JMBIE MP3＋U 盘 学习板指导书

一. 硬件概述

1. 硬件方案

MP3 和 U 盘功能的实现主要依赖于 MP3 解码器、音频接口和 DA、USB 通信电路、FLASH 存储器以及 MCU。可选的方案有多种，但选用 AT89C51SND1 单片机来构成 MP3 和 U 盘具有明显的性价比和可行性。因为这款单片机内含了 MP3 硬件解码器和数字音频输出电路、USB 接口电路从而使得方案变的简单易行；AT89C51SND1 也是基于 8 位 C51 核的单片机，所以程序设计也很简单，与普通的 8051 单片机完全兼容。该单片机内部提供的 2K SRAM 等资源也为实现 MP3 和 U 盘功能作为保证，同时具有 MMC 接口、ATA 接口以及充足的 IO 端口，来扩展 FLASH 存储器等介质芯片。

图 1 是本 MP3 和 U 盘学习板的硬件框架：

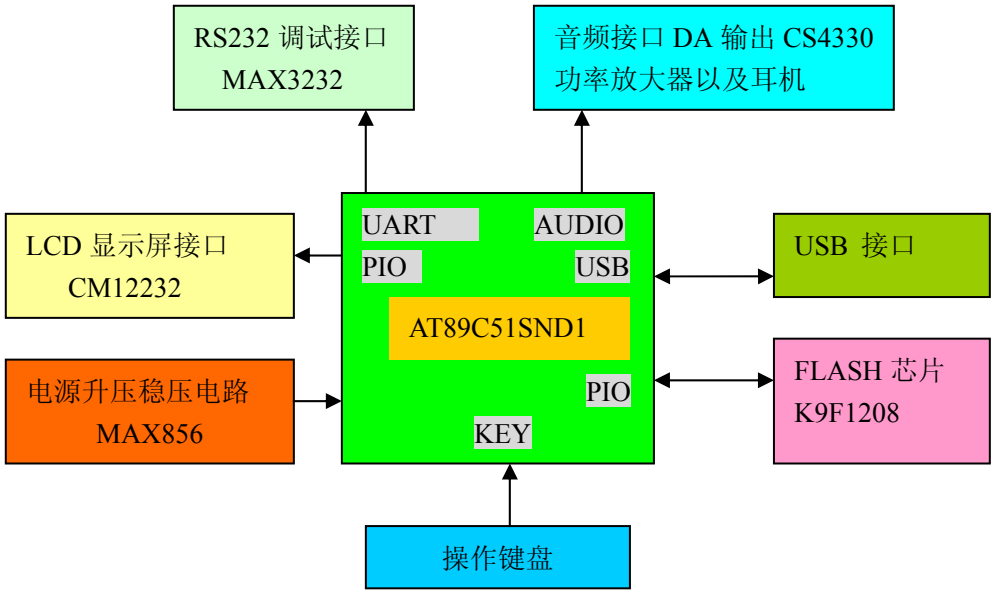


图 1 MP3＋U 盘学习板硬件构成

从图中可以看出，整个学习板由单片机 AT89C51SND1、64M FLASH 存储器 K9F1208、UART 转 RS232 芯片 MAX3232、音频 DA 芯片 CS4330 和功放芯片 TDA2822、电源升压芯片 MAX856 和稳压芯片 AS1117-3.3、操作按键和接插件等构成，并可以扩展型号为 CM12232 兼容 LCD。

首先 AT89C51SND1 可以作为普通单片机来使用其一般功能，比如串口和定时器等。这里为了调试方便，提供了 RS232 串口电路和 DB9 连接器。FLASH 芯片是构成 U 盘的重要组成部分，MCU 通过 USB 部件来实现和 PC 机的数据通信，并按 USB 海量存储的相关协议响应 PC 机的命令，将数据写入 FLASH 或从中读取数据，同时 FLASH 芯片内部逻辑结构将按磁盘文件系统来组织。FLASH 芯片是通过数据总线和若干 IO 端口和 MCU 连接的，详细情况见原理图。MP3 功能中，MCU 从 FLASH 芯片中读取歌曲数据，再送给 MP3 硬件解码器的缓冲区，解码器的输出经过音频部件输出到外部 DA 芯片，转换为模拟音频信号后经过功率放大器即可推动耳机。

学习板的电源电路支持从 USB 的 5V 电压中取电，也支持外部输入电源。前者用 AS1117-3.3

芯片稳压为 3.3V 供给大部分电路，后者则可使用两节干电池为电路供电。在使用外部输入电源的情况下，如果输入电压低于 3.3V 则利用升压芯片 MAX856 升压为 3.3V，如果输入电压大于 4.5V 的话则利用 AS1117-3.3 降压为 3.3V，这点需要通过跳线选择的。

本学习板可以扩展一个 LCD 液晶屏，型号为 CM12232 的点阵模块，或与此兼容的其他 12232 模块，具体显示中的问题可能因型号不同而异，需要依靠软件解决。在硬件上 LCD 通过 IO 端口和 MCU 连接。但是 LCD 的电源电压是 5V 的，需要从 USB 取电或外部输入 5V 电压。

2. 芯片摘要

上述的各个芯片详细的技术细节请参考英文的 DataSheet，本学习板的配套光盘里有相关的 PDF 文件。在此只给出部分说明。

1） AT89C51SND1

AT89C51SND1 是 ATMEL 公司的基于 8 位 C51 核的单片机，与 AT89C51 等 8051 单片机兼容。该芯片共 80 引脚，封装为 TQFP80，就是四边各 20 引脚的贴片封装。

AT89C51SND1 内部主要资源见图 2：

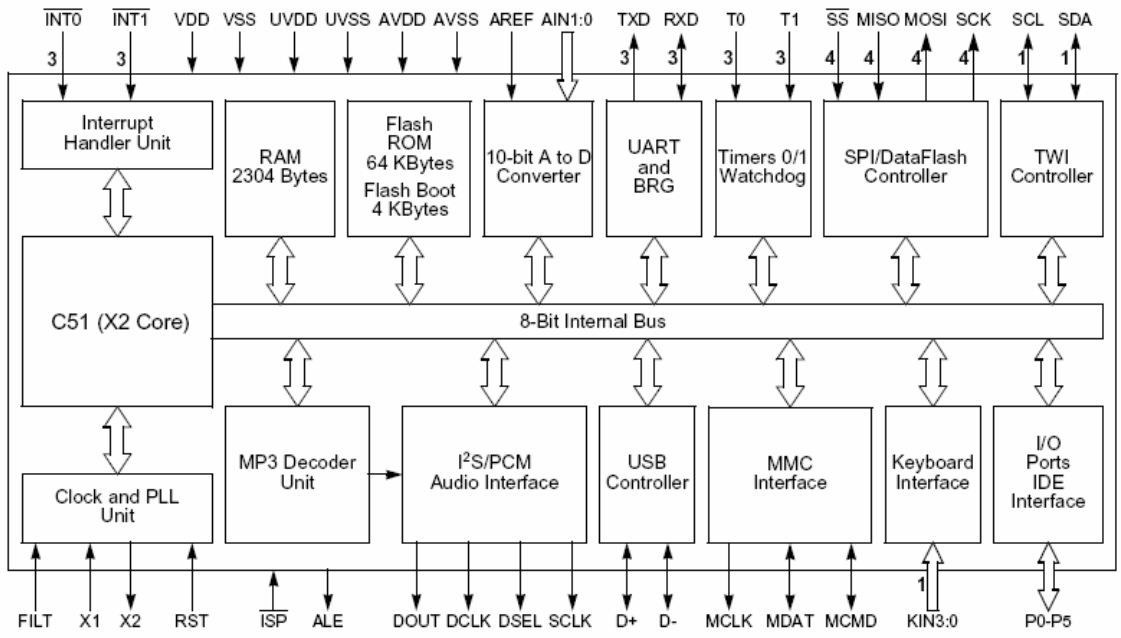


图 2　AT89C51SND1 内部架构框图

图 3 是 AT89C51SND1 的引脚定义。对于功能复用的引脚要慎重考虑。本学习板并没有使用单片机的全部功能，部分没有占用的 IO 端口和 MMC 等专用接口都用扩展槽引出了，用户可以自己扩展其他电路。

光盘里给出了 AT89C51SND1 的电路原理图库和 PCB 封装库元件，可以用来设计其他基于该单片机的系统。

图 3-1 是 AT89C51SND1 的数据存储器配置，内部具有 2K 大小的 ERAM 扩展数据 RAM。由于 MP3 的程序需要较大的数据空间，所以编译的时候必须使用该空间才能保证软件正常运行。AT89C51SND1 通过寄存器位 EXTRAM 来控制在 0x00－0x7ff 的外部存储器地址空间是选择片内 ERAM 还是片外 XRAM，都是通过 MOVX 指令访问的。另外 ERAM 的大小在 2K 范围内还是可变的，而且可以设置当用 MOVX 指令访问 XRAM 时是否在 P2 口上产生地址信号，有利于 P2 口保留为 IO 端口。具体介绍请参考 AT89C51SND1 的英文数据手册。

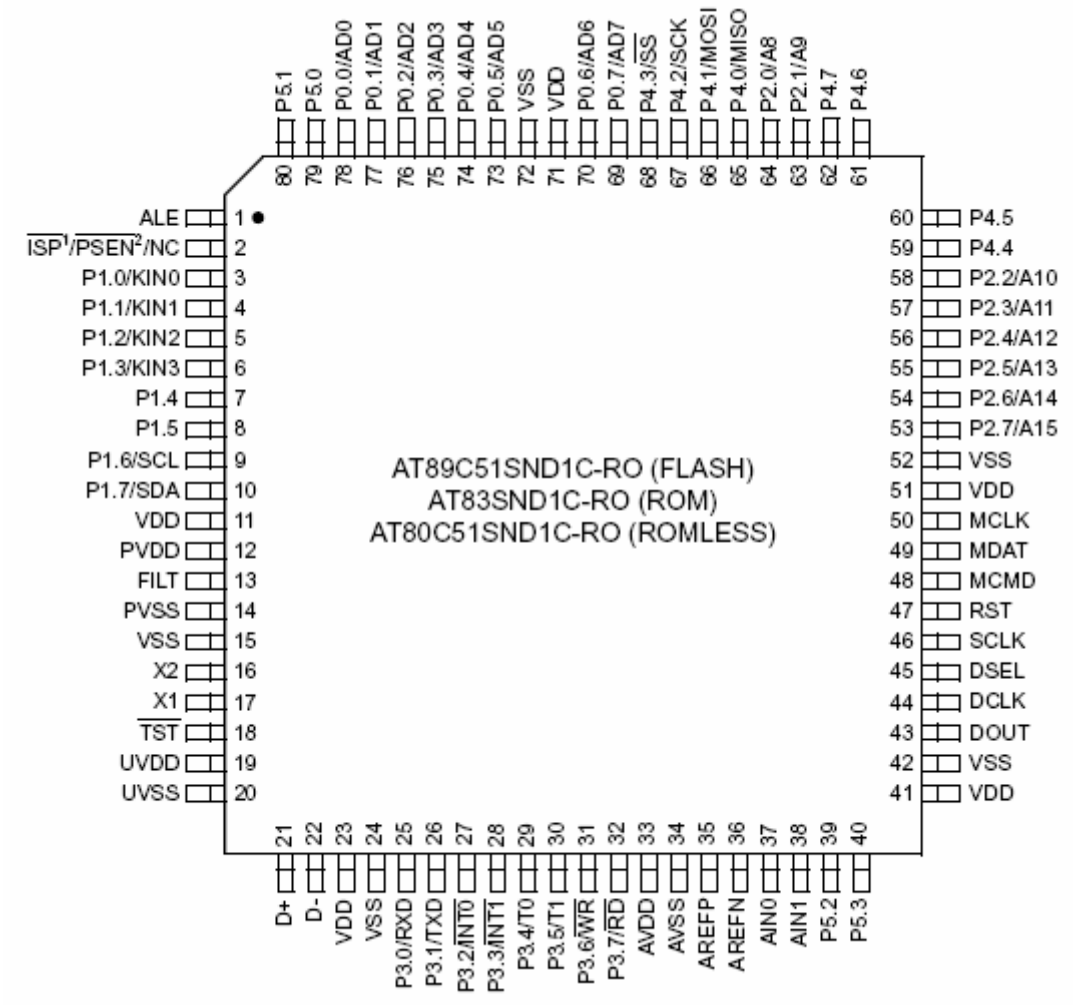


图 3　AT89C51SND1 的引脚定义

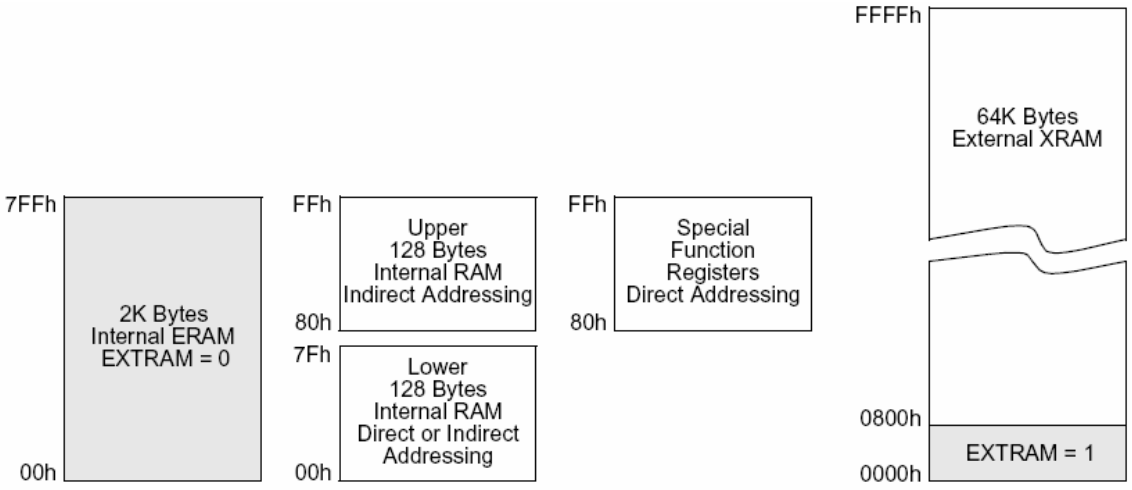


图 3-1　AT89C51SND1 的存储器配置

2） NAND FLASH　K9F1208

K9F1208 是 64M 容量的 NAND Flash 芯片，韩国三星公司产品。这种 FLASH 芯片被大量的应用在各种需要存储的设备上，比如 U 盘，商用 MP3 和数码相机等。因为 AT89C51SND1 单片机没有 NAND FLASH 的控制器，所以只能用 IO 端口来完成对该芯片的读写时序。

本学习板采用常见的 TSOP48 引脚封装，引脚定义如图 4 所示。这种芯片只有 8 位数据/地址复用接口和 CLE、ALE、WE、RE 等控制信号，硬件接口比较简单，但读写时序相对复杂，需要依次送入命令，地址，数据等，而内部存储矩阵也是按块页逻辑结构组织的，所以相对于具有独立地址线并线性编址的 NOR FLASH 来说，有的时候称其为非线性 FLASH。

