量,用于切换多个 CH374 芯片
	CH374HostTransact	执行 USB 基本传输事务（最底层 USB 传输）
	CH374CtrlTransfer	执行 USB 控制传输（不支持控制读数据）
	CH374BulkOnlyCmd	执行基于 BulkOnly 协议的命令
	CH374DelaymS	延时指定毫秒，有效值是 1 毫秒到 255 毫秒
	CH374EmbHubAttach	检查当前的内置 HUB 端口是否有 USB 设备（尚未声明）

调用各个子程序之前需要输入的参数和子程序返回后的结果，都在全局结构变量 mCmdParam 中，调用前需参考 CMD_PARAM 结构准备参数，返回后需参考 CMD_PARAM 结构获取结果。例如，CH374FileReadX 子程序的输入参数和输出结果在 mCmdParam.ReadX 中。

注：上表中 CH374FileEnumer、CH374FileQuery、CH374Reset 等子程序实际由.H 头文件间接提供。

5.2. 常用步骤简述，可以根据实际情况进行调整

5.2.1. 初始化（进行任何一种文件操作之前的必要步骤，参考 EXAM1 和 EXAM13）

⑴ CH374Init，进入 USB-HOST 工作方式

⑵ 等待 U 盘连接（中断方式依靠 CH374 事件通知，查询方式依靠 CH374DiskConnect 主动查询）

⑶ CH374DiskReady，必要步骤（理论上可选的），如果多次失败那么应该判断是否为 USB-HUB

⑷ CH374DiskSize，可选步骤，通常不需要

5.2.2 顺序读文件

⑴ CH374FileOpen，打开文件

⑵ 多次 CH374FileRead 或 CH374ByteRead，读取数据

⑶ CH374FileClose，关闭文件

5.2.3 读文件的指定位置
(1) CH374FileOpen，打开文件
(2) CH374FileLocate 或 CH374ByteLocate，移动文件指针到指定位置
(3) 多次 CH374FileRead 或 CH374ByteRead，读取数据，期间还可以移动文件指针
(4) CH374FileClose，关闭文件

5.2.4 顺序改写文件（覆盖原数据，超过原文件长度后转变为追加数据）
(1) CH374FileOpen，打开文件
(2) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据
(3) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(4) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.5 改写文件的指定位置（覆盖原数据，超过原文件长度后转变为追加数据）
(1) CH374FileOpen，打开文件
(2) CH374FileLocate 或 CH374ByteLocate，移动文件指针到指定位置
(3) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据，期间还可以移动文件指针
(4) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(5) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.6 向已有文件追加数据
(1) CH374FileOpen，打开文件
(2) CH374FileLocate 或 CH374ByteLocate，移动文件指针到末尾，0xFFFFFFFF
(3) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据
(4) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(5) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.7 新建文件
(1) CH374FileCreate，新建文件
(2) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据
(3) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(4) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.8 先读文件再改写文件
(1) CH374FileOpen，打开文件
(2) 多次 CH374FileRead 或 CH374ByteRead，读取数据
(3) CH374FileLocate 或 CH374ByteLocate，移动文件指针到头部，00000000
(4) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据，覆盖原数据
(5) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(6) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.9 先写文件再读文件复查
(1) CH374FileOpen，打开文件
(2) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据
(3) CH374FileLocate 或 CH374ByteLocate，移动文件指针到头部，00000000
(4) 多次 CH374FileRead 或 CH374ByteRead，读取数据，复查
(5) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(6) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.10 如果文件已经存在则追加数据，如果文件不存在则新建文件再写入数据

(1) CH374FileOpen，打开文件，如果返回 ERR_MISS_FILE 文件不存在，则转到步骤(4)
(2) CH374FileLocate 或 CH374ByteLocate，移动文件指针到末尾，0xFFFFFFFF
(3) 转到步骤(5)，准备追加数据
(4) CH374FileCreate，新建文件，准备写入数据
(5) 多次 CH374FileWrite 或 CH374ByteWrite，写入数据
(6) 如果是追加数据导致文件长度增大，那么需要参考修改文件长度的几种方法
(7) CH374FileClose，关闭文件，如果是字节模式，将自动更新文件长度

5.2.11 定期采集数据（适用于数据量较小的情况）
(1) 采集之前，CH374FileCreate，新建文件
(2) 采集数据，转换为相应的格式，例如二进制数据、字符串等
(3) CH374ByteWrite，写入数据，一次写不完，可以分多次写入
(4) 如果要等很长时间才有下一组数据，为了避免在此期间发生断电、U 盘拔出等事件，导致文件长度不正确，可以用 CH374ByteWrite 写入空数据，强制更新文件长度
(5) 如果整个采集过程结束，或者文件已经太大，那么转到步骤(6)，否则转到步骤(2)
(6) CH374FileClose，关闭文件，自动更新文件长度
(7) 如果是因为文件已经太大的原因，那么转到步骤(1)，新建另一个文件继续

5.2.12 搜索和枚举文件名
请参考有关 EXAM13 例子的说明

6、变量和子程序

6.1 子程序库提供的全局变量

CH374IntStatus; /* CH374 操作的中断状态 */
当检测到 USB 设备插拔事件以及执行完 USB 传输事务后，CH374 将以中断方式通知单片机，INT#引脚输出低电平，直到单片机清除中断标志才恢复高电平。单片机也可以用查询方式查询 CH374 的 INT#引脚或者中断标志寄存器，以判断 CH374 是否操作完成。该单元由子程序库中的中断标志分析程序刷新，用于产生与 CH375 芯片相兼容的中断状态。

CH374DiskStatus; /* 磁盘及文件状态 */
CH374 的子程序以及中断服务程序或者查询程序共同维护一个全局变量，用于表示当前 CH374 以及磁盘的工作状态。子程序根据这个状态决定是否执行某操作，或者决定是否重新初始化。只有 CH374 的初始化程序和各个 U 盘文件级子程序，以及中断服务程序或者查询程序可以修改该单元的数值，其它程序不应该修改但可以查询，以便了解当前 CH374 或 U 盘的工作状态。

下面的全局变量用于特定用途，通常情况下，应用程序不会用到，并且绝对不应该修改。
CH374vDiskFat; /* 当前逻辑盘的 FAT 标志：1=FAT12，2=FAT16，3=FAT32 */
CH374vSecPerClus; /* 当前逻辑盘的每簇扇区数 */
CH374vStartCluster; /* 当前文件或者目录的起始簇号 */
CH374vFileSize; /* 当前文件的长度，必要时可以人为修改 */
CH374vCurrentOffset; /* 当前文件指针，当前读写位置的字节偏移 */
CH374vDataStart; /* 逻辑盘的数据区域的起始 LBA */
CH374vFdtLba; /* 当前 FDT 所在的 LBA 地址 */
CH374vFdtOffset; /* 当前 FDT 在扇区内的偏移地址 */
CH374vDiskRoot; /* 对于 FAT16 为根目录占用扇区数,对于 FAT32 为根目录起始簇号 */
CH374vRetryCount; /* 位 7 为 1 则 NAK 无限重试,为 0 则 NAK 不重试,位 5 为 1 则 USB 存储设