K9F1208 的内部结构如图 5 所示，存储矩阵的块页逻辑结构如图 6 所示。K9F1208 共有 4K 个 Block（块），每个 Block 包括 32 个 Page（页），每个页则包含 528 Bytes，其中有 512 Bytes 基本空间和 16Bytes 扩展空间。这种逻辑结构很容易和磁盘文件系统联系起来，文件系统中的扇区也是 512 字节。K9F1208 需要 26 位地址来定位操作的块号和页号，也就是 4 个周期写入地址信息，见图 7 所示。FLASH 最小读写操作单位是页，而擦除是以块为单位进行的。在进行某种操作前先打入相应命令，K9F1208 的命令集如图 8 所示。

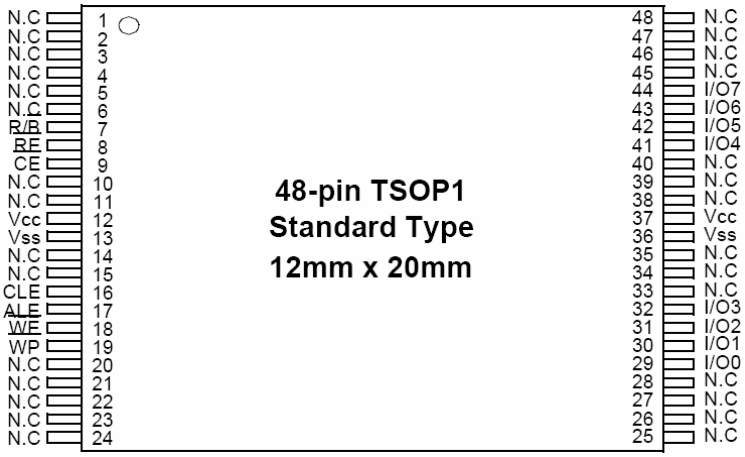


图 4　NAND FLASH　K9F1208 芯片引脚定义

Pin Name	Pin Function
I/O0 ~ I/O7	Data Input/Outputs
CLE	Command Latch Enable
ALE	Address Latch Enable
$\overline{CE}$	Chip Enable
$\overline{RE}$	Read Enable
$\overline{WE}$	Write Enable
$\overline{WP}$	Write Protect
R/$\overline{B}$	Ready/Busy output
Vcc	Power(+2.7V~3.6V)
Vss	Ground
N.C	No Connection

图 4-1 K9F1208 Flash 芯片引脚功能

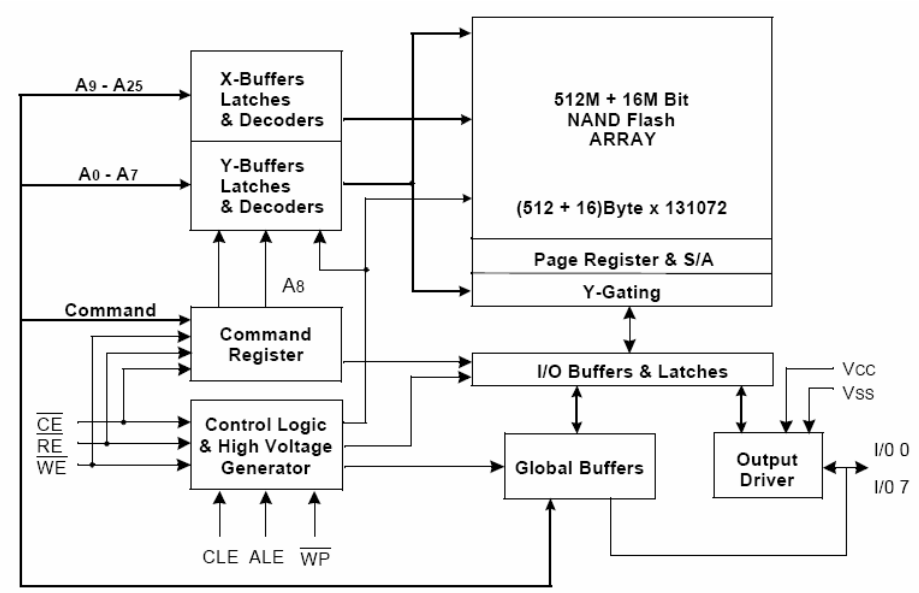


图 5　K9F1208 Flash 芯片的组成结构

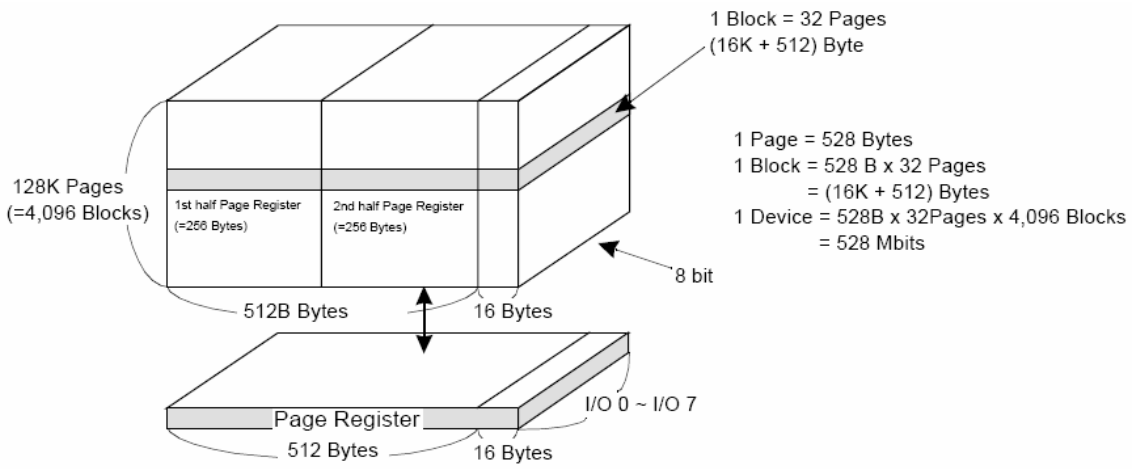


图 6　K9F1208 Flash 存储器的逻辑结构

	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7	
1st Cycle	A0	A1	A2	A3	A4	A5	A6	A7	Column Address
2nd Cycle	A9	A10	A11	A12	A13	A14	A15	A16	Row Address (Page Address)
3rd Cycle	A17	A18	A19	A20	A21	A22	A23	A24	
4th Cycle	A25	*L	*L	*L	*L	*L	*L	*L	

NOTE : Column Address : Starting Address of the Register.
00h Command(Read) : Defines the starting address of the 1st half of the register.
01h Command(Read) : Defines the starting address of the 2nd half of the register.
* A8 is set to "Low" or "High" by the 00h or 01h Command.
* L must be set to "Low".
* The device ignores any additional input of address cycles than reguired.

图 7　K9F1208 的地址周期

Function	1st. Cycle	2nd. Cycle	3rd. Cycle
Read 1	00h/01h(1)	-	-
Read 2	50h	-	-
Read ID	90h	-	-
Reset	FFh	-	-
Page Program (True)(2)	80h	10h	-
Page Program (Dummy)(2)	80h	11h	-
Copy-Back Program(True)(2)	00h	8Ah	10h
Copy-Back Program(Dummy)(2)	03h	8Ah	11h/10h
Block Erase	60h	D0h	-
Multi-Plane Block Erase	60h----60h	D0h	-
Read Status	70h	-	-
Read Multi-Plane Status	71h(3)	-	-

图 8　K9F1208 的命令集

K9F1208 的操作时序和格式在代码分析部分将作介绍。

3） CS4330 音频 DA 芯片

音频 DA 芯片在 MP3 播放器中承担音频信号的数字/模拟转换功能，其性能直接影响最终的音乐效果和音质。本 MP3 学习板采用 CS4330，该芯片没有配置引脚，无需软件配置。CS4330 的内部框图如图 9 所示。SDATA 是位流数据输入，SCLK 是位流时钟，LRCK 是声道选择时钟，其频率即采样率。MCLK 则是 DAC 电路所需的主时钟。

CS4330 支持的数字音频格式如图 10 所示。

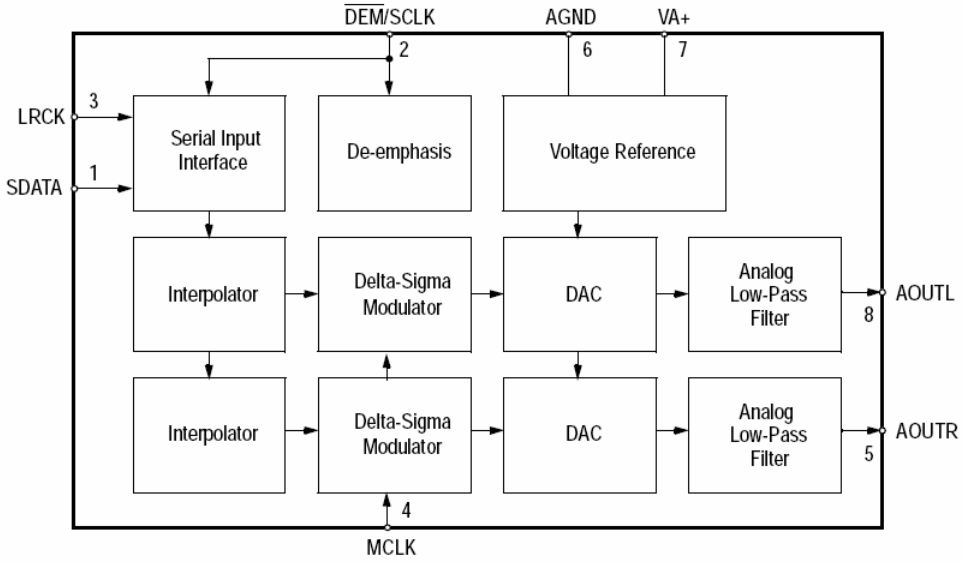
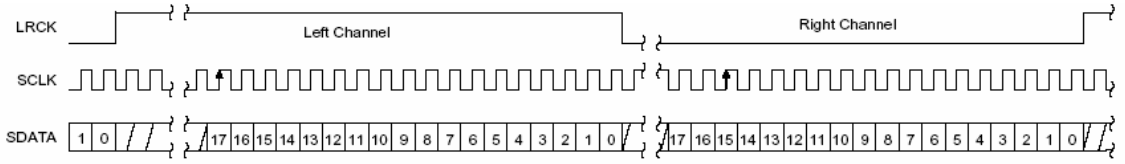


图 9　CS4330 的内部框图



Internal SCLK Mode	External SCLK Mode
Right Justified, 18-Bit Data Data Valid on Rising Edge of SCLK INT SCLK = 64 Fs if MCLK/LRCK = 256 or 512 INT SCLK = 48 Fs if MCLK/LRCK = 384	Right Justified, 18-Bit Data Data Valid on Rising Edge of SCLK SCLK must have at least 36 cycles per LRCK

图 10　CS4330 的数据格式

从图中可以看出，CS4330 支持右对齐的 18 位数据格式，工作于外部 SCLK 模式。在软件中应该正确配置 AT89C51SND1 的音频部件，选择这种数据格式保证正确工作。

3. 电路原理图

光盘里提供完整的电路原理图文件，在此只对部分电路作局部分析。

AT89C51SND1 单片机各引脚在本学习板的具体定义和功能划分见图 11。RESET 引脚在单片机内部有下拉电阻，所以只需一个上拉电容即可完成上电复位。ISP 按键在后边关于下载用户程序的部分中介绍。

AT89C51SND1 的引脚使用情况分类说明如下：

1）键盘中断输入引脚 KIN0~KIN3，和 P1.0~P1.3 复用。外接 4 个键盘作为 MP3 控制。键盘电路见图 12。

2）时钟电路 FILT、OSCX1 和 OSCX2。时钟电路见图 13。接在 FILT 引脚的阻容网络是单片机内部 PLL 锁相环的滤波电路，以给 USB 和 MP3 部件提供更高的频率。OSCX1 和 OSC2 外接晶体振荡器，为系统提供基本的 20M 时钟。

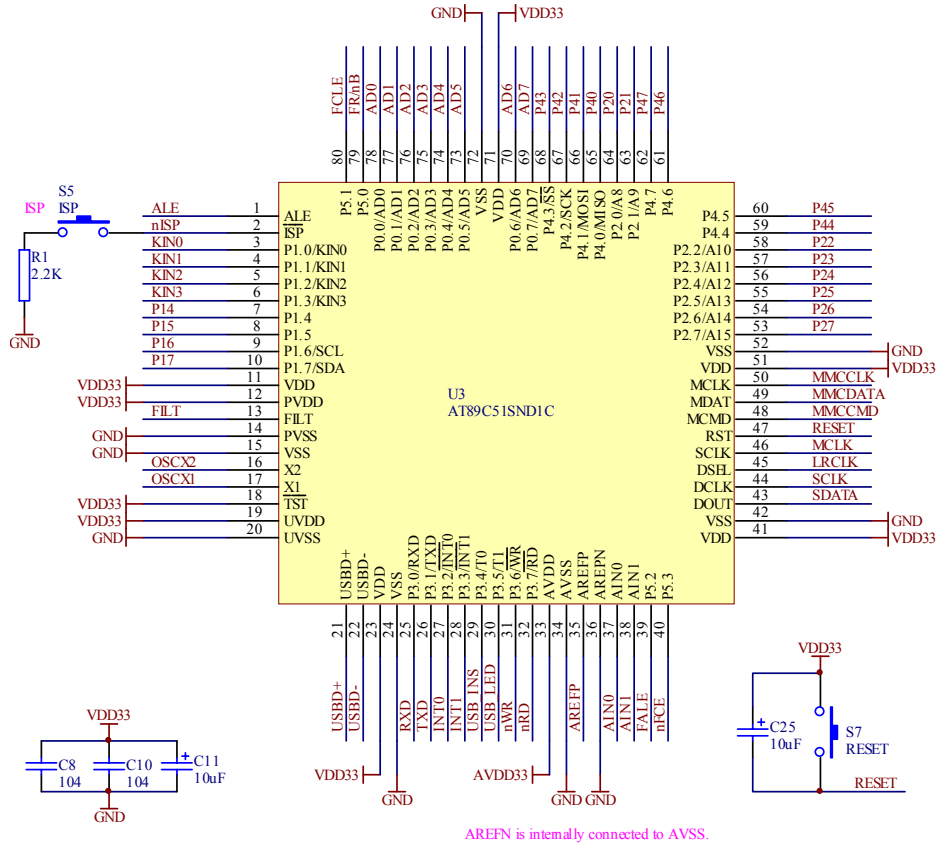


图 11　AT89C51SND1 电路原理图

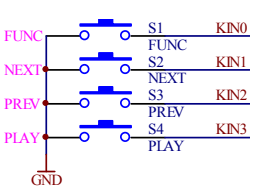


图 12　键盘电路

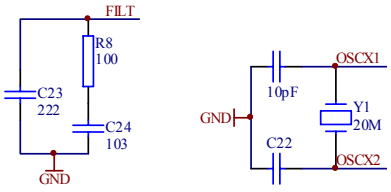


图 13　时钟电路

3）USB 接口 USBD+和 USBD-。这两个引脚就是单片机内部 USB 部件的差分信号输入输出引脚，通过图 14 所示的电路接到 USB Device 插口上。USB_INS 定义的是 IO 端口 P3.4，用来检测 USB 电缆上的 5V 电压，判断是否插入 USB 电缆，以决定是 U 盘功能还是 MP3 功能。

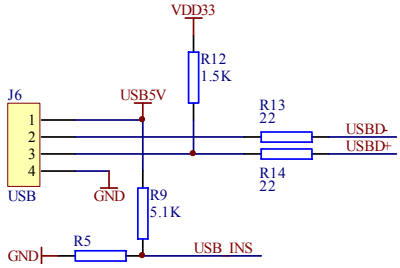


图 14　USB 电路

4）UART-RS232 电路 TXD、RXD 引脚，通过图 15 的串口电路接到一个 DB9 标准串口插座上，可以连接 PC 机用串口调试助手或超级终端等工具来查看学习板输出的提示。

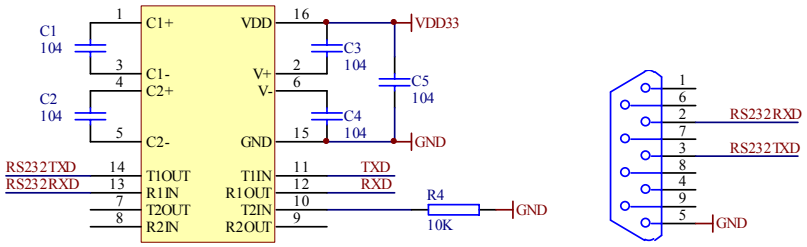


图 15 UART-RS232 电路

5）ADC 电路引脚 AREFP、AREFN、AIN0 和 AIN1。这些引脚在学习板上没有使用，但用一个 5 芯插排引出，以供用户使用。如图 16 所示。AREFN 已经接地。

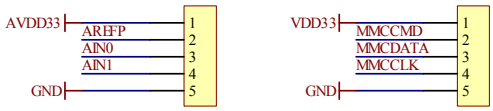


图 16　ADC 引脚和 MMC 引脚扩展

6）MMC 扩展引脚 MMCCMD、MMCDATA、MMCCLK。这是单片机的 MMC 部件接口，本学习板并没有使用，但用一个 5 芯插排引出，以供用户使用。如图 16 所示。

7）FLASH 芯片接口引脚：FALE（P5.2）、nFCE（P5.3）、FCLE（P5.1）、FR/nB（P5.0）。除此之外还有读写控制 nWR、nRD 以及 8 数据线 AD0~AD7 接到 FLASH 芯片。这部分电路如图 17 所示。

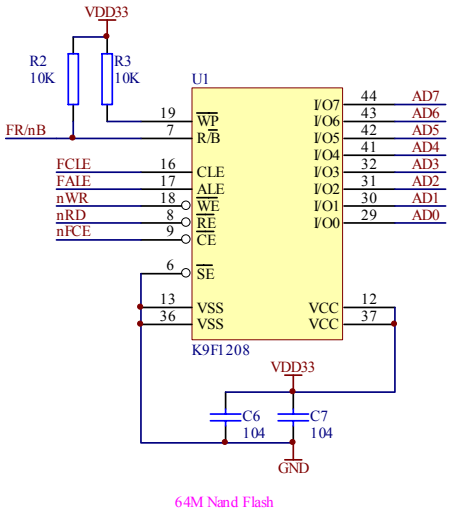


图 17　NAND Flash 芯片电路图

8）音频输出信号 MCLK、LRCLK、SCLK、SDATA 等。这些信号将数字音频信息送到立体声音频 DA 芯片 CS4330 中，再通过 TDA2822 放大后推动耳机。电路如图 18 所示。

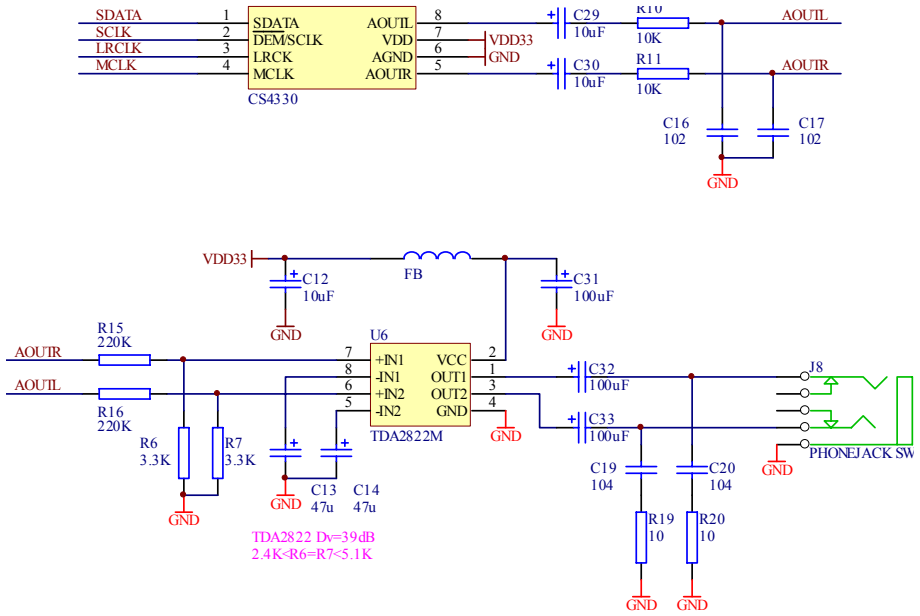


图 18　音频 DA 及功放电路

9）IO 口和总线扩展电路。学习板尚未使用的信号用两个 20 脚插座引出，可以在上面扩展其他电路。这部分信号包括总线 AD0~AD7，nWR 和 nRD，ALE，中断信号 INT0 和 INT1，IO 端口 P4.0~P4.7，P1.0~P1.7，其中 P1.0~P1.3 是和键盘 KIN0~KIN3 复用的。插座见图 19 所示。

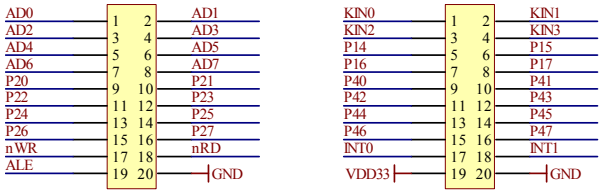


图 19　扩展总线和 IO 端口

10）LCD 液晶屏扩展模块。本学习板配套的 CM12232 图形点阵液晶屏是接在扩展插座上的，占用 P1.4~P1.7 和 P4 口。LCD 扩展接线如图 20 所示。关于 LCD 模块的资料参考光盘里的相关文档。

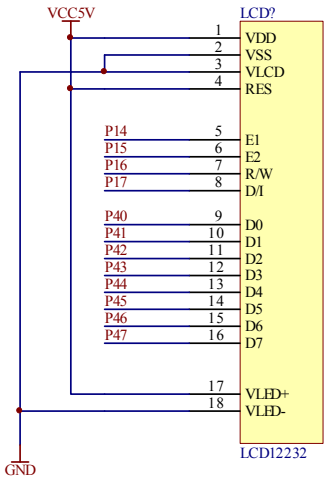


图 20　LCD 扩展模块接线图

11）电源电路，包括升压芯片 MAX856 和降压芯片 AS1117。电源电路如图 21 所示。

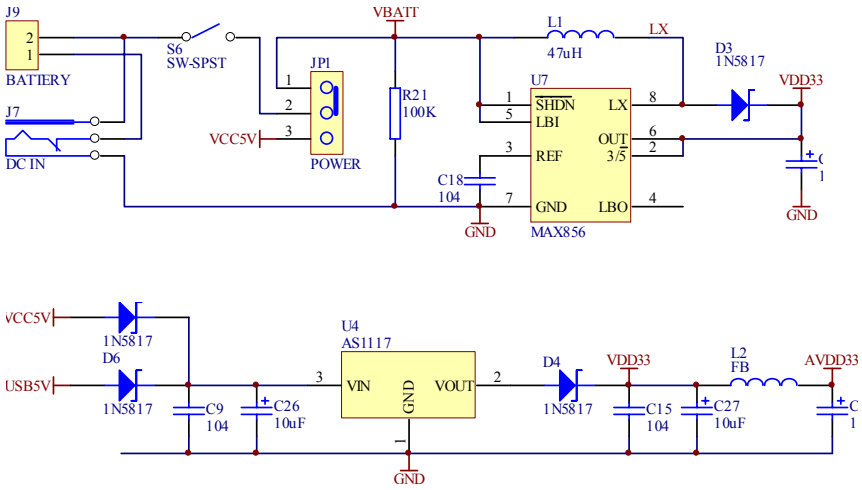


图 21　　电源电路

USB 的 5V 电压经过 AS1117-3.3 稳压芯片就可以作为板上电路的电源使用。另外，学习板还具有由 MAX856 构成的升压电路，支持电压低于 3.3V 的电池供电，但 AT89C51SND1C 和 CS4330 芯片本身的功耗并不适用于低功耗应用，所以无法苛求使用电池作为电源。所以，板上的电源输入配置有两种插座，在 PCB 上一个标有 DC IN，一个标有 BATTERY。同时，从这两个电源插座输入的电压如果大于 4.5V 的时候，通过跳线 JP1 选择到相应位置（PCB 丝印中有 JP1 跳线的说明），就可不经过升压电路，而直接送入 AS1117 降压到 3.3V 使用。但如果 JP1 跳线选择升压电路时，由于升压电路当输入电压大于 3.3V 后其输出不再受控制（将随输入电压而增加），所以此时输入电压严禁超过 3.3V，否则会烧毁其他负载芯片。电路中加入若干低压降二极管 1N5817 来分别防止 USB5V 和外接电源 VCC5V、升压器输出的 3.3V 和 AS1117 稳压器输出的 3.3V 之间的冲突。MAX856 升压电源输入端对地的 100K 电阻用来保证在没有外接电源的时候，升压芯片处于关断状态。

4. PCB 指导

学习板的 PCB 丝印说明见图 22，上面大致标出了主要元件和电路单元的分布。在光盘里也提供了 PCB 丝印的独立文件和 PCB 实物的照片，供焊接或对照参考之用。

光盘里也给出了电路原理图和 PCB 文件中的用到的元件库，用户可以自己绘制基于 AT89C51SND1 单片机的项目的原理图和 PCB，可以直接使用库中的元件或封装。但本学习板在发布的时候未能提供 Protel 格式的原理图和 PCB 图。

可以使用电烙铁、焊锡、助焊剂以及镊子等工具进行元件焊接。对于 0805 的阻容件可以先在右边焊盘点少许焊锡，左手用镊子夹住元件中间，右手用烙铁熔化锡点的同时左手将元件放到焊盘上，焊接右边焊盘同时用镊子定位元件，离开烙铁并等待该焊盘冷却后再焊接左边焊盘；而芯片则关键是定位，放好元件后左手压住芯片不动，右手焊接分开的 2 个以上焊盘，使芯片固定再整体焊接，注意控制焊锡用量，配合助焊剂使得焊盘上焊锡适中，如果出现焊锡粘连多个引脚的现象，可以用吸锡线吸掉多余的焊锡，切勿用力刮蹭芯片引脚，一旦将纤细的引脚碰歪后更难以分离粘连，甚至损坏芯片。焊接技巧因人而异，在此不多描述。

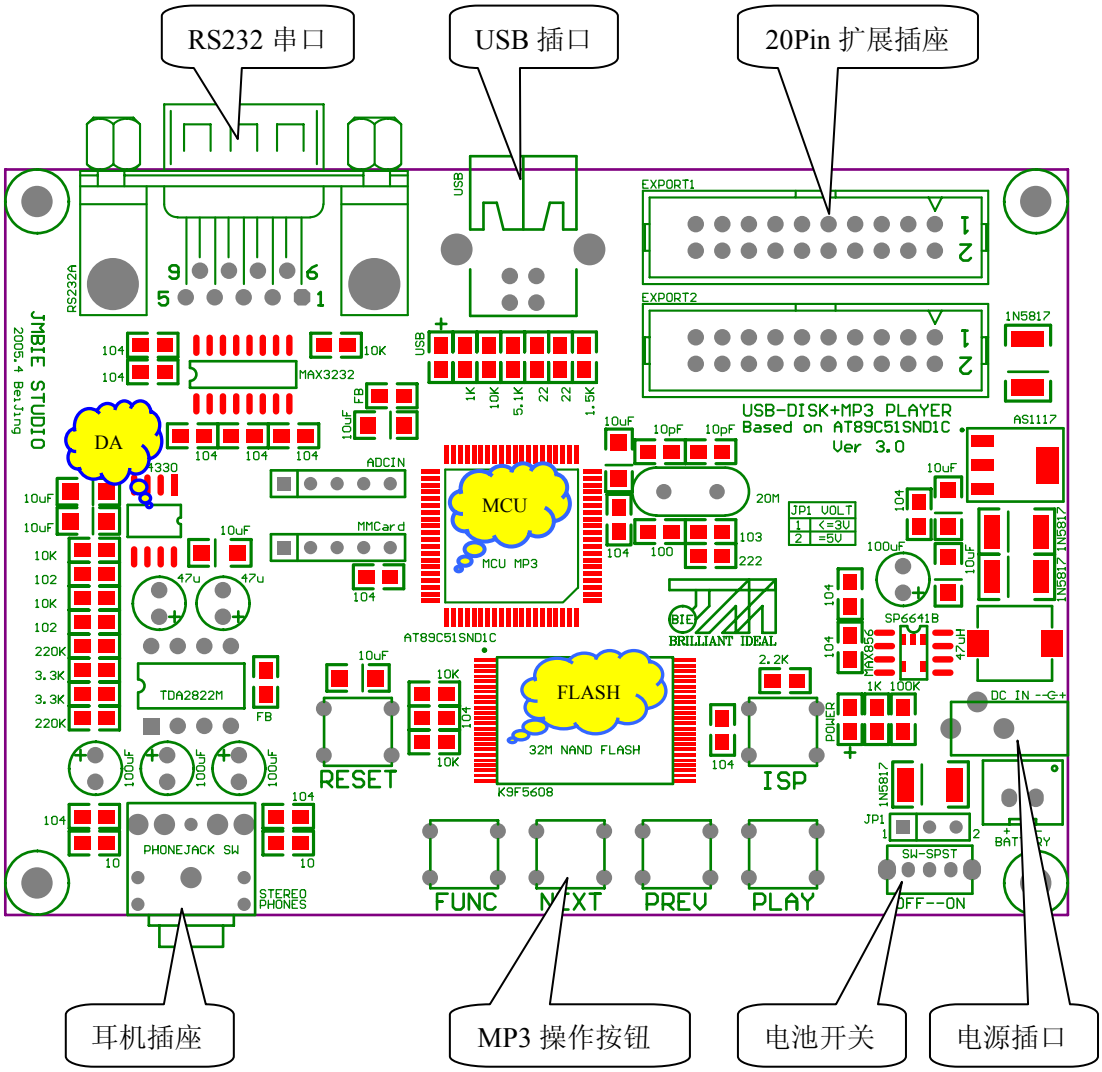


图 22　　学习板 PCB 说明

5. 连接和按键使用说明

学习板在使用时需要和 PC 机通信来下载固件，复制歌曲，或调试程序。图 23 给出了可能的线缆或附件的连接示意图。

学习板上的按键使用说明如下：

- **RESET** 按键给单片机提供硬件复位电平。必要时按此键是系统复位。
- **ISP** 按键用以执行 AT89C51SND1 的 BootLoader 程序，从而可以用 FLIP 软件下载固件。先按下此键，再插入 USB 电缆，稍等片刻 AT89C51SND1 就作为一个特定 USB 设备被 PC 机识别，正确安装驱动程序后，FLIP 软件即可与其通信。
- **PLAY** 按键是 MP3 的播放/暂停控制键，系统进入 MP3 播放功能后，按此键开始播放歌曲；播放过程中按此键暂停，再按此键继续播放，如此反复。
- **FUNC** 按键是 MP3 的功能选择键，决定 **NEXT** 和 **PREV** 两个按键的作用。默认状态下 **NEXT** 和 **PREV** 为下一曲和上一曲的换曲控制键；按一下 **FUNC** 键后，**NEXT** 和 **PREV** 则是调节音量的增减控制键；再按一下 **FUNC** 键后，**NEXT** 和 **PREV** 则是音调控制键；再按一下 **FUNC** 键后，返回默认状态。