备的子类为 6,为 0 则子类为非 6,位 3 至位 0 为出错重试次数 */
CH374vUsbPidIn; /* 当前 USB 传输的 PID 是否为 IN:1=是 IN,0=是 OUT 或者 SETUP */
CH374vDevEndpTog; /* USB 存储设备的端点的数据同步标志:位 7 对应 BIT_HOST_RECV_TOG,位 6 对应 BIT_HOST_TRAN_TOG,位 3 必须为 1,其它位必须为 0 */
CH374vCurrentLun; /* USB 存储设备的当前逻辑单元号 */
CH374vDiskRetry; /* U 盘读写失败后的重试计数,位 7 为 1 则启用磁盘存取的外部接口 */
CH374vEmbHubIndex; /* CH374 内置 HUB 的当前端口号,为 0 则不启用内部 HUB,否则为端口号 */
CH374vHubPortCount; /* HUB 上的端口数,为 0 则没有 HUB */
CH374vHubPortIndex; /* HUB 上的当前操作端口号,位 7 为 0 则自动查询,为 1 则指定端口号 */
CH374vSectorSize; /* 当前 U 盘的扇区大小,通常是 512 字节,个别大扇区 U 盘会大于 512 */
该变量在调用 CH374DiskReady 后为当前 U 盘的实际扇区大小,可能值是 512、1024、2048、4096
pDISK_BASE_BUF; /* 指向 RAM 磁盘数据缓冲区,缓冲区长度不小于 CH374vSectorSize */
该变量指向外部 RAM 中的磁盘数据缓冲区,由应用程序在调用 CH374Init 之前初始化

6.2 子程序库提供的子程序

```
CH374GetVer( );          /* 获取当前子程序库的版本号 */

CH374Reset( );           /* 复位 CH374 */

CH374Init( );            /* 初始化 CH374，在开机后或者复位 CH374 之后，应该初始化 */

CH374DiskConnect( );     /* 检查磁盘是否连接 */
```

检查 USB 磁盘是否连接。示例：

```
i=CH374DiskConnect( ); /* 查询 U 盘是否连接，返回 ERR_SUCCESS 则说明当前已连接 */
if ( i==ERR_SUCCESS ) /* 已经连接 */
else if ( i==ERR_DISK_DISCON ) /* 已经断开 */
else if ( i==ERR_HUB_PORT_FREE ) /* USB-HUB 已经连接但是 HUB 端口尚未连接磁盘 */
```

```
CH374DiskReady( );       /* 查询磁盘是否准备好 */
```

检查 USB 磁盘是否准备好，大多数 U 盘不需要这一步，但是某些 U 盘必须要执行这一步才能允许数据读写，所以强烈建议在 U 盘连接后先执行该命令，再进行文件读写。示例：

```
i=CH374DiskReady( ); /* 查询 U 盘是否准备好，有些 U 盘必须要调用一次该子程序 */
if ( i!=ERR_SUCCESS ) /* 还未准备好 */ else /* 准备好了，可以读写数据 */
```

```
CH374FileOpen( );        /* 打开文件或者目录，或者枚举文件 */
```

① 打开文件

调用前，应该在 mCmdParam.Open.mPathName 中提供文件名，包括完整的路径名，例如

```
C:\WINDOWS/SYSTEM/MYLIB.DLL
\TODAY1.TXT
\YEAR2004\MONTH05\DATE18.DAT
```

路径名和文件名的格式与 DOS 文件名格式相同，但是盘符和冒号可以省略，左斜杠与右斜杠等效，所有字符必须是大写，不能使用通配符，文件名长度不超过 11 个字符，其中主文件名不超过 8 个字符，扩展名不超过 3 个字符，如果有扩展名，则用小数点与主文件名隔开。由于单片机 RAM 有限，路径名有长度限制，最大长度是 MAX_PATH_LEN-1 个字符，所以子目录的深度不宜超过 5 级，如果是为 RAM 较大的单片机定制子程序库，则可以没有长度限制。示例：

```
strcpy( mCmdParam.Open.mPathName, "\\YEAR2004\\CH374HFT.C" );
CH374FileOpen( );  /* 打开文件 */
```

如果路径名太长，那么可以分多次逐级打开，首先打开子目录，直到最后再打开文件，其中，

首次打开必须是从根目录开始，所以路径名首字符必须是斜杠，以后接着前级再打开时的首字符必须不是斜杠。示例：打开文件"/YEAR2004/MONTH05/DATE18/HOUR08/ADC.TXT"

```
strcpy( mCmdParam.Open.mPathName, "/YEAR2004/MONTH05/DATE18" ); /* 目录名 */
i=CH374FileOpen( ); /* 因为路径名太长，所以分两次打开，先打开前 3 级子目录 */
if ( i==ERR_SUCCESS ) {  /* 前 3 级子目录成功打开，下面接着打开下级目录及文件 */
  strcpy( mCmdParam.Open.mPathName, "HOUR08/ADC.TXT" ); /* 首字符不是斜杠 */
  i=CH374FileOpen( ); /* 打开第 4 级子目录和文件 */
}
if ( i!=ERR_SUCCESS ) /* 出错 */ else /* 成功打开文件 */
```

如果文件名参数为"/"则可以打开根目录，从而便于自行处理长文件名，用完后必须关闭。如果打开的是子目录，那么文件长度 CH374vFileSize 总是 0xFFFFFFFF，否则为真正的文件长度。

② 枚举文件（查询文件），与 CH374FileEnumer 相同，但是返回值不同（ERR_FOUND_NAME）

```
CH374FileEnumer( );      /* 枚举文件 */
```