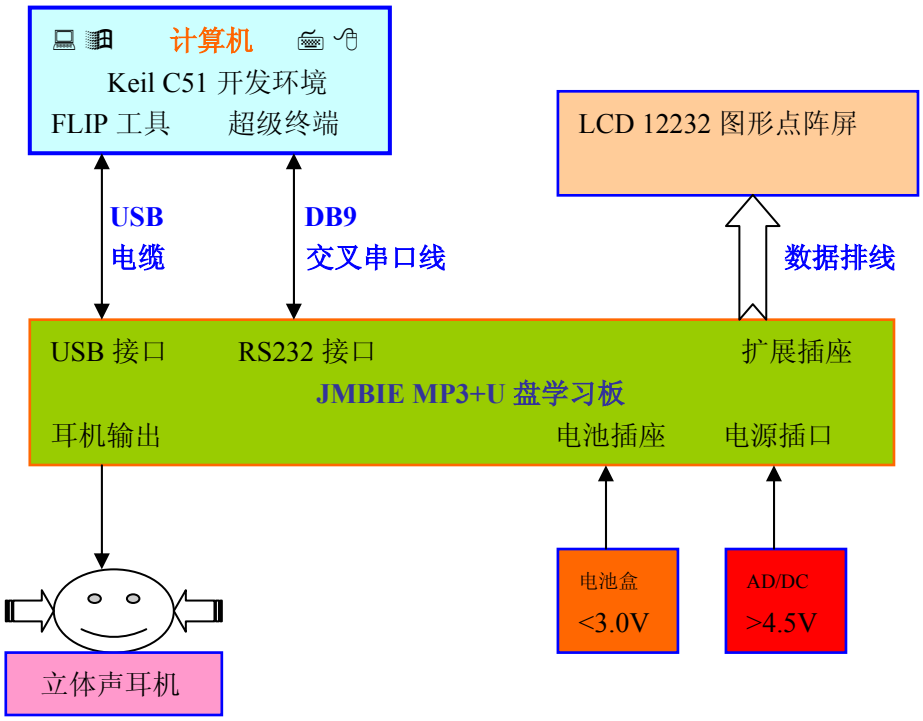


图 23　MP3＋U 盘学习板外部连接示意图

二. 软件概述

1. 软件方案及实现

根据系统功能的构成，软件可分为主控程序、MP3 播放程序、U 盘程序等主要结构，而提供支持的则有文件系统和 Flash 读写程序。还有用于交互的串口通讯、LCD 显示、键盘控制等程序。软件的构成如图 24 所示。

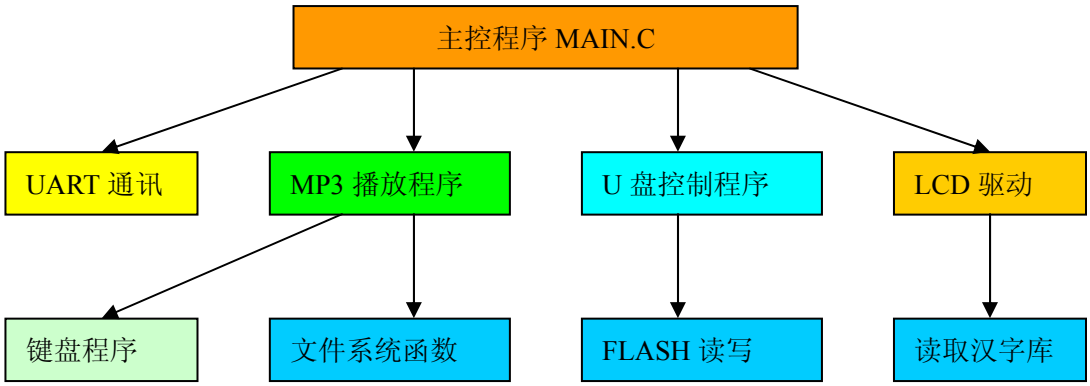


图 24 MP3 学习板软件构成

由于 MP3 播放只需要读取 MP3 文件，文件系统函数中也只提供读文件扇区功能。由于 LCD 部分的汉字库也是以文件形式保存在 Flash 中的，所以读取汉字库也是依赖于文件系统的。

为了保护作者的权益，图 24 中“文件系统函数”，“FLASH 读写”，“读取汉字库”等文件的源代码不再免费公开提供，而是以函数库的形式提供可用的库文件，用户可以直接使用。

本学习板的源代码中，图 24 里所示的各单元都对应一个 C 文件，并提供已经设置好的工程文件。用户可以在 Keil C51 集成环境下对该工程文件进行编译连接。所提供的源文件都是经过验证可用的，但这并不意味着它是完美的或者完全符合用户要求的，用户完全可以分析并提出问题，解决问题本身就是学习，这也是利用学习板进行开发的内容。

和其他单片机系统的开发过程一样，借助于 Keil C51 集成开发环境，按图 25 的思路调试程序：

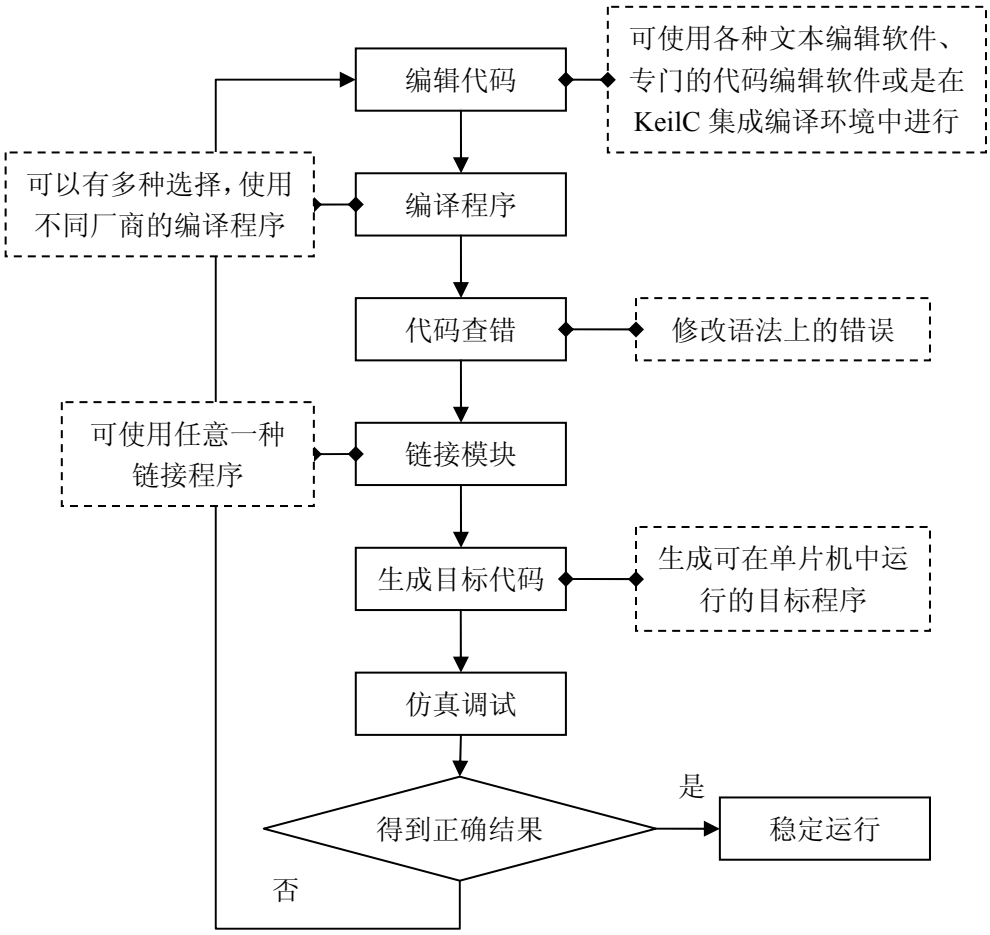


图 25　单片机系统开发流程

在本学习板的学习过程中，可以使用 Keil C51 集成开发环境完成源代码编辑、编译和连接等全部工作。光盘里提供了 Keil C51 的安装版，所在目录是“开发软件”下的“KeilC51v750a”，安装后产生的程序组为 Keil uVision2。利用集成开发环境进行编程的时候，需要建立一个 Keil C51 工程文件，这点和其他集成开发环境类似。本学习板的 Keil C51 工程文件是 JmbieC51MP3.uv2，可在软件中打开该文件。在工程文件中主要包括针对该工程的设置选项和工程树管理等信息，并不包括源程序，真正的源代码是那些.C 和.H 文件。但工程文件中包含着必须的编译和连接设置，和单片机的硬件属性息息相关，在 MP3 学习板的工程文件中，比较关键的设置如图 26，图 27，图 28，图 29 所示。

选取工程根部，右键菜单里查看“Options for Target ‘JMBIE MP3’”，出现的对话框里，Device 选项中选择 AT89C51SND1；Output 选项中，选择 Create Executable 并选中 Create HEX File，以能产生 16 进制可执行文件。Target 选项中 Xtal 填写 20.0（晶振 20MHz），Memory Model 选 Large：variables in XDATA，Code Rom Size 选 Compact：2K functions，64K Program；同时选择 Use on-chip XRAM 复选框。

选中树形列表中的 FLASH_RW.C 文件，右键菜单中查看“Options for File ‘FLASH_RW.C’”，必须选择 Generate Assembler SRC File 和 Assemble SRC File 两个复选框。对于包含汇编代码的 C 文件，选择这两个选项可以顺利通过编译；而纯 C 文件也可以选择这两个选项，能产生对应的反汇编.SRC 文件，可以方便的对比 C 源码和汇编代码。

在 64M 的版本中，含汇编的 FLASH_RW.C 已经包含在库里，对其设置可以忽略。在此仅作此类问题的设置说明。

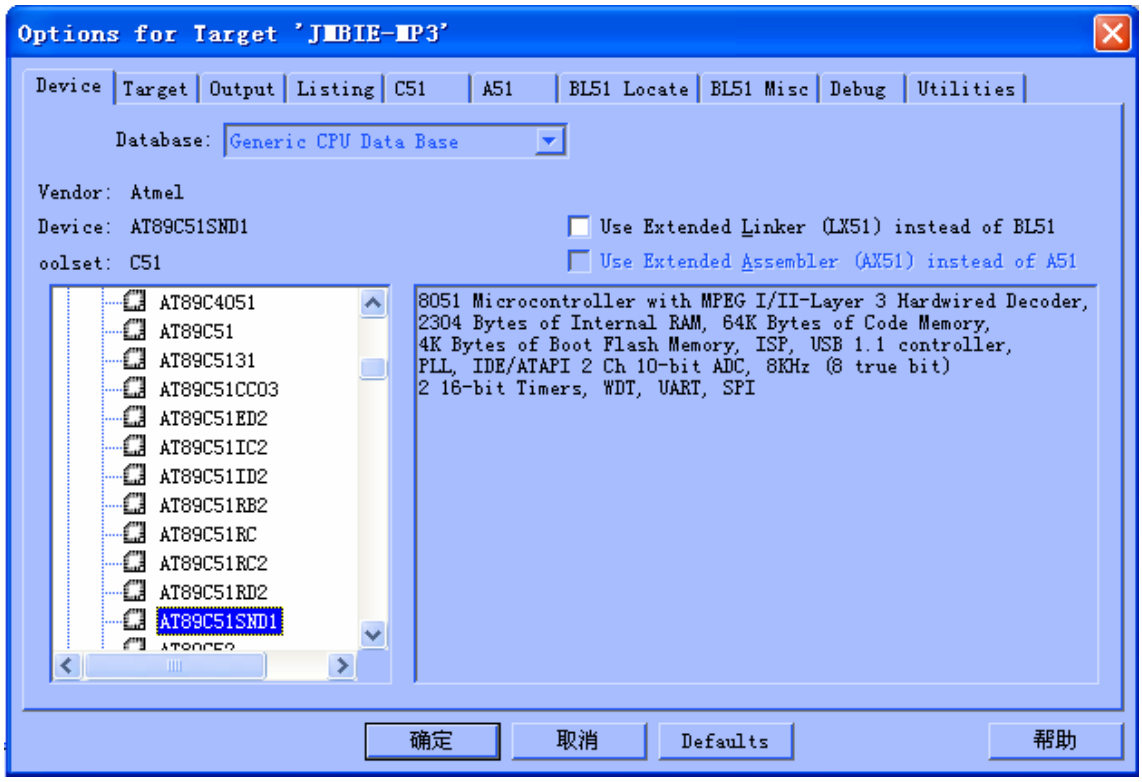


图 26　选择单片机型号

图 27　对晶振和存储模式进行设置

图 28　设置生成目标代码

图 29　设置嵌入汇编的 C 文件编译选项

用户可以练习新建工程文件，按上述方法进行正确设置，将必要的 C 文件和 H 文件添加到工程中。选取工程树根部，右键菜单里查看“Manage Components”，出现的对话框如图 30

所示。可以添加工程目标，组，并在该组添加 C 文件。

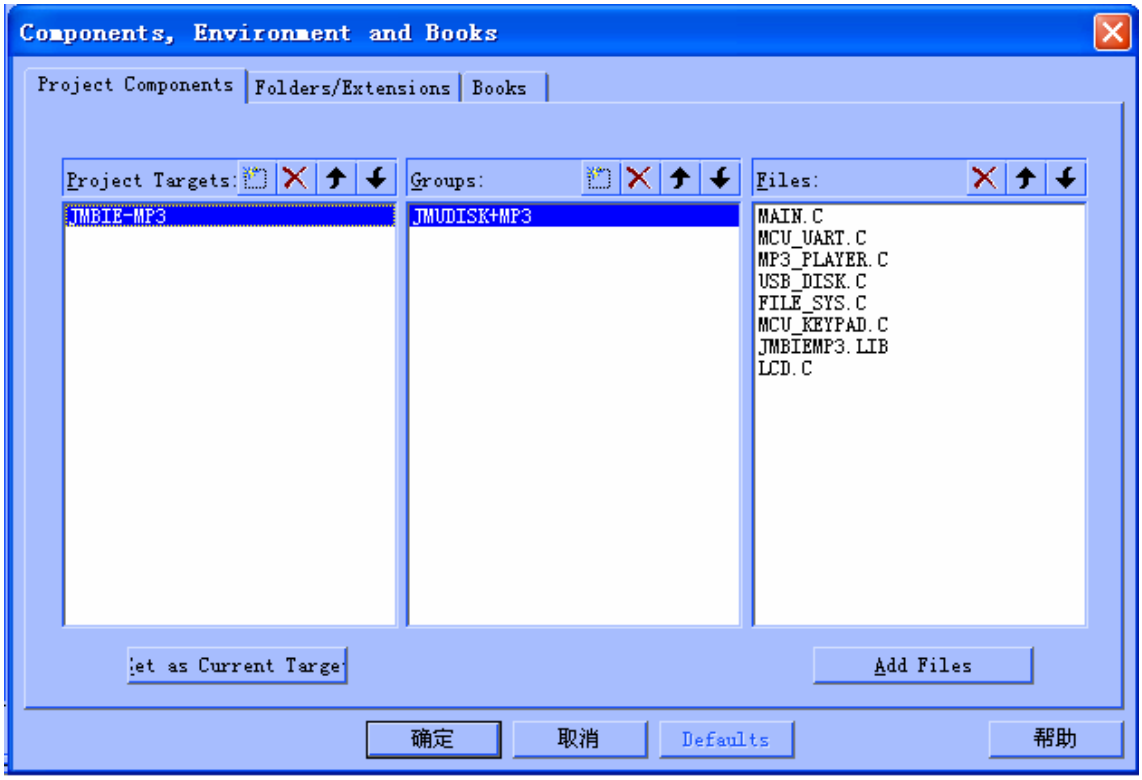


图 30　在工程里添加 C 文件

打开 JmbieC51MP3 工程后会看到图 31 所示的 Keil C51 界面，一般情况下点“Rebuild all target files”按钮即可自动完成编译连接过程。

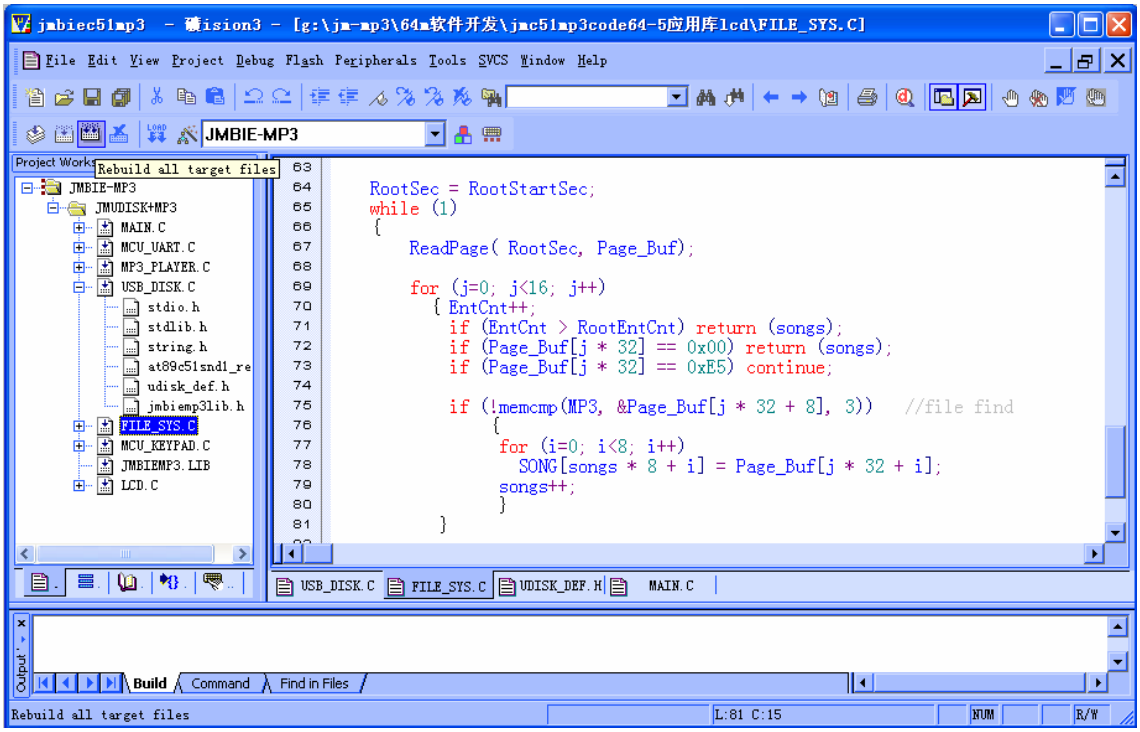


图 31　打开工程文件后的界面

对于 Keil C51 软件的使用，在这里不在进行更详细的说明，用户可以专门阅读相关书籍，了解关于该软件的菜单、按钮、设置等信息，以及 C51 语言的语法、表达式、和硬件相关的定义和约束等，因为 C51 语言来源于 C 语言，和 C 语言很相似但毕竟编程目标是硬件资源有限的单片机，所以还是和 PC 机上的 C 语言编程有所差别。详细介绍这些问题的书籍在书店可以买到多种版本。当然，Keil C51 集成开发环境和其他集成开发环境很相似，摸索一段时间就会熟练使用的。

2. 主控程序代码分析

主控程序的任务是，检测 USB_INS（P3.4）的电平来决定进入 MP3 播放过程还是执行 U 盘功能。这种检测只在上电复位时进行，所以播放过程中无法进入 U 盘。无论是 MP3 播放程序还是 U 盘程序都需要一系列初始化工作，之后开始循环进行对应功能的执行。主控程序的流程图如图 32 所示。

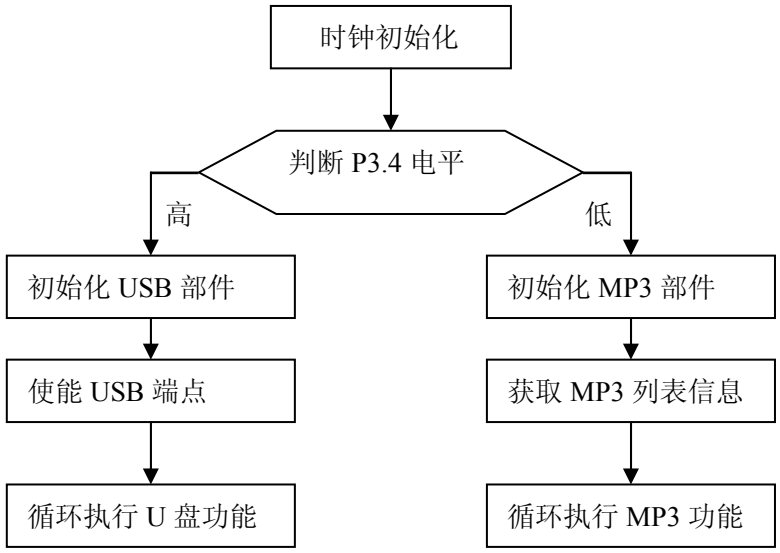


图 32　主控程序流程图

主控程序的主要代码如下：

```
CKCON|=X2;   //X2 Mode
Flash_Reset();
if(USB_INS)      //USB Cable Insert Sensor
  {//USB Disk    //如果 P3.4 为高电平则执行 U 盘功能
     AtmelUSBInit();
     EpEnable();
     while(1)
      {  if (UEPINT & EP0)   Ep0();
         if (UEPINT & EP1)   main_txdone();  //UEPINT 地址 0xF8 却不能位寻址
         if (UEPINT & EP2)   main_rxdone(); //所以这里不能用位定义 EPxINT
      }
  }
 else
  {//MP3 Player  //如果 P3.4 为低电平则执行 MP3 播放功能
     MP3_Init();
     KeyBoardInit();
```

```
        MP3InitFlag = 1;
        NowPlaying=0;
        Init_FAT_Info();
        NumofSong = GetMP3List();   //获取 MP3 歌曲列表信息
        for(i=0;i<8;i++)
                PlayingSong[i]= SONG[8 * NowPlaying + i];
       while(1)
       {if(NumofSong != 0)
            {if(MP3InitFlag)   //MP3 歌曲开始播放时需要初始化该歌曲的信息
                {if ( ! PlayInit(PlayingSong) )
                       {printu("Bad MP3 file or error!\n");
                        while(1);  //读取失败
                       }
                  MP3InitFlag = 0;
                }
             PlayMP3(PlayingSong); //初始化歌曲信息后进入播放过程
            }  //直到该歌曲播放完毕返回
       }
    }
```

3. U 盘相关代码分析

在主控程序中就可以看到 U 盘功能最开始先用 AtmelUSBInit()和 EpEnable()函数来初始化单片机 AT89C51SND1 的 USB 部件，之后便循环查询各端点并作出相应处理。

实际上，关于 USB 海量存储（USB Mass Storage）的知识涉及到 USB 通信原理、设备描述符、Bulk-Only 协议、SCSI 命令集等，相当复杂。在这里我们只给出简要的说明和图表。更详细的内容请参考专业书籍或者光盘里的英文文档。

预备知识

@USB 协议规定，所有的 USB 设备都必须支持以下通用的操作集：

A.动态插接与拔开

USB 设备必须在任意时刻允许被插入与拆除。提供连接点或端口的集线器应当负责汇报端口的状态改变情况。

当主机探测到连接操作后，会使得所连的集线器端口生效，设备也会因此而复位，一个被复位了的 USB 设备有如下特性：

- 对缺省 USB 地址发生响应
- 没有被配置
- 初始状态不是挂起

当设备从一个集线器端点拆除时，集线器会使得原来连接的端口失效，并且通知主机设备已移去。

B.地址分配

当 USB 设备连接以后，主机通过缺省地址对设备进行管理，给此设备分配一个唯一的地址，这个操作是在设备复位及端点使能操作以后。

C.配置

USB 设备在正常被使用以前，必须被配置，由主机负责配置设备。主机一般会从 USB 设备获取配置信息后再确定此设备有哪些功能。作为配置操作的一部分，主机会设置设备的配置值，并且，如果必要的话会选择合适的接口的备选设置。在单个配置操作内，一个设备可能支持多重接口。一个接口是一组端结点集合，它们代表了设备向主机提供的单一的功能或特性，用来与这组相关端结点通信的协议以及接口内各端结点的用途，都可以作为一个设备类的一部分或者由厂商制定具体定义。另外，一个配置中的接口可能有备选设置。这些备选设置会重定义相关端结点的数目或特性。如果是这样的话，设备必须支持 Get Interface(获取接口)与 Set Interface(设置接口)请求，来提交及选择指定的接口的设备选设置。在每个设备配置下，每个接口描述表可能包括用来标识接口的及备选设置的域，接口被从 0～N-1 编号。n 为配置所支持的能同时使用的接口数目，类似的设置的编号也从 0 开始。当设备初始化配置后，缺省设置是备选设置 0。为了支持通用的设备驱动程序管理一组相关的 BUS 设备，设备与接口描述表中分组含了类(Class)，子类(Sub class)，及协议(Protocol)域。这些域用来标识一个设备的功能及用于通信的协议。

D.数据传送

数据可能以四种方式在 USB 设备端结点与主机之间传送。在不同设置下，一个终端结点可能被用于不同的传输方式，但一旦设置选定，传送方式就选定了。

@常见 USB 设备请求

设备请求是一个标准请求集，但在实际的某个应用中，往往只用到其中的几个请求，为了更加清晰，现将常见的几个请求的数据格式及其含义罗列如下：

（1）Get Descriptor

USB 设备一插入到 USB 总线后，主机为了了解设备的情况，即使用此设备请求，来获取设备信息。

bmRequestType(1)	bRequest(1)	wValue(2)	Windex(2)	wLength(2)	Data
0x80	0x06	00 01	00 00	08 00	

其中 bmRequestType ＝0x80，bRequest＝06，代表请求类型为 Get Descriptor，而描述符类型由 wValue＝00 01 来表达，表示 DEVICE，即获取设备描述符，长度由 wLength＝08 给出。设备接收到此请求后，应该向主机返回本设备的设备描述符，长度为 8 个字节。描述符的格式见下节所述。

bmRequestType(1)	bRequest(1)	wValue(2)	Windex(2)	wLength(2)	Data
0x80	0x06	00 02	00 00	0xff 00	

除 wValue＝00 02，wLength=0xff 外，其他内容与前相同，表示读取全部配置描述符,其中 wValue＝00 02 表示 CONFIGURATION。

（2）Set Configuration

得到描述符以后，主机将为设备选择一个配置，成功进行配置以后，描述符中所定义的各个端点便可以正常使用了。

bmRequestType(1)	bRequest(1)	wValue(2)	Windex(2)	wLength(2)	Data
0x00	0x09	01 00	00 00	00 00	

bmRequestType ＝0，bRequest＝0x09，代表请求类型为 SET_CONFIGURATION。

（3）Set Address

设置地址，主机在通过缺省管道读取设备描述符后，便对设备设置一个非 0 地址，此后的通信在此地址进行。

bmRequestType(1)	bRequest(1)	wValue(2)	Windex(2)	wLength(2)	Data
0x00	0x05	00 02	00 00	00 00	

bmRequestType ＝0，bRequest＝0x05，代表请求类型为 SET_ADDRESS。

@USB海量存储概述

USB 组织定义了海量存储设备类（Mass Storage Class）的规范，这个类规范包括四个独立的子类规范，即：

- USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
- USB Mass Storage Class Bulk-Only Transport
- USB Mass Storage Class ATA Command Block
- USB Mass Storage Class UFI Command Specification

前两个子规范定义了数据/命令/状态在USB 上的传输方法。CBI 传输规范使用Control/Bulk/Interrupt 三种类型的端点进行数据/命令/状态传送，而Bulk- Only 传输规范仅仅使用Bulk 端点传送数据/命令/状态。后两个子规范则定义了存储介质的操作命令。ATA 命令规范用于硬盘，UFI 命令规范是针对USB 移动存储。