如果需要查询文件，可以通过该子程序进行枚举，方法是，以通配符*代替需要查询的文件名中的全部或者部分字符，通配符*后面不能再有字符，而必须跟有枚举序号，例如

C:\WINDOWS\SYSTEM*

枚举 C:\WINDOWS\SYSTEM\目录下的所有文件，例如 ABC.EXE 或者 NEW.TXT 等

\TODAY1.*

枚举根目录下以 TODAY1 为主文件名的所有文件，例如 TODAY1.C 或者 TODAY1

\YEAR2004\MONTH05\DATE1*

枚举\YEAR2004\MONTH05\目录下以 DATE1 开头的所有文件，例如 DATE12.TXT 或者 DATA1

由于符合条件的文件名通常会很多，所以需要指定序号，序号指定在通配符*之后，例如

\TODAY1.*\x0　　（这是 C 语言表达方式，“\x0”代表一字节数据 00H）

枚举根目录下以 TODAY1 为主文件名的文件，返回搜索到的第 1 个匹配文件名

\TODAY1.*\x5　　（这是 C 语言表达方式，“\x5”代表一字节数据 05H）

枚举根目录下以 TODAY1 为主文件名的文件，返回搜索到的第 6 个匹配文件名

\YEAR2004\DATE1*\x10　　（这是 C 语言表达方式，“\x10”代表一字节数据 10H）

枚举\YEAR2004\目录下以 DATE1 为开头的文件，返回搜索到的第 17 个匹配文件名

\YEAR2004\DATE1*\xFF　　（这是 C 语言表达方式，“\xFF”代表一字节数据 255）

枚举\YEAR2004\目录下以 DATE1 为开头的文件，返回搜索到的由 CH374vFileSize 指定序号的匹配文件名，即当通配符*后的序号为 255 时，将由变量 CH374vFileSize 指定序号

如果调用 CH374FileEnumer 时指定带有通配符*的文件名，那么该子程序将枚举文件名，并且计数到通配符*后的序号再返回搜索到的匹配文件名。枚举文件并不会打开文件，如果需要对搜索到的文件进行读写，可以再调用 CH374FileOpen 打开返回的文件名。在评估板资料 LIB3 子目录下的 CH374HFT.C 中有枚举程序示例，可以显示根目录或者子目录下的所有文件。示例：

```
for ( c=0; c<254; c++ ) {  /* 最多搜索前 254 个文件 */
  strcpy( mCmdParam.Enumer.mPathName, "\\C51\\CH374*" );
  /* 在 C51 子目录下搜索以 CH374 开头的文件名，*为通配符 */
  i = strlen( mCmdParam.Enumer.mPathName );  /* 计算文件名长度，指向结束符 */
  mCmdParam.Enumer.mPathName[i] = c;  /* 将结束符替换为搜索的序号，从 0 到 254 */
  i=CH374FileEnumer( );  /* 文件名中含有通配符*，枚举/搜索文件而不打开 */
  if ( i==ERR_MISS_FILE ) break;  /* 再也搜索不到匹配文件，已经没有匹配的文件名 */
  if ( i!=ERR_SUCCESS ) break;  /* 出错 */
  printf( "found name %d#: %s\n", (unsigned int)c, mCmdParam.Enumer.mPathName );
  /* 搜索到相匹配的文件名，显示序号和搜索到的匹配文件名或者子目录名 */
}
```

枚举更多数量的文件的例子：

```
for ( int count=0; count<20000; count++ ) {  /* 最多搜索前 20000 个文件 */
  strcpy( mCmdParam.Enumer.mPathName, "\\*" );  /* 在根目录下搜索所有文件名 */
```

```
    i = strlen( mCmdParam.Enumer.mPathName );  /* 计算文件名长度，指向结束符 */
    mCmdParam.Enumer.mPathName[i] = 0xFF;  /* 将结束符替换为 255 说明序号在变量中 */
    CH374vFileSize = count;  /* 指定搜索的序号，几乎没有上限 */
    i=CH374FileEnumer( );  /* 文件名中含有通配符*，枚举/搜索文件而不打开 */
    if ( i==ERR_MISS_FILE ) break;  /* 再也搜索不到匹配文件，已经没有匹配的文件名 */
    if ( i!=ERR_SUCCESS ) break;  /* 出错 */
    printf( "found name %d#: %s\n", count, mCmdParam.Enumer.mPathName );
    /* 搜索到相匹配的文件名，显示序号和搜索到的匹配文件名或者子目录名 */
    if ( CH374vFileSize!=0xFFFFFFFF ) printf( "this is a file\n" );  /* 枚举到文件 */
    else printf( "this is a directory\n" );  /* 枚举到子目录 */
  }
```

如果要快速地连续枚举多个文件，那么还可以通过文件名枚举回调子程序 xFileNameEnumer 实现。具体做法是，指定文件名枚举序号 CH374vFileSize 为 0xFFFFFFFF 后调用 CH374FileOpen（或者 CH374FileEnumer），那么每搜索到一个文件 CH374FileOpen 都会调用 xFileNameEnumer 回调子程序。在回调子程序中由 CH374vFdtOffset 得到结构 FAT_DIR_INFO，分析结构中的 DIR_Attr 以及 DIR_Name 判断是否为所需文件名或者目录名，并记录相关信息后返回，CH374FileOpen 会依次枚举全部文件直到找不到匹配的文件名后才返回应用程序，即一次调用多次枚举。

CH374FileErase( ); /* 删除文件并关闭 */

删除当前已打开的文件或者指定文件名的文件。如果当前有文件已经打开或者尚未关闭，该子程序直接删除该文件并关闭，如果当前没有文件被打开，那么应该在 mCmdParam.Erase.mPathName 中指定被删除文件的路径名和文件名，格式与 CH374FileOpen 相同，不支持通配符。

CH374FileCreate( ); /* 新建文件并打开，如果文件已经存在则先删除后再新建 */

新建文件，调用该子程序前，应该在 mCmdParam.Create.mPathName 中指定新文件的路径名和文件名，格式与 CH374FileOpen 相同，不支持通配符。如果存在同名文件，那么该同名文件将首先被删除，然后再新建文件。如果不希望已有文件被删除，那么应该事先调用 CH374FileOpen 确认文件不存在后再新建。新建文件的属性默认为存档 ATTR_ARCHIVE，文件日期和时间默认为 2004 年 1 月 1 日 0 时 0 分 0 秒，文件默认长度为 1，如果需要修改则可以调用 CH374FileModify。示例：

```
strcpy( mCmdParam.Create.mPathName, "/C51/NEWFILE.TXT" );
/* 新文件名，在 C51 子目录下新建文件 NEWFILE.TXT，要求 C51 已经事先存在 */
CH374FileCreate( );  /* 新建文件并打开,如果文件已经存在则先删除后再新建 */
```

CH374FileClose( ); /* 关闭当前文件 */

打开文件使用完毕后，应该关闭文件。对于读操作，关闭文件是可选操作。对于写操作，关闭文件的同时，可以让子程序自动更新文件长度。在扇区模式下，自动更新的文件长度是以扇区为单位计算的，所以文件长度通常是扇区大小 CH374vSectorSize（通常是 512，个别 U 盘更大）的倍数，mCmdParam.Close.mUpdateLen 为 1 时自动更新文件长度（如果已经对该文件执行写操作添加了数据），为 0 时不要自动更新文件长度。在字节模式下，自动更新的文件长度是以字节为单位，所以可以获得适当的长度。在扇区模式下，如果希望文件长度不是 CH374vSectorSize 的倍数，那么单片机可以在关闭文件前调用 CH374FileModify 修改文件为指定的长度，并且在关闭文件时指定不要自动更新文件长度。对于以字节为单位的文件读写，关闭文件时能够自动更新为适当的文件长度，所以不需要调用 CH374FileModify 修改文件长度。