Microsoft Windows 中提供对Mass Storage 协议的支持，因此USB 移动设备只需要遵循Mass Storage 协议来组织数据和处理命令，即可实现与PC 机交换数据。而Flash 的存储单元组织形式采用FAT16 文件系统，这样，就可以直接在Windows 的浏览器中通过可移动磁盘来交换数据了，Windows 负责对FAT16 文件系统的管理，USB 设备不需要干预FAT16 文件系统操作的具体细节。

端点是USB 中一个独特的概念，它是一个可以与USB Host 交换数据的硬件单元。USB Host 与USB 设备之间都是通过端点来传输数据的，端点是桥梁和纽带，不同的端点其传输数据的能力不同，适于不同的应用场合。来自主机的通信流将止于设备上的端点，而从设备上向主机提供的数据也起于端点。管道是设备上的一个端点和主机上的软件的联合体。它表示经过一个存储缓冲区和一个设备上的端点，可以在主机上的软件之间传送数据的能力。与0号端点对应的0号管道总是可用的，主机通过这个管道来向设备查询信息，进行初始化，并配置其他端点。其他端点一经配置，与之对应的管道便建立起来了。系统软件使用缺省管道（与端点0相对应）来管理USB设备。

每个设备可以有一个或多个配置（Configuration），配置用于定义设备的功能。如果某个设备有几种不同的功能，则每个功能都需要一个配置。配置（*configuration*）是接口（*interface*）的集合。接口指定设备中的哪些硬件与USB 交换数据。每一个与USB 交换数据的硬件就叫做一个端点（*endpoint*）。因此，接口是端点的集合。USB 的设备类别定义（USB Device Class Definitions）定义特定类或子类中的设备需要提供的缺省配置、接口和端点。

USB海量存储设备（USB Mass Storage Class）包括General Mass Storage Subclass、CD－ROM、Tape、Solid State四个子类。Mass Storage Class只需要支持一个接口，即数据（Data）接口，选择缺省配置时此接口即被激活。数据接口允许与设备之间进行数据传输，它提供三个端点：Bulk Input端点、Bulk Output端点和中断端点。

通用海量存储设备（General Mass Storage Device）是随机存取、基于块 / 扇区存储的设备。它只能存储和取回来自 CPU 的数据。**这种设备的接口遵循 SCSI-2 标准的直接存取存储设备（Direct Access Storage Device）协议。**USB 设置上的介质使用与 SCSI-2 设备相同的逻辑块（Logical Block Address）方式寻址。

描述符（*descriptor*）描述设备、配置、接口或端点的一般信息。USB（Host）唯一通过描述符了解设备的有关信息，根据这些信息，建立起通信，在这些描述符中，规定了设备所使

用的协议、端点情况等。因此，正确地提供描述符，是USB设备正常工作的先决条件。每个USB 设备都必须有一个设备描述符。

@描述符

(1). 设备描述符**Device Descriptor**

Offset	Field	Size	Value	Description
0	*bLength*	Byte	12h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	01h	DEVICE descriptor type.
2	*bcdUSB*	Word	????h	*USB Specification* Release Number in Binary-Coded Decimal (i.e. 2.10 = 210h). This field identifies the release of the *USB Specification* with which the device and is descriptors are compliant.
4	*bDeviceClass*	Byte	00h	**Class is specified in the interface descriptor.**
5	*bDeviceSubClass*	Byte	00h	**Subclass is specified in the interface descriptor.**
6	*bDeviceProtocol*	Byte	00h	**Protocol is specified in the interface descriptor.**
7	*bMaxPacketSize0*	Byte	??h	Maximum packet size for endpoint zero. (only 8, 16, 32, or 64 are valid **(08h, 10h, 20h, 40h)**).
8	*idVendor*	Word	????h	Vendor ID (assigned by the USB-IF).
10	*idProduct*	Word	????h	Product ID (assigned by the manufacturer).
12	*bcdDevice*	Word	????h	Device release number in binary-coded decimal.
14	*iManufacturer*	Byte	??h	Index of string descriptor describing the manufacturer.
15	*iProduct*	Byte	??h	Index of string descriptor describing this product.
16	*iSerialNumber*	Byte	??h	Index of string descriptor describing the device's serial number. **(Details in 4.1.1 below)**
17	*bNumConfigurations*	Byte	??h	Number of possible configurations.

(2). 配置描述符**Configuration Descriptor**

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	02h	CONFIGURATION Descriptor Type.
2	*wTotalLength*	Word	????h	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	Byte	??h	Number of interfaces supported by this configuration. **The device shall support at least the Bulk-Only Data Interface.**
5	*bConfigurationValue*	Byte	??h	Value to use as an argument to the *SetConfiguration()* request to select this configuration.
6	*iConfiguration*	Byte	??h	Index of string descriptor describing this configuration.
7	*bmAttributes*	Byte	?0h	Configuration characteristics: **Bit** **Description** 7 Reserved (set to one) 6 Self-powered 5 Remote Wakeup 4..0 Reserved (reset to zero) **Bit** 7 is reserved and must be set to one for historical reasons. **For a full description of this *bmAttributes* bitmap, see the *USB 1.1 Specification*.**
8	*MaxPower*	Byte	??h	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e. 50 = 100mA)

(3). 接口描述符**Bulk-Only Data Interface Descriptor**

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	04h	INTERFACE Descriptor Type.
2	*bInterfaceNumber*	Byte	0?h	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	Byte	??h	Value used to select alternate setting for the interface identified in the prior field.
4	*bNumEndpoints*	Byte	??h	Number of endpoints used by this interface (excluding endpoint zero). **This value shall be at least 2.**
5	*bInterfaceClass*	Byte	08h	**MASS STORAGE Class.**
6	*bInterfaceSubClass*	Byte	0?h	Subclass code (assigned by the USB-IF). **Indicates which industry standard command block definition to use. Does not specify a type of storage device such as a floppy disk or CD-ROM drive. (See *USB Mass Storage Overview Specification*)**
7	*bInterfaceProtocol*	Byte	50h	**BULK-ONLY TRANSPORT. (See *USB Mass Storage Overview Specification*)**
8	*iInterface*	Byte	??h	Index to string descriptor describing this interface.

应至少支持一个接口，这里为Bulk-Only Data接口，此接口使用三个端点：控制端点（默认）、Bulk-In端点和Bulk-Out。其中***bInterfaceSubClass***指定所使用的工业标准命令块，这里应该是SCSI-2命令集；***bInterfaceProtocol***为所使用的传输协议，指定为BULK-ONLY。相关的工业标准命令块和传输协议见下页表格。

(4). 子类代码对应的命令块**Command Block Specifications**

SubClass Code	Command Block Specification	Comment
01h	Reduced Block Commands (RBC) T10 Project 1240-D	Typically, a Flash device uses RBC command blocks. However, any Mass Storage device can use RBC command blocks.
02h	SFF-8020i, MMC-2 (ATAPI)	Typically, a C/DVD device uses SFF-8020i or MMC-2 command blocks for its Mass Storage interface.
03h	QIC-157	Typically, a tape device uses QIC-157 command blocks.
04h	UFI	Typically a floppy disk drive (FDD) device
05h	SFF-8070i	Typically, a floppy disk drive (FDD) device uses SFF-8070i command blocks. However, an FDD device can be in another subclass (for example, RBC) and other types of storage devices can belong to the SFF-8070i subclass.
06h	SCSI transparent command set	
07h – FFh	Reserved for future use.	

(5). 传输协议**Mass Storage Transport Protocol**

bInterfaceProtocol	Protocol Implementation	Comment
00h	Control/Bulk/Interrupt protocol (with command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
01h	Control/Bulk/Interrupt protocol (with no command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
50h	Bulk-Only Transport	USB Mass Storage Class Bulk-Only Transport
02h – 4Fh	Reserved	
51h – FFh	Reserved	

(6). 端点描述符**Bulk-In Endpoint Descriptor**

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT Descriptor Type.
2	*bEndpointAddress*	Byte	8?h	The address of this endpoint on the USB device. The address is encoded as follows. **Bit** **Description** 3..0 The endpoint number 6..4 Reserved, set to 0 7 1 = In
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

(7). 端点描述符**Bulk-Out Endpoint Descriptor**

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	Byte	0?h	The address of this endpoint on the USB device. This address is encoded as follows: **Bit** **Description** 3..0 Endpoint number 6..4 Reserved, set to 0 7 0 = Out
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, or 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

由于控制端点为每个设备都使用的缺省端点，因此不需要定义，只需定义Bulk-In 和Bulk-Out 两个端点。

@Bulk－Only 传输协议

设备插入到USB 后，USB 即对设备进行搜索，并要求设备提供相应的描述符。在USB Host得到上述描述符后，即完成了设备的配置，识别出为Bulk－Only 的Mass Storage 设备，然后即进入Bulk－Only 传输方式。在此方式下，USB 与设备间的所有数据均通过Bulk-In和Bulk-Out 来进行传输，不再通过控制端点传输任何数据。

在这种传输方式下，有三种类型的数据在USB 和设备之间传送，CBW、CSW 和普通数据。CBW（Command Block Wrapper，即命令块包）是从USB Host 发送到设备的命令，命令格式遵从接口中的bInterfaceSubClass 所指定的命令块，这里为SCSI 传输命令集。USB设备需要将SCSI 命令从CBW 中提取出来，执行相应的命令，完成以后，向Host 发出反映当前命令执行状态的CSW（Command Status Wrapper），Host 根据CSW 来决定是否继续发送下一个CBW 或是数据。Host 要求USB 设备执行的命令可能为发送数据，则此时需要将特定数据传送出去，完毕后发出CSW，以使Host 进行下一步的操作。

（1）. **Command Block Wrapper**

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

其中 dCBWSignature 的值为 43425355h（LSB），表示当前发送的是一个 CBW；dCBWTag 的内容需要原样作为 dCSWTag 再发送给 Host；dCBWDataTransferLength 为本次 CBW 需要传输的数据，bmCBWFlags 反映数据传输的方向，0 表示来自 Host，1 表示发至 Host；bCBWLUN 一般为零，但当设备有多个逻辑单元时，用此位指定本次命令是发给谁的；bCBWCBLength 为本次命令字的长度；CBWCB 即为真正的传输命令集的命令。

得到一个 CBW 后，解析出 CBWCB 中所代表的命令，然后按照 SCSI 命令集中的定义来执行相应的操作，或是需要接收下一个 Bulk－Out 发来的数据，或是需要向 Host 传送数据，完成以后需要向 USB Host 发送 CSW，反映命令执行的状态。USB 也是通过此来了解设备的工作情况的。

（2）. **Command Status Wrapper**

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

dCSWSignature 的内容为 53425355h，dCSWTag 即为 dCBWTag 的内容，dCSWDataResidue 还需要传送的数据，此数据根据 dCBWDataTransferLength－本次已经传送的数据得到。Host 端根据此值决定下一次 CBW 的内容，如果没有完成则继续；如果命令正确执行，bCSWStatus 返回 0 即可。按这个规则组装好 CSW 后，通过 Bulk-In 端点将其发出即可。

◎SCSI-2 命令集

Bulk-Only 的 CBW 中的 CBWCB 中的内容即为如下格式的命令块描述符（Command Block Descriptor）。SCSI-2 有三种字长的命令，6 位、10 位和 12 位。

（1）. SCSI-2命令描述块**command descriptor block（12 Bytes）**

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code							
1	Logical unit number			Reserved				
2	(MSB)							
3				Logical block address (if required)				
4								
5								(LSB)
6	(MSB)							
7				Transfer length (if required) Parameter list length (if required) Allocation length (if required)				
8								
9								(LSB)
10	Reserved							
11	Control							

Operation Code 是操作代码，表示特定的命令。高3位为Group Code，共有8种组合，低5 五位为Command Code，可以有32 种命令。这样最多可以存在256个命令。后面还会给出几个常见命令的具体格式。

（2）. SCSI-2 命令集 Direct Access Storage Device

SCSI – Direct Access	Opcode
INQUIRY	12h
TEST UNIT READY	00h
FORMAT UNIT	04h
LOCK-UNLOCK CACHE	36h
MODE SELECT(6) MODE SENSE(6)	15h 1Ah
PRE-FETCH	34h
PREVENT-ALLOW MEDIUM REMOVAL	1Eh
READ(6) READ(10)	08h 28h
READ CAPACITY	25h
READ DEFECT DATA	37h
READ LONG WRITE LONG	3Eh 3Fh
REASSIGN BLOCKS	07h
RECEIVE DIAGNOSTIC RESULTS SEND DIAGNOSTIC	1Ch 1Dh
RELEASE	17h
RESERVE	16h
REZERO UNIT	01h
SEEK(10)	2Bh

SET LIMITS	33h
START STOP UNIT	1Bh
SYNCHRONIZE CACHE	35h
VERIFY	2Fh
WRITE(6) WRITE(10)	0Ah 2Ah
WRITE AND VERIFY	2Eh
WRITE SAME	41h

对于不同的命令，其命令块描述符略有不同，其要求的返回内容也有所不同，根据相应的文档，可以对每种请求作出适当的回应。比如，下面是 INQUIRY 请求的命令块描述符和其返回内容的数据格式：

（3）. INQUIRY 命令描述块符

Bit Byte	7	6	5	4	3	2	1	0
0	Operation code (12h)							
1	Logical unit number			Reserved				EVPD
2	Page code							
3	Reserved							
4	Allocation length							
5	Control							

（4）. INQUIRY 命令返回的数据格式

Bit Byte	7	6	5	4	3	2	1	0
0	Peripheral qualifier			Peripheral device type				
1	RMB	Device-type modifier						
2	ISO version		ECMA version			ANSI-approved version		
3	AENC	TrmIOP	Reserved			Response data format		
4	Additional length (n-4)							
5	Reserved							
6	Reserved							
7	RelAdr	WBus32	WBus16	Sync	Linked	Reserved	CmdQue	SftRe
8 15	(MSB) Vendor identification (LSB)							
16 31	(MSB) Product identification (LSB)							
32 35	(MSB) Product revision level (LSB)							

（5）. READ CAPACITY 命令

Bit Byte	7	6	5	4	3	2	1	0
0	Operation code (25h)							
1	Logical unit number			Reserved				RelAdr
2	(MSB)							
3				Logical block address				
4								
5								(LSB)
6	Reserved							
7	Reserved							
8	Reserved							PMI
9	Control							

（6）. READ CAPACITY 命令返回数据格式

Bit Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				Returned logical block address				(LSB)
4	(MSB)							
7				Block length In bytes				(LSB)

（7）. MODE SENCE 命令

Bit Byte	7	6	5	4	3	2	1	0
0	Operation code (1Ah)							
1	Logical unit number			Reserved	DBD	Reserved		
2	PC		Page code					
3	Reserved							
4	Allocation length							
5	Control							

（7）. MODE SENCE 命令返回数据格式

Bit Byte	7	6	5	4	3	2	1	0
0 - n	Mode parameter header							
0 - n	Block descriptor(s)							
0 - n	Page(s)							

依据MODE SENCE 命令中不同的**Page code**返回不同的Page。

下表是MODE SENCE 命令返回数据中**Mode parameter header**的格式：

Bit / Byte	7	6	5	4	3	2	1	0
0	Mode data length							
1	Medium type							
2	Device-specific parameter							
3	Block descriptor length							

下表是MODE SENCE 命令返回数据中**Mode parameter block descriptor**的格式：

Bit / Byte	7	6	5	4	3	2	1	0
0	Density code							
1	(MSB)							
2	Number of blocks							
3								(LSB)
4	Reserved							
5	(MSB)							
6	Block length							
7								(LSB)

在SISC-2文档中有**direct-access**设备的**Page code**列表，但在MP3的开发中只需要响应PageCode＝0x3f（Return all pages）的情况，实际的代码中并没有关心PageCode，返回的数据中只包括**parameter header**和**block descriptor**两部分，没有返回任何Page。**parameter header**中的**Medium type** 和**Device specific parameter**均填0x00，含义请查看SISC文档**direct-access**设备的**Mode parameters**部分。

（5）. READ(10) 命令

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (28h)							
1	Logical unit number			DPO	FUA	Reserved		RelAdr
2	(MSB)							
3	Logical block address							
4								
5								(LSB)
6	Reserved							
7	(MSB)							
8	Transfer length							(LSB)
9	Control							

Host 会依次发出 INQUIRY、Read Capacity、Mode Sense 请求，如果上述请求的返回结果都正确，则 Host 会发出 READ 命令，读取文件系统 0 簇 0 扇区的 MBR 数据，进入文件系统识别阶段。

SCSI-2 命令在此无法全部列举，也不能作过多的分析，有兴趣的读者可以从光盘中提供的英文文档里获取有用的信息。除了上述的知识之外还有 USB 标准请求和 AT89C51SND1 单片机的 USB 部件配置等等内容，在光盘的文档里都能找到相关内容。

回到 JMBIE MP3+U 盘

对于 U 盘固件程序设计而言，难点在于充分了解 USB 海量存储的工作过程和每个阶段所用到的协议和详细定义。说的通俗些，U 盘的工作过程描述如下：插入 U 盘后，USB 主机发出 USB 标准请求，也就是配置阶段，这时 USB 设备需要返回正确的设备描述符等，主机可以得知设备的相关信息并对其进行配置，U 盘被主机识别为 Bulk-Only 的 Mass Storage 设备，然后即进入 Bulk-Only 传输方式，USB 设备需要解析 CBW 中的 SCSI 命令并执行对应功能，首先需要正确响应 INQUIRY、Read Capacity、Mode Sense 等请求，使主机获得关于 U 盘更具体的信息，之后便是 U 盘读写操作了，程序需要将 USB 传下来的数据写入 FLASH 芯片，在此并不用关系 FAT 文件系统，U 盘程序只需要将数据写入 Windows 指定的地址，包括 FAT 本身。

从主控程序可以看出，U 盘程序的流程图如图 33 所示。

对于一个 USB Mass Storage 设备而言，必须支持三个端点的数据处理。0 号端点：控制端点，用于控制传输，主机通过与端点 0 相对应的管道来读取设备描述符，完成对设备地址的设置，并完成配置。此端点为双向数据传输端点。两个非 0 端点：批量传输端点。这种端点为单向数据传输端点，分别为 Bulk-In 端点和 Bulk-Out 端点。AT89C51SND1 提供 Mass Storage 设备工作在批量传输时所需要的三个端点。最开始 USB 标准请求是通过端点 0 发给 USB 设备的。

首先是对 AT89C51SND1 单片机的 USB 硬件进行设置，包括时钟设置和使能 USB 部件。程序中用函数 AtmelUSBInit()完成该初始化过程，请参考 AT89C51SND1 的文档。然后用函数 EpEnable()设置两个批量传输端点，对 FIFO 缓冲区进行复位操作，允许相应端点的中断。

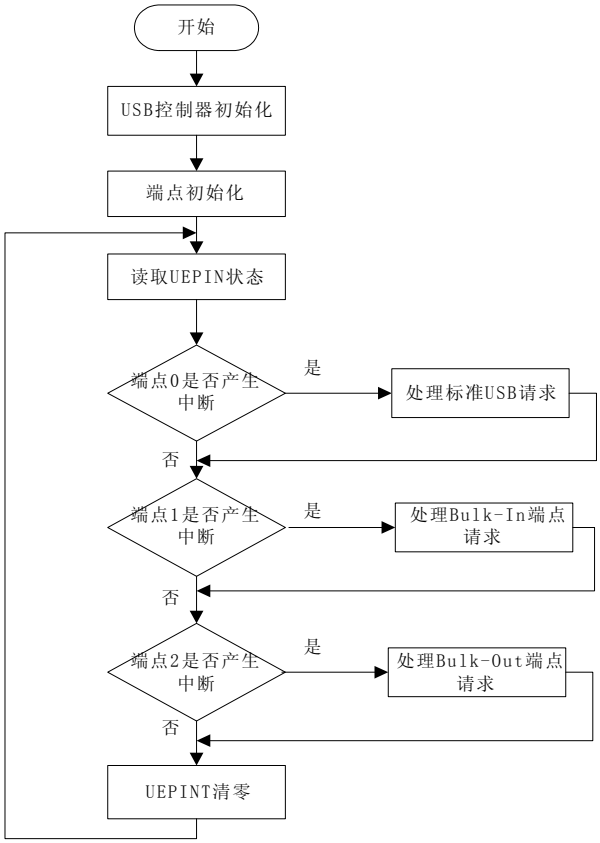


图 33 U 盘程序的流程图

AT89C51SND1 的 USBADDR 寄存器中的 FEN 位本应在接收到 USB 复位请求后才将其设置为 1，以允许其对缺省地址的配置过程进行响应。这里在初始化时直接置 1，在 U 盘流程图中固件不再对 USB RESET 请求进行响应。

USB 设备插入到 PC 机的 USB 接口以后，主机检测到设备插入，就会通过控制管道向缺省地址发送 USB 标准请求，进入设备配置阶段。配置阶段其实就是主机向设备索取各种描述符的过程。每当设备收到主机发送的请求后，便会触发端点中断寄存器中端点 0 所对应的位。此时，应该读取端点 0 的数据缓冲区，读取指定长度（长度由 UBYCTX 寄存器表示）的数据。然后对照 USB 标准设备请求的数据格式，对请求类型进行识别，然后转向相应的标准请求处理函数。

函数 Ep0()就是用来响应 USB 配置过程的，分别响应主机发出的 Set Address、Set Configuration、Get Descriptor 等标准请求，而且这种响应会进行多次。Get Descriptor 要求 USB 设备向主机传送相应的描述符，程序里面定义了设备描述符 Device_Descriptor[18]、Configuration_Descriptor_All[32]等数据结构，后者包括端点描述符等。Set Address 将为设备分配一个有效地址，此请求不要求向主机回传数据，但必须返回状态。返回状态的方法是向端点写入长度为 0 的字节信息。然后，需要根据芯片资料中有关 Set Address 的要求，对芯片的 USBADDR 和 USBCON 两个控制位进行设置。Set Configuration 的作用是为设备选择一个配置，一个功能设备可以有多个配置，配置将按其提供描述符时的顺序编号，而此请求是为设备选定一个配置。

Ep0()函数调用 Set_Address()、Get_Descriptor()、Set_Configuration()三个函数完成上述配置过程的具体工作，包括更改 AT89C51SND1 的相关寄存器。

```
void Ep0()
{ unsigned char data DT[32]={0,};
  unsigned char data i;
  i = ReadEp(0,DT);
  if (((DT[0] & 0x60)==0) && i)
    {switch (DT[1])
       {  case set_address           :Set_Address(DT[2]);           break;
          case get_descriptor        :Get_Descriptor(DT[3],DT[6]);  break;
          case set_configuration     :Set_Configuration(DT[2]);     break;
          default                    :break;
       }
    }
  else if(DT[0]==0xa1)
    {WriteEp(0,0,0);
    }
}
```

接下来就是要处理 Bulk-Only 传输中的 SCSI-2 命令解析了。Mass Storage 协议通过 Bulk 传输方式来传送命令与数据。在这种传输方式下，有三种类型的数据在 USB 和设备之间传送，CBW、CSW 和普通数据。CBW 和 CSW 其实都有一定的格式，普通数据根据其前面的命令块来决定其归属和意义。

程序中用 main_rxdone()函数来处理 Bulk-Out 端点，用 main_txdone()函数来处理 Bulk-In 端点。在 64M 版本中，main_txdone()函数内容为空，程序中实际不存在这个函数。

这是因为 Bulk-In 端点的事务其实都在 main_rxdone()函数中处理掉了，也就是说当 main_rxdone()中解析到某命令需要从 Bulk-In 端点向主机发送数据时，尤其无法一次传送完毕时，并不退出 main_rxdone()函数，而是就地循环陆续发送数据。这样在主控程序循环里就基本上没有机会执行 main_txdone()函数了。

在 main_rxdone()函数里主要解析 CBW 中的 CBWCB 命令块，根据不同的 SCSI-2 命令调用对应的函数进行响应。

```
void main_rxdone()
{ unsigned char data i;
  unsigned char data Buf[64];
  ReadEp(2,Buf);
  bulk_CSW[4] = Buf[4]; bulk_CSW[5] = Buf[5];
  bulk_CSW[6] = Buf[6]; bulk_CSW[7] = Buf[7];
  for(i=0;i<12;i++) bulk_CBWCB[i] = Buf[i+15];
  switch(bulk_CBWCB[0])
    {case Inquiry              : WriteEp(1,36,B_InquiryData);
                                 WriteEp(1,13,bulk_CSW);     break;
     case Mode_Sense           : WriteEp(1,12,B_Mode_Sense);
                                 WriteEp(1,13,bulk_CSW);     break;
     case Read_Capacity        : WriteEp(1,8, B_Read_Capacity);
                                 WriteEp(1,13, bulk_CSW);    break;
     case Read_Format_Capacity  : WriteEp(1,20,B_Read_Format_capacity);
```

```
                                    WriteEp(1,13, bulk_CSW);    break;
     case Test_Unit_Ready       : WriteEp(1,13, bulk_CSW);    break;
     case Verify                : WriteEp(1,13, bulk_CSW);    break;
     case Medium_Removal        : WriteEp(1,13, bulk_CSW);    break;
     case Read10                : SCSI_Read10(bulk_CBWCB);
                                  WriteEp(1,13, bulk_CSW);    break;
     case Write10               : SCSI_Write10(bulk_CBWCB);
                                  WriteEp(1,13, bulk_CSW);    break;
    }
}
```

有趣的是 Read_Format_Capacity 并不是 SCSI 命令，而是 UFI 命令。但是这里如果不响应该命令，就会出现插入 USB 电缆后要等近半分钟才能被 PC 机识别为 U 盘的现象。实际上无需返回任何数据，返回任意数据后再返回 CSW 即可。

主机会向设备发出下列几个 SCSI 命令：Inquiry、Read Capacity、Mode Sense，这几个命令用于告知主机这个设备的存储器的具体情况，主机基于此创建一个可移动存储设备。程序中依据 SCSI-2 命令集定义了这几个命令的返回数据结构 B_InquiryData[]、B_Read_Capacity[]、B_Mode_Sense[]等，其中填充了和 FLASH 逻辑结构即可移动磁盘容量、逻辑块数量、大小等相关参数。如果更换 FLASH 芯片的话需要改写相关数据，这些数据结构的详细情况请查阅文档。

```
code unsigned char B_Read_Capacity[] = {
            0x00, 0x01, 0xff, 0x7f,   //Last  Logical Block Address for 64MB
            0x00, 0x00, 0x02, 0x00    //block length in bytes
                                };
```

64M Flash 一共 4096 个 Block，每 Block 包含 32 个 Page，每 Page 包含 512 Bytes。这里留下 4 个 Block 作为 Flash 的读写缓冲区，可用的最大 Block 编号为 4091。B_Read_Capacity[]数据结构中规定的逻辑块大小为 512 字节，则最后逻辑块编号设定为 4092×32-1＝0x1ff7f。

然后，比较重要的就是 U 盘读写操作了，也就是利用 SCSI_Read10()、SCSI_Write10()两个函数响应 Read10、Write10 两个 SCSI 命令，对 Flash 芯片进行读写操作。

为了保护作者的权益，64M 版本中不再免费公开提供 SCSI_Read10()、SCSI_Write10()的源代码，仅提供包含该函数的库。但为了分析问题，这里给出 32M 版本中的 SCSI_Read10()、SCSI_Write10()的源码分析，仅供参考。

```
  void SCSI_Read10()
  {unsigned char data i;
   union
      { unsigned long   Block;
        unsigned char   Addr[4];
      }data BLK;
   unsigned char length;
   BLK.Addr[2] = bulk_CBW.CBWCB[4];
   BLK.Addr[3] = bulk_CBW.CBWCB[5]; //32M 容量只需两个字节的逻辑块地址
   length = bulk_CBW.CBWCB[8];   //每次读取的逻辑块数由一个字节表示足够
   while(length>0)
     {
      K9F_FUN = COMMAND;   //请参考 FLASH 芯片手册中命令的时序说明
```

```
        K9F5608 = 0x00;
        K9F_FUN = ADDRESS;
        K9F5608 = 0;
        K9F5608 = BLK.Addr[3];
        K9F5608 = BLK.Addr[2];

        K9F_FUN = D_DATA;
        UEPNUM=0x01;     //设置端点 1，Bulk-In
        UEPSTAX|=DIR;
        while(!(K9F_FUN & RB));  //等待 Flash 芯片读命令执行完毕
        for(i=0;i<8;i++)          //USB 每次传输 512 字节数据，但端点 1 只能传送 64 字节
        {                         //需要 8 次端点传输才能完成一次 USB 传输任务
            ReadFlash();   //利用汇编语言从 Flash 芯片读取数据到 UEPDATX 寄存器
            UEPSTAX|=TXRDY;
            while(!(UEPSTAX&TXCMP));  //等 64 字节传送完毕
            UEPSTAX&=(~(TXCMP));
        }
        K9F5608 = INACTIVE;
        length--;
        BLK.Block ++;
      }
  TransmitCSW();   //数据传送完毕返回 CSW 数据结构
}
```

上述代码是相当精简的读 U 盘程序，或许未必完善。USB 每次可以传输多个逻辑块，每个逻辑块 512 个字节需要分 8 次传输完毕。程序中在每次从 Flash 芯片读取 64 个字节到 UEPDATX 寄存器的过程中，调用了汇编语言编写的 ReadFlash()函数，目的是大大缩短字节传送的时间，从而提高宏观 U 盘读取速度，这点在写 U 盘的程序中也有所应用，写 U 盘函数 SCSI_Write10()中使用了 WriteFlash()汇编语言函数来提高速度。