CH374FileQuery( ); /* 查询当前文件的信息 */

查询当前已打开文件的属性、日期、时间、长度等信息。返回时

mCmdParam.Modify.mFileSize 中是文件的长度，以字节为单位，长度可以是 0

mCmdParam.Modify.mFileDate 中是文件修改时间，格式参考 CH374HF?.H 中的说明

mCmdParam.Modify.mFileTime 中是文件修改时间，格式参考 CH374HF?.H 中的说明

mCmdParam.Modify.mFileAttr 中是文件属性，例如数值 01H 说明该文件是只读文件

CH374FileModify( ); /* 查询或者修改当前文件的信息 */

与 CH374FileQuery 对应，该子程序用于修改当前已打开文件的属性、日期、时间、长度等。输入参数中，如果参数的新值指定为 0xFFFFFFFF，那么该参数将保持原值，否则将被修改为新值。

mCmdParam.Modify.mFileSize 指定新的文件长度，为 0FFFFFFFFH 则不修改，返回原长度

mCmdParam.Modify.mFileDate 指定新的文件日期，为 0FFFFH 则不修改，返回原日期

mCmdParam.Modify.mFileTime 指定新的文件时间，为 0FFFFH 则不修改，返回原时间

mCmdParam.Modify.mFileAttr 指定新的文件属性，为 0FFH 则不修改，返回原属性

例如在 mCmdParam.Modify.mFileSize 中指定 1105，而在其它单元中指定 0FFFFH，则将已打开文件的长度修改为 1105，但是不修改文件属性、时间和日期。示例：

```
mCmdParam.Modify.mFileAttr = ATTR_READ_ONLY;  /* 指定新的文件属性为只读 */
mCmdParam.Modify.mFileTime = 0xffff;  /* 不修改原文件时间 */
mCmdParam.Modify.mFileDate = MAKE_FILE_DATE( 2006, 3, 28 );  /* CH374HF?.H 宏定义 */
/* 指定新的文件日期是 2006.03.28 */
mCmdParam.Modify.mFileSize = 1105;  /* 指定新的文件长度是 1105 字节 */
CH374FileModify( );  /* 修改当前文件的信息，修改属性、日期和长度 */
```

CH374FileLocate( ); /* 以扇区为单位移动当前文件指针 */

移动当前已打开文件的指针，用于从指定位置读取数据，或者向指定位置写入数据。例如，单片机希望跳过文件的前 CH374vSectorSize*2（通常 CH374vSectorSize 为 512）个字节再读取数据，那么可以在 mCmdParam.Locate.mSectorOffset 参数中输入 2，调用该子程序将文件指针移动到 2 个扇区后，也就是从 CH374vSectorSize*2 个字节处开始。对于写操作，如果单片机打算在原文件的尾部继续添加数据，而不希望影响前面的原有数据，那么可以指定一个很大的扇区偏移，例如在 mCmdParam.Locate.mSectorOffset 参数中输入 0FFFFFFFFH，将文件指针移动原文件的末尾，以便追加数据。该子程序将文件长度 CH374vFileSize 取整转换为扇区数后作为最大文件指针，如果文件长度不是扇区大小 CH374vSectorSize 的倍数，那么文件尾部不足一个扇区的零碎数据部分将被忽略。该子程序返回时，mCmdParam.Locate.mSectorOffset 为当前文件指针所指向的数据在磁盘中的 LBA 扇区号，如果是 0FFFFFFFFH 则说明已到文件尾。

CH374FileReadX( ); /* 以扇区为单位从当前文件读取数据到指定缓冲区 */

从当前已打开文件中读取数据，每次读取后自动移动文件指针，第二次调用时将从第一次读取数据的后面继续读取数据。在调用该子程序前，应该在 mCmdParam.ReadX.mDataBuffer 中指定缓冲区起始地址，子程序返回后更新为当前缓冲区地址（也就是起始地址加上已经读取的长度），并在 mCmdParam.ReadX.mSectorCount 中指定需要读取的扇区数，所以读取数据的长度总是扇区大小 CH374vSectorSize 的倍数。该子程序会根据 CH374vFileSize 自动检查文件是否结束，如果文件已经结束，那么返回时在 mCmdParam.ReadX.mSectorCount 中为实际读出的扇区数，判断 mCmdParam.ReadX.mSectorCount 如果变小就说明文件已经结束。如果文件长度不是 CH374vSectorSize 的倍数，那么文件尾部不足一个扇区的零碎数据部分将被忽略，如果必须读出文件尾部不足一个扇区的零碎数据，那么可以临时增大 CH374vFileSize 以读出最后一个扇区（实际有效数据不足一个扇区），然后再恢复原 CH374vFileSize，注意此时的文件指针已经无效。

CH374FileWriteX( ); /* 以扇区为单位向当前文件写入指定缓冲区的数据 */

向当前已打开文件中写入数据，每次写入后自动移动文件指针，第二次调用时将从第一次写入数据的后面继续写入数据。在调用该子程序前，应该在 mCmdParam.WriteX.mDataBuffer 中指定缓冲区起始地址，子程序返回后更新为当前缓冲区地址（也就是起始地址加上已经写入的长度），并在 mCmdParam.WriteX.mSectorCount 中指定需要写入的扇区数，所以写入数据的长度总是扇区大小 CH374vSectorSize 的倍数。该子程序会检查文件结束簇，并且在需要时会自动分配磁盘空间以便继续写入。该子程序只负责修改或者追加数据，而不修改文件长度。如果调用 CH374FileWriteX 写空数据（指定写入扇区数 mCmdParam.WriteX.mSectorCount 为 0），那么该子程序将更新文件长度。如果文件长度不是 CH374vSectorSize 的倍数，那么可以在关闭文件前调用 CH374FileModify

指定文件长度。示例：

```
mCmdParam.Locate.mSectorOffset = 0xffffffff;  /* 移动文件指针到文件末尾 */
CH374FileLocate( );  /* 移动文件指针，以便在原文件的末尾追加数据 */
mCmdParam.WriteX.mSectorCount = 2;  /* 写入 2 个扇区，追加 CH374vSectorSize*2 字节 */
mCmdParam.WriteX.mDataBuffer = 0x3600;  /* 指定被写数据的缓冲区的起始地址 */
CH374FileWriteX( );  /* 以扇区为单位向文件写入指定缓冲区的数据 */
```

CH374ByteLocate( ); /* 以字节为单位移动当前文件指针，进入字节模式 */

移动当前已打开文件的指针，用于从指定位置读取数据，或者向指定位置写入数据。例如，单片机希望跳过文件的前 18 字节再读取数据，那么可以在 mCmdParam.ByteLocate.mByteOffset 参数中输入 18，调用该子程序将文件指针移动到 18 个字节后，也就是紧接在后面的读写操作将从第 18 字节开始。对于写操作，如果单片机打算在原文件的尾部继续添加数据，而不希望影响前面的原有数据，那么可以指定一个很大的字节偏移，例如在 mCmdParam.ByteLocate.mByteOffset 参数中输入 OFFFFFFFFH，将文件指针移动原文件的末尾，以便追加数据。示例：

```
mCmdParam.ByteLocate.mByteOffset = 192; /* 移动文件指针到第 193 字节 */
CH374ByteLocate( );  /* 移动文件指针，跳过文件头部的 192 个字节 */
/* 以字节为单位进行文件读写操作，读写操作从第 193 字节的位置开始 */
mCmdParam.ByteLocate.mByteOffset = 0xffffffff; /* 移动文件指针到文件末尾 */
CH374ByteLocate( );  /* 移动文件指针，以便在原文件的末尾追加数据 */
```

CH374ByteRead( ); /* 以字节为单位从当前文件读取数据块，进入字节模式 */

从当前已打开文件中读取数据，每次读取后自动移动文件指针，第二次调用子程序时将从第一次读取数据的后面继续读取数据。输入参数应该在 mCmdParam.ByteRead.mByteCount 中指定需要读取的字节数，字节数不能超过 MAX_BYTE_IO 和 sizeof(mCmdParam.ByteRead.mByteBuffer)，子程序返回后，读出的数据块被存放在 mCmdParam.ByteRead.mByteBuffer 中。该子程序会自动检查文件是否结束，如果文件已经结束，那么返回时在 mCmdParam.ByteRead.mByteCount 中为实际读出的字节数，所以判断 mCmdParam.ByteRead.mByteCount 如果变小就说明文件已经结束。示例：

```
mCmdParam.ByteRead.mByteCount = 9; /* 准备读出 9 个字节，如果返回小于 9 则文件结束 */
i=CH374ByteRead( ); /* 以字节为单位从文件读出数据块 */
if ( i!=ERR_SUCCESS ) /* 出错 */ else /* 成功 */
if ( mCmdParam.ByteRead.mByteCount < 9 ) 已经到文件结尾，所以实际读出的长度变小了
/* 在 mCmdParam.ByteRead.mByteBuffer 中为读出的数据块 */
mCmdParam.ByteRead.mByteCount = 23; /* 准备再读出 23 个字节，接着刚才的位置向后读 */
CH374ByteRead( ); /* 以字节为单位从文件读出数据块，如果返回长度变小说明文件结束 */
```

CH374ByteWrite( ); /* 以字节为单位向当前文件写入数据块，进入字节模式 */

向当前已打开文件中写入数据，每次写入后自动移动文件指针，第二次调用子程序从第一次写入数据的后面继续写入数据。输入参数应该在 mCmdParam.ByteWrite.mByteCount 中指定需要写入的字节数，字节数不能超过 MAX_BYTE_IO 和 sizeof(mCmdParam.ByteWrite.mByteBuffer)，准备写入的数据块被存放在 mCmdParam.ByteWrite.mByteBuffer 中。该子程序会自动检查文件是否结束，并且在需要时会自动分配磁盘空间以便继续写入。该子程序只负责修改或追加数据，而不修改文件长度。如果调用 CH374ByteWrite 写空数据（指定写入字节数 mCmdParam.ByteWrite.mByteCount 为 0），那么该子程序将更新文件长度。示例：

```
mCmdParam.ByteWrite.mByteCount = 28; /* 写入 28 个字节的数据 */
/* 将准备写入的数据块复制到 mCmdParam.ByteWrite.mByteBuffer 中 */
i=CH374ByteWrite( ); /* 以字节为单位向文件写入数据块 */
if ( i!=ERR_SUCCESS ) /* 出错 */ else /* 成功 */
mCmdParam.ByteWrite.mByteCount = 7; /* 再写入 7 个字节，如果为 0 则更新文件长度 */
/* 将准备写入的数据块复制到 mCmdParam.ByteWrite.mByteBuffer 中 */
CH374ByteWrite( ); /* 以字节为单位向文件写入数据块 */
```

为了提高读写操作的效率，尽量一次读写较大的数据块，字节数不超过 MAX_BYTE_IO。

CH374DirtyBuffer( ); /* 清除磁盘缓冲区 */

CH374 的子程序库中，文件操作子程序 CH374File????和磁盘查询子程序 CH374DiskQuery 都可能会用到磁盘数据缓冲区 DISK_BASE_BUF，并且有可能在 DISK_BASE_BUF 中保存了磁盘信息，所以必须保证 DISK_BASE_BUF 不被用于其它用途，如果 RAM 资源有限，要将 DISK_BASE_BUF 临时用于其它用途，那么在临时用完后必须调用 CH374DirtyBuffer 清除磁盘缓冲区。

CH374DiskSize( ); /* 查询磁盘容量 */

查询磁盘物理总容量，调用后返回信息

mCmdParam.DiskSize.mDiskSizeSec 为整个物理磁盘的总扇区数，每扇区通常为 512 字节，实际上个别 U 盘可能会大于 512，实际值可以参考 CH374vSectorSize 变量

CH374DiskQuery( ); /* 查询磁盘信息 */

查询磁盘信息，该子程序在 FAT32 文件系统的磁盘中调用时最快，在 FAT16 文件系统的磁盘中调用时最慢，磁盘容量越大，操作越慢。调用后返回信息

mCmdParam.Query.mTotalSector 为当前磁盘的总扇区数

mCmdParam.Query.mFreeSector 为当前磁盘的剩余扇区数，为 0 时说明磁盘已满。

mCmdParam.Query.mDiskFat 为当前磁盘（第一逻辑盘）的文件系统标志：0 对应于未知文件系统，1 对应于 FAT12，2 对应于 FAT16，3 对应于 FAT32

CH374BulkOnlyCmd( ); /* 执行基于 BulkOnly 协议的命令 */

这是底层协议操作子程序，通常不需要用到，用于执行基于 BulkOnly 协议的磁盘命令，输入参数在 mBOC.mCBW 结构中，执行后在 mBOC.mCBW.mCBW_DataLen0 中返回剩余长度，如果有数据传输那么数据在 DISK_BASE_BUF 中。示例：

```
mBOC.mCBW.mCBW_DataLen0 = 0; /* 该命令没有数据传输 */
mBOC.mCBW.mCBW_Flag = 0x00;
mBOC.mCBW.mCBW_CB_Len = 6; /* 命令块长度 */
mBOC.mCBW.mCBW_CB_Buf[0] = 0x1E; /* 防止或者允许媒体移除的 SCSI 命令码 */
mBOC.mCBW.mCBW_CB_Buf[4] = 0x01; /* 防止媒体移除 */
mBOC.mCBW.mCBW_CB_Buf[2] = mBOC.mCBW.mCBW_CB_Buf[1] = 0;
mBOC.mCBW.mCBW_CB_Buf[5] = mBOC.mCBW.mCBW_CB_Buf[3] = 0;
i = CH374BulkOnlyCmd( ); /* 执行 BulkOnly 协议的命令 */
if ( i==ERR_SUCCESS ) /* 操作成功 */
```

如果有数据传输，那么剩余长度在 mBOC.mCBW.mCBW_DataLen0 中，用原先计划传输的长度减去该剩余长度就得到实际传输长度，数据保存在 DISK_BASE_BUF 缓冲区中

CH374SaveVariable( ); /* 备份/保存/恢复子程序库的变量 */

备份/保存/恢复子程序库的变量，用于子程序库在多个 CH374 芯片之间进行切换，以及用于在通过 USB-HUB 连接多个 U 盘时进行多个 U 盘之间的切换。如果另外的 CH374 芯片使用过 DISK_BASE_BUF，那么还需要另外调用 CH374DirtyBuffer 清除磁盘缓冲区。

CH374HostTransact( ); /* 执行 USB 基本传输事务 */

执行 USB 基本传输事务，用于实现最底层的 USB 传输。调用前应该在 CH374 芯片的缓冲区中准备好待发数据，设置好 CH374UsbPidIn 标志，设置 CH374 的寄存器 REG_USB_H_PID 为所需要的 PID 事务令牌和目的端点地址，设置 CH374 的寄存器 REG_USB_H_CTRL 为所需要的同步标志并设置启动传输位 BIT_HOST_START，然后再调用该子程序。USB 传输完成后，CH374 产生传输完成中断，经过该子程序分析处理，最终返回中断状态，该状态除 USB_INT_SUCCESS 改为 ERR_SUCCESS 之外，完全同 CH375 芯片。该子程序自动处理出错重试，并在收到 NAK 应答时，根据 CH374vRetryCount 中的位 7 决定是否进行重试。对于数据 IN 操作，可以返回成功后从 CH374 芯片的缓冲区中获取数据。

CH374CtrlTransfer( ); /* 执行 USB 控制传输 */

执行 USB 控制传输，支持无数据控制操作、控制写数据操作，但不支持控制读数据操作。调用前必须向 CH374 芯片的主机发送缓冲区 RAM_HOST_TRAN 中写入 8 字节的控制传输请求码，然后再调用该子程序。该子程序自动处理出错重试，最终返回状态同 CH374HostTransact 子程序。如果有数据进行传输，那么收发的数据在 DISK_BASE_BUF 缓冲区中。

CH374DelaymS(UINT8 iDelay); /* 延时指定毫秒，输入参数为毫秒数，不大于 255 */

延时指定毫秒，有效值为 1 毫秒到 255 毫秒。误差不超过 1 毫秒，且不受单片机速度影响。

CH374EmbHubAttach(void); /* 未声明子程序，检查当前的内置 HUB 端口是否有 USB 设备 */

输入在 CH374vEmbHubIndex 中指定内置 HUB 的端口号，0 表示禁止使用内置 HUB，1/2/3 分别指定相应的端口号。返回：0 表示该端口上未检测到 USB 设备，非 0 表示检测到 USB 设备，如果该值为中为(BIT_HUB0_ATTACH & BIT_HUB0_EN)那么说明有设备且已被枚举（未变化过）。

6.3 由应用程序提供的子程序

xQueryInterrupt( ) /* 查询 CH374 中断 */

适用于查询方式，用于查询 CH374 是否产生中断，等待产生中断后才返回。当 CH374 检测到 USB 设备插拔事件或者 USB 传输操作完成时，输出 INT#引脚为低电平，该子程序返回。

在 CH374HF?.H 头文件中提供了该子程序，不过，在主程序中可以定义 NO_DEFAULT_CH374_INT 禁止头文件中的该子程序，然后在主程序中自行定义一个特殊的 xQueryInterrupt 子程序。

xDelayAfterWrite( ) /* 写操作后延时 */

写操作后延时程序，应该根据 U 盘的实际情况进行调整，建议为 200uS 左右。

在 CH374HF?.H 头文件中提供了该子程序，不过，在主程序中可以定义 NO_DEFAULT_DELAY_WRITE 禁止头文件中的该子程序，然后在主程序中自行定义一个更实用的 xDelayAfterWrite 子程序。

xFileNameEnumer( ) /* 文件名枚举回调子程序 */

文件名枚举回调子程序，应该根据实际枚举需求进行调整，.H 头文件中只是参考的例子。

该回调子程序用于配合 CH374FileOpen 实现一次调用多次枚举，从而提高搜索文件的速度。

xDiskSectorAccess(PUINT32I mLba, UINT8 mMode) /* 以扇区为单位存取磁盘 */

如果库开放外部扇区接口 EXT_SEC_INTERFACE，那么应用程序就必须提供该子程序。当启用磁盘存取的外部接口后，所有的磁盘存取操作将转发给该子程序处理，该子程序由 CH374 子程序库调用。

6.4 硬件 I/O 接口子程序

硬件 I/O 接口子程序用于单片机与 CH374 芯片之间交换数据。除了部分 MCS51 总线 I/O 子程序库内置了 I/O 接口子程序之外，其它的子程序库都需要由应用程序为其提供硬件 I/O 接口子程序，应用程序应该根据实际的硬件连接方式（并口或者 SPI）实现 I/O 接口子程序，子程序库本身不依赖于硬件 I/O 接口方式。

以下 I/O 接口子程序，后三个可以由前几个子程序为基础生成，对 U 盘读写速度影响最大的子程序是 CH374_READ_BLOCK64 和 CH374_WRITE_BLOCK64 以及 CH374_WRITE_BLOCK。

CH374_READ_REGISTER(UINT8 mAddr); /* 从指定寄存器读取数据 */

单字节读，从 CH374 芯片的地址为 mAddr 的寄存器或缓冲区中读取一个字节的数据并返回

CH374_WRITE_REGISTER(UINT8 mAddr, UINT8 mData); /* 向指定寄存器写入数据 */

单字节写，向 CH374 芯片的地址为 mAddr 的寄存器或缓冲区中写入一个字节的数据 mData

CH374_READ_BLOCK(UINT8 mAddr, UINT8 mLen, PUINT8 mBuf);/* 从指定起始地址读出数据块 */
多字节连续读，从 CH374 芯片的地址为 mAddr 的寄存器或缓冲区开始，依次读取 mLen 个字节的数据到 mBuf 缓冲区中

CH374_WRITE_BLOCK(UINT8 mAddr, UINT8 mLen, PUINT8 mBuf);/* 向指定起始地址写入数据块 */
多字节连续写，从 CH374 芯片的地址为 mAddr 的寄存器或缓冲区开始，将 mBuf 缓冲区中的 mLen 个字节的数据依次写入

CH374_READ_BLOCK64(UINT8 mAddr, PUINT8 mBuf);/* 读出 64 字节的数据块，返回当前地址 */
连续读 64 字节，从 CH374 芯片的地址为 mAddr 的缓冲区开始，依次读取 64 个字节的数据到 mBuf 缓冲区中，并返回新的缓冲区地址（即 mBuf+64）

CH374_WRITE_BLOCK64(UINT8 mAddr, PUINT8 mBuf);/* 写入 64 字节的数据块，返回当前地址 */
连续写 64 字节，从 CH374 芯片的地址为 mAddr 的缓冲区开始，将 mBuf 缓冲区中的 64 个字节的数据依次写入，并返回新的缓冲区地址（即 mBuf+64）

CH374_WRITE_BLOCK_C(UINT8 mLen, PUINT8 mBuf);/* 向 RAM_HOST_TRAN 写入常量型数据块 */
向主机发送缓冲区写入多个字节，将 mBuf 缓冲区中的 mLen 个字节的数据依次写入 CH374 芯片的主机发送缓冲区 RAM_HOST_TRAN，主要是写入常量字符串，用于控制传输等。

6.5 由应用程序提供的变量或存储单元

mCmdParam; /* 命令参数结构，用于在调用 CH374 子程序时输入参数和返回结果 */
mBOC; /* BulkOnly 协议的命令包 */
DISK_BASE_BUF; /* 外部 RAM 的磁盘数据缓冲区，缓冲区长度为一个扇区的长度 */
该缓冲区不是必须的，如果在包含 CH374HF?.H 文件之前定义 DISK_BASE_BUF_LEN 为 0，那么将禁止定义该缓冲区，而是由应用程序在调用 CH375Init 进行初始化之前将自行定义的缓冲区起始地址置入 pDISK_BASE_BUF 变量中，从而能够与其它应用程序合用一个缓冲区以节约 RAM
FILE_DATA_BUF; /* 可选，外部 RAM 的文件数据缓冲区，缓冲区长度不小于一次读写的数据长度 */
以字节为单位读写 U 盘文件不需要用到 FILE_DATA_BUF，所以可以不分配 RAM。

6.6 针对 MCS51 的速度优化

由于 MCS51 单片机复制外部 RAM 中的数据时比较慢，所以在 U 盘文件子程序库之外又提供了几种 I/O 接口子程序库，用于针对不同的硬件环境优化速度，可在链接时选择相应的 I/O 接口库。

C51DPTR1.LIB：″单 DPTR 复制″，最常规的数据复制方式，使用一个 DPTR 来回切换，每传输一个字节需要 11.25 个机器周期，速度最慢，适用于所有 MCS51 单片机

C51DPTR2.LIB：″双 DPTR 复制″，针对特定硬件的数据复制方式，使用两个 DPTR，每传输一个字节需要 8.25 个机器周期，速度较快，适用于 ATMEL/PHILIPS/SST 等具有双 DPTR 的单片机

C51P2R0.LIB：″单 DPTR 和 P2+R0 复制″，用 P2+R0 指向 CH374 的 I/O 端口并且用 DPTR 指向外部 RAM 进行数据复制，每传输一个字节需要 6.25 个机器周期，速度最快，适用于所有标准 MCS51 单片机，但是某些单片机在启用内置的外部 RAM 时会关闭 P2+R0 功能而可能不适用

7、示例

各子程序库的子程序调用方式完全统一，同一种单片机的示例程序完全通用，只需要在链接时指定不同的子程序库就可以实现不同的功能。不同单片机的 C 语言示例程序基本通用，只需要修改硬件

相关部分，main 主程序通常无需修改，重新编译和链接就可以使用。

目前为 MCS51 单片机提供了多个示例程序，其它单片机也可以参考。

7.0 单片机控制 USB 主机模式切换，示例 EXAM0

C 语言示例程序，演示 USB-HOST 主机接口和 USB-DEVICE 设备接口的应用。

默认工作于 USB-HOST 主机方式，当有 USB 设备连接时自动处理，需要作为 USB 设备与计算机通讯时，可以按评估板上的按钮由主程序进行切换。

主机方式下，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，所以速度较慢，适用于所有 MCS51 单片机，需要 1K 以上的 RAM 空间。插入 U 盘后，该程序将 U 盘中的/C51/CH374HFT.C 文件中的前 600 个字符显示出来，如果找不到文件，那么该程序将显示当前目录下所有文件名。

设备方式下，CH374 的 INT#引脚采用中断方式，程序结构为“请求+应答模式”，强调可靠性和交互性，不追求传输速度。计算机端可以通过 CH372/CH375/CH374 的调试工具中的 MCS51 监控工具程序 CH37XDBG.EXE 实现对 MCS51 单片机的“完全”控制，可以读写 MCS51 单片机的任意外部 RAM、内部 RAM 以及绝大多数 SFR，当然也能够进行数据通讯。

涉及到 USB 主从模式切换，字节模式的文件读写，文件打开/关闭/新建/删除，USB 设备方式下的数据通讯等。

7.1 扇区模式读写，查询，示例 EXAM1

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，所以速度较慢，适用于所有 MCS51 单片机，需要 1K 以上的 RAM 空间。

该程序将 U 盘中的/C51/CH374HFT.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 CH374HFT.C，那么该程序将显示 C51 子目录下所有以 CH374 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。该程序使用了 32K 外部 RAM。

涉及到扇区模式的文件读写，文件打开/关闭/新建/删除，文件枚举/搜索等。

7.2 扇区模式读写，中断，MCS51 双 DPTR，示例 EXAM2

C 语言示例程序，CH374 的 INT#引脚采用中断方式处理，数据复制方式为“双 DPTR 复制”，所以速度较快，适用于 ATMEL/PHILIPS/SST 等具有双 DPTR 的单片机。

其它功能与 EXAM1 相同。注意，由于写 U 盘文件耗时较多，所以中断处理过程占用很多时间。

7.3 字节模式读写，使用较少 RAM，示例 EXAM6

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，兼容性最好但是速度最慢，该程序只使用 512 字节的外部 RAM，不需要文件数据缓冲区 FILE_DATA_BUF，适用于所有 MCS51 单片机。

该程序将 U 盘中的/C51/CH374HFT.C 文件中的前 600 个字符显示出来。如果找不到原文件 CH374HFT.C，那么该程序将显示 C51 子目录下所有以 CH374 开头的文件名。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。最后将程序 ROM 中的一个字符串写入新建的文件 NEWFILE.TXT 中。

涉及到字节模式的文件读写，文件打开/关闭/新建/删除，文件枚举/搜索等。

7.4 比较实用的字节模式应用，示例 EXAM7

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，兼容性最好但是速度最慢，该程序只使用 512 字节的外部 RAM，不需要文件数据缓冲区 FILE_DATA_BUF，

适用于所有MCS51单片机。

该程序用于演示将ADC模数采集的数据保存到U盘文件MY_ADC.TXT中。该程序以字节为单位读写U盘文件，读写速度较扇区模式慢，但是由于字节模式读写文件不需要文件数据缓冲区FILE_DATA_BUF，所以总共只需要620字节的RAM（含80字节内部RAM），适用于单片机硬件资源有限、数据量小并且读写速度要求不高的系统。

该程序检查MY_ADC.TXT文件是否存在，存在则添加数据，不存在则新建文件。

涉及到字节模式的文件读写，文件打开/关闭/新建/删除，文件枚举/搜索，字节模式的移动文件指针/添加数据等。

7.5 比较实用的扇区模式应用，示例EXAM8

C语言示例程序，CH374的INT#引脚采用查询方式处理，数据复制方式为“单DPTR和P2+R0复制”，速度最快，如果用于带有内置XRAM的单片机需改用其它复制方式。

本程序用于演示将ADC模数采集的数据保存到U盘文件MY_ADC.TXT中。该程序以扇区为单位读写U盘文件，读写速度较字节模式快，由于扇区模式以扇区为基本单位，对于需要处理零碎数据的应用，不如字节模式方便。

该程序演示在扇区模式下处理零碎数据，同时兼顾操作方便和较高速度，需要不少于1K长度的文件数据缓冲区FILE_DATA_BUF，该程序使用了16K外部RAM。

涉及到扇区模式的文件读写，文件打开/关闭/新建/删除，文件枚举/搜索，扇区模式的移动文件指针/添加数据，以及非扇区大小CH374vSectorSize倍数的任意文件长度的处理等。

7.6 创建子目录的应用，示例EXAM9

C语言示例程序，CH374的INT#引脚采用查询方式处理，数据复制方式为“单DPTR复制”，兼容性最好但是速度最慢，适用于所有MCS51单片机。

本程序用于演示在根目录下创建子目录，以及在该子目录下再创建二级子目录，由于FAT12或者FAT16的文件系统中，根目录下不能超过512个文件或者目录，所以在需要大量产生新文件或者需要按文件类型进行分类处理时，可以参考该程序。

涉及到扇区模式的文件（目录）写，文件（目录）打开/关闭/新建等。

7.7 处理文件目录项的应用，示例EXAM10

C语言示例程序，CH374的INT#引脚采用查询方式处理，数据复制方式为“单DPTR复制”，兼容性最好但是速度最慢，适用于所有MCS51单片机。

本程序用于演示处理文件目录项，例如：修改文件名，设置文件的创建日期和时间等。

涉及到文件目录项的结构、查询和修改文件信息等。

7.8 处理小写文件名和长文件名的应用，示例EXAM11

C语言示例程序，CH374的INT#引脚采用查询方式处理，数据复制方式为“单DPTR复制”，兼容性最好但是速度最慢，适用于所有MCS51单片机。

本程序用于演示处理小写文件名和长文件名，仅供具有一定技术水平和相关知识的高级用户参考。例如：获取某文件对应的长文件名，创建长文件名的文件等。

涉及到新建/打开/关闭目录，扇区模式的目录读写，文件目录项的结构，长文件名结构等。

7.9 检查U盘写保护和安全移除U盘，示例EXAM12

C语言示例程序，CH374的INT#引脚采用查询方式处理，数据复制方式为“单DPTR复制”，兼容性最好但是速度最慢，适用于所有MCS51单片机。

本程序用于演示检查U盘是否写保护，演示模拟计算机端的安全移除，也可以参考用于自行处理

其它命令。

涉及到 USB 配置、USB MassStorage 和 SCSI 规范，新建文件/关闭，字节模式写文件等。

7.10 搜索和枚举文件名的应用，示例 EXAM13

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，兼容性最好但是速度最慢，适用于所有 MCS51 单片机。

本程序用于演示搜索和枚举文件名，列出指定目录下的所有文件，为整个 U 盘的文件建立索引表，优化某些慢启动 U 盘执行 CH374DiskReady 时的等待时间等，仅供具有一定技术水平和相关知识的高级用户参考。

涉及到打开/关闭目录，扇区模式的目录读操作，文件目录项的结构等。

7.11 通过内置 HUB 同时连接 U 盘和键盘鼠标，示例 EXAM14

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，兼容性最好但是速度最慢，适用于所有 MCS51 单片机。

本程序用于通过 CH374U 的 3 端口根集线器 ROOT-HUB 同时连接最多三个 USB 设备，包括 U 盘、USB 打印机、USB 键盘、USB 鼠标等，演示读写 U 盘中的文件，演示操作 USB 鼠标等，仅供具有一定技术水平和相关知识的高级用户参考。有关外部 HUB 级联请参考 EVT\EMB_HUB\ROOTHUB.C 程序。

涉及到 CH374U 的 ROOT-HUB 根集线器、打开/删除/新建/关闭文件，字节模式的文件读写操作等。

7.12 同时连接两个 U 盘并在之间复制文件，示例 EXAM15

C 语言示例程序，CH374 的 INT#引脚采用查询方式处理，数据复制方式为“单 DPTR 复制”，兼容性最好但是速度最慢，适用于所有 MCS51 单片机，例子程序需要 32K 的外部 RAM 存储器。

本程序用于通过 CH374U 的 3 端口根集线器 ROOT-HUB 同时连接两个 U 盘，并将一个 U 盘中的指定类型的文件（例子中指定*.TXT 文件，当然也可以是任意文件）复制到另一个 U 盘中。

先插入的 U 盘为源盘，后插入的 U 盘为目的盘，如果两个 U 盘在上电前已经插入则按 HUB 端口顺序，本程序自动将源盘中的指定类型的文件复制到目的盘中，然后等待另一组新的源盘和目的盘重新插入，复制中途有串口打印的进度显示。仅供具有一定技术水平和相关知识的高级用户参考。

涉及到 CH374U 的 ROOT-HUB 根集线器、打开/新建/关闭文件，扇区模式的文件读写操作，备份和恢复子程序库的变量 CH374SaveVariable 等。