下面是 SCSI_Write10()函数的源代码：

```
void SCSI_Write10()
{ union     //定义联合体是为了按字节操作和按 LONG 型变量整体加 1 操作的方便
    { unsigned long page;
      unsigned char addr[4];
    }data PG;
   unsigned char data i=0,length=0,nBeginPage=0;
   K9F_FUN = COMMAND;       //擦除缓冲区 2046 Block
   K9F5608 = 0x60;
   K9F_FUN = ADDRESS;
   K9F5608 = BuffBlock;
    K9F5608 = 0xff;
   K9F_FUN = COMMAND;
   K9F5608 = 0xd0;
   K9F_FUN = D_DATA;
```

```
 delay();
while(!(K9F_FUN & RB));
K9F_FUN = COMMAND;        //擦除缓冲区 2047 Block
K9F5608 = 0x60;
K9F_FUN = ADDRESS;
K9F5608 = BuffBlock|0x20;    //通过位运算给地址的相应位加 1
 K9F5608 = 0xff;
K9F_FUN = COMMAND;
K9F5608 = 0xd0;
K9F_FUN = D_DATA;

 PG.addr[2] = bulk_CBW.CBWCB[4]; //获得所写逻辑块的地址
 PG.addr[3] = bulk_CBW.CBWCB[5];

 length = bulk_CBW.CBWCB[8];     //要写的逻辑块数量
 nBeginPage = PG.addr[3]&0x1f;   //Flash 地址的低 5 位是 Block 内的 Page 偏移
 UEPNUM = 0x02;  //指定从 EP2 读取数据
 delay();
 while(!(K9F_FUN & RB));

if(nBeginPage>0) //如果所要写的逻辑块，即 Flash 芯片的 Page 偏移不在一个 Block 的开头
  {          //就将该 Block 的前面几个 Page 都复制到缓冲区
     for(i=0;i<nBeginPage;i++)
       {
          K9F_FUN = COMMAND;
          K9F5608 = 0x00;
          K9F_FUN = ADDRESS;
         K9F5608 = 0;                      //A0-A7
         K9F5608 = (PG.addr[3]&0xe0)|i;   //A9-A16
          K9F5608 = PG.addr[2];            //A17-A24
           K9F_FUN = D_DATA;
         delay();
          while(!(K9F_FUN & RB));

          K9F_FUN = COMMAND;  //COPY-BACK 操作在 Flash 内部复制整个 Page
          K9F5608 = 0x8a;
          K9F_FUN = ADDRESS;
         K9F5608 = 0;                              //A0-A7
         K9F5608 = BuffBlock|(PG.addr[3]&0x20)|i;  //位运算决定奇偶缓冲区
          K9F5608 = 0xff;                          //A17-A24
           K9F_FUN = D_DATA;
           delay();
          while(!(K9F_FUN & RB));
        }
```

```
        nBeginPage=0;
    }
  while(length>0) //剩余长度不为 0 就继续循环
    {
        K9F_FUN = COMMAND;  //从 USB 端点取得数据并写入 Flash 芯片缓冲区
       K9F5608 = 0x80;
       K9F_FUN = ADDRESS;
       K9F5608 = 0;                                  //A0-A7
        K9F5608 = (PG.addr[3]&0x3f)|BuffBlock;   //位运算决定奇偶缓冲区
        K9F5608 = 0xff;                              //A17-A24
        K9F_FUN = D_DATA;
        for(i=0;i<8;i++)
          {
             while (!(UEPINT & EP2));  //等待 USB 接收数据包
             WriteFlash(); //调用汇编函数复制数据，从 USB 到 Flash 芯片
             UEPSTAX &= 0xB9;   //清楚标志表示已经读取完毕
          }
        K9F_FUN = COMMAND;
        K9F5608 = 0x10;
        K9F_FUN = D_DATA;
        length--;  //复制完一个逻辑块，剩余长度减 1
        delay();
        while(!(K9F_FUN & RB));

       if(((PG.addr[3]&0x1f)==0x1f)||(length==0))
             //如果剩余长度为 0 或者当前缓冲区 Block 已经写满
         { for(i=((PG.addr[3]&0x1f)+1);i<32;i++)
             { K9F_FUN = COMMAND; //将当前 Block 剩余的 Page 复制到缓冲区，
               K9F5608 = 0x00;    //如果缓冲区满的话，这次复制 0 个 Page
               K9F_FUN = ADDRESS;
               K9F5608 = 0;                              //A0-A7
               K9F5608 = (PG.addr[3]&0xe0)|i;            //A9-A16
                K9F5608 = PG.addr[2];                    //A17-A24
               K9F_FUN = D_DATA;
               delay();
               while(!(K9F_FUN & RB));

               K9F_FUN = COMMAND;
               K9F5608 = 0x8a;
               K9F_FUN = ADDRESS;
               K9F5608 = 0;                                 //A0-A7
               K9F5608 = (PG.addr[3]&0x20)|BuffBlock|i; //A9-A16
                K9F5608 = 0xff;                            //A17-A24
               K9F_FUN = D_DATA;
```

```
    delay();
    while(!(K9F_FUN & RB));
  }

K9F_FUN = COMMAND;
K9F5608 = 0x60;                 //擦除当前 Block
K9F_FUN = ADDRESS;
K9F5608 = PG.addr[3];          //A9-A16
 K9F5608 = PG.addr[2];          //A17-A24
 K9F_FUN = COMMAND;
 K9F5608 = 0xd0;
 K9F_FUN = D_DATA;
 delay();
 while(!(K9F_FUN & RB));        //等待操作完成

for(i=0;i<32;i++)   //将已经组织好的缓冲区复制到当前 Block
  {
     K9F_FUN = COMMAND;
     K9F5608 = 0x00;
     K9F_FUN = ADDRESS;
   K9F5608 = 0;                                 //A0-A7
   K9F5608 = (PG.addr[3]&0x20)|BuffBlock|i; //A9-A16
    K9F5608 = 0xff;                             //A17-A24
      K9F_FUN = D_DATA;
      delay();
    while(!(K9F_FUN & RB));

      K9F_FUN = COMMAND;
      K9F5608 = 0x8a;
      K9F_FUN = ADDRESS;
   K9F5608 = 0;                              //A0-A7
   K9F5608 = (PG.addr[3]&0xe0)|i;      //A9-A16
    K9F5608 = PG.addr[2];             //A17-A24
      K9F_FUN = D_DATA;
      delay();
    while(!(K9F_FUN & RB));
  }

if(length>0)   //如果是剩余长度不为 0 的情况则擦除缓冲区
  {
     K9F_FUN = COMMAND;
   K9F5608 = 0x60;
   K9F_FUN = ADDRESS;
   K9F5608 = (PG.addr[3]&0x20)|BuffBlock;  //A9-A16
```

```
                K9F5608 = 0xff;                          //A17-A24
                K9F_FUN = COMMAND;
                K9F5608 = 0xD0;
                  K9F_FUN = D_DATA;
                  delay();
                  while(!(K9F_FUN & RB));
                }
            }
          PG.page++;  //逻辑块地址加 1，继续从 USB 接收数据
      }

    TransmitCSW();  //本次 USB 传输完毕，返回 CSW
}
```

写 U 盘的处理过程远比读 U 盘的过程要复杂，因为 Flash 芯片必须先擦除再编程，而且每次至少擦除 1 个 Block，也就是说为了写入一个 Page（即一个 USB 逻辑块）可能要擦除 32 个 Page，连续写入的逻辑块不在同一个 Block 就的擦掉多个 Block，那么如何处理这复杂多变的情况呢？我们使用 Flash 芯片最后的若干 Block 作为缓冲区，解决即将被擦掉的数据的备份问题，因为 AT89C51SND1 的 SRAM 资源有限，无法在其中实现缓冲区。实际程序中定义的 BuffBlock 为 2046，使用第 2046 和 2047 Block 作为缓冲区，使用两个连续奇偶 Block 作为缓冲区是和 Flash 芯片的 COPY-BACK 操作有关的。

64M 版本的 Flash 缓冲区实际是 4092、4093、4094、4095 共 4 个 Block，K9F1208 芯片有 4 个 Plane，请阅读芯片文档获得详情。

详细的写 U 盘过程算法叙述如下：

S1. 擦除第 2046 和第 2047 Block，清空缓冲区。

S2. 从 CBWCB 中获得起始逻辑块号和长度，计算出起始逻辑块号对应的第 M 个 Block 内的第 N 个 Page。

S3. 指定端口 2。

S4. 如果 N>0，就把第 M 个 Block 中从 0 到 N-1 的所有 Page 都复制到缓冲区的对应位置。

S5. 计算 M 和 N。如果数据长度为 0 则结束写 U 盘过程。从 USB 端点接收数据并写入缓冲区中的第 N 个 Page，数据长度减 1。

S6. 如果数据长度不到 0 并且缓冲区没有写满的话，起始逻辑块加 1，返回步骤 S5。

S7. 如果数据长度已经为 0 或者缓冲区写满的话，将第 M Block 中第 N+1 到第 31 的 Page 全部复制到缓冲区对应位置。

S8. 擦除第 M Block。

S9. 将缓冲区全部 Page 复制到第 M Block 中。

S10. 如果数据长度不为 0，则擦除缓冲区。

S11. 起始逻辑块加 1，返回步骤 S5。

上述步骤中，对缓冲区的操作指当前有效缓冲区。

在上述写 U 盘的算法中，在 Flash 内部整 Page 的复制都依赖于芯片的 COPY-BACK 命令，这也是提高 U 盘读写速度的一个关键。COPY-BACK 命令要求起始 Page 和目标 Page 属于同一个 Plane，而 32M Flash 芯片有两个 Plane，分别由奇数 Block 和偶数 Block 构成。也就是说 COPY-BACK 命令要求起始 Page 所在 Block 和目标 Page 所在 Block 的奇偶性相同。在实际程

序中是依靠位运算操作，根据当前操作对象 Block 的奇偶性来自动决定缓冲区 Block。程序中采用多处位运算来将逻辑块地址转化为需要的 Block 和 Page 编号，也在一定程度上提高了速度，但需要用户清楚了解 Flash 芯片的编址格式，一般可以认为逻辑块地址的低 5 位就是 Page 偏移，高位则是 Block 编号。Block 编号的最低位体现其奇偶性，通过位运算将该位保留到缓冲区的 Block 编址中即自动选择了有效缓冲区。

学习板上关于 U 盘的分析就到此为止了，仅供参考。要想真正弄清楚这个问题还需要阅读很多的资料和专业书籍，本文档对此无法满足用户的更多要求。

上述内容是基于 32M 版本进行的分析介绍，64M 版本与此类似，有能力的读者可以尝试编写基于 64M Flash 的读写 U 盘函数，注意 K9F1208 具有 4 个 Plane，COPY-BACK 命令只能在同一个 Plane 内使用。

4. MP3 相关代码分析

JMBIE MP3 学习板的 MP3 播放程序不是很复杂，涉及到的文件系统知识和 MP3 文件格式等可以参考相关文档。下面给出一些简单的相关知识。

预备知识

@关于 FAT 文件系统

一个 FAT（FAT12 / FAT16 / FAT32）文件系统卷（卷可以理解为是一张软盘、一个硬盘或是一个 Flash 电子盘）由四个部分组成：

1）保留区（Reserved Region）

分区的保留区（Reserved Region）中的第一个扇区必须是 BPB(BIOS Parameter Block)，此扇区有时也称作“引导扇区”、“保留扇区”或是“零扇区”，因为它含有对文件系统进行识别的关键信息，因此十分重要。

2）FAT 区

FAT 即 File Allocation Table，文件分配表。操作系统分配磁盘空间按簇来分配的。因此，文件占用磁盘空间时，基本单位不是字节而是簇，即使某个文件只有一个字节，操作系统也会给他分配一个最小单元——即一个簇。为了可以将磁盘空间有序地分配给相应的文件，而读取文件的时候又可以从相应的地址读出文件，我们把整个磁盘空间分成 32K 字节长的簇来管理，每个簇在 FAT 表中占据着一个 16 位的位置，称为一个表项。

对于大文件，需要分配多个簇。同一个文件的数据并不一定完整地存放在磁盘的一个连续的区域内，而往往会分成若干段，像一条链子一样存放。这种存储方式称为文件的链式存储。为了实现文件的链式存储，磁盘上必须准确地记录哪些簇已经被文件占用，还必须为每个已经占用的簇指明存储后继内容的下一个簇的簇号，对一个文件的最后一簇，则要指明本簇无后继簇。这些都是由 FAT 表来保存的，FAT 表的对应表项中记录着它所代表的簇的有关信息：诸如是否空，是否是坏簇，是否已经是某个文件的尾簇等。

下表是 BPB 的结构。

名称	偏移	长度	说明
BS_jmpBoot	0	3	jmpBoot[0] = 0xEB， jmpBoot[1] = 0x??， jmpBoot[2] = 0x90 和 jmpBoot[0] = 0xE9， jmpBoot[1] = 0x??， jmpBoot[2] = 0x?? 0x??表示此处可以为任意字节，任一种选择都可以。典型值 EB 03 90
BS_OEMName	3	8	Microsoft 的操作系统并不关心此域，但一些 FAT 驱动比较在乎，推荐使用“MSWIN4.1”字符串，你完全可以用你自己的名字，无所谓。
BPB_BytsPerSec	11	2	每扇区字节数。只能是：512， 1024， 2048 或 4096。对于一些老的系统，只能使用 512，Dos 下均为 512，建议采用 512。三星的 Flash 的每个 Page 的大小为 512，这在做电子盘的时候也比较方便。
BPB_SecPerClus	13	1	每簇扇区数。此值不能为零，且必须是 2 的整数次方。如 1， 2， 4， 8， 16， 32， 64 及 128。但是此值不要使 BPB_BytsPerSec*BPB_SecPerClus>32K (32 * 1024)。即每簇不要超过 32K 字节。
BPB_RsvdSecCnt	14	2	保留区域中的保留扇区数。保留扇区从第 1 个扇区开始，对于 FAT12 和 FAT16，这时必须填 1。对于 FAT32，此处为 32。
BPB_NumFATs	16	1	此卷中 FAT 结构的份数。必须为 2
BPB_RootEntCnt	17	2	对于 FAT12 和 FAT16 卷，此域中为根目录项数（每个项长度为 32 字节），对于 FAT32，此域为零。此值乘上 32 后必须为 BPB_BytsPerSec 的整数倍。为了达到最好的兼容性，FAT16 卷应使用 512。
BPB_TotSec16	19	2	此域为存储卷上的扇区总数。包括 FAT 表的四个区域的所有扇区数。此域可以为零，当为零时，BPB_TotSec32 必须非零。对于 FAT32，此域必须为零。对于 FAT12 或 FAT16，此域为扇区总数，如果总扇区数小于 0x10000，则 BPB_TotSec32 为零。
BPB_Media	21	1	对于固定存储介质，使用 0xF8，对于可移动存储介质，使用 F0。0xF0， 0xF8， 0xF9， 0xFA， 0xFB， 0xFC， 0xFD， 0xFE， 和 0xFF 都是合法的值，但是在 FAT 表中的 FAT[0]的低位必须与之一致。
BPB_FATSz16	22	2	FAT12/FAT16 每个分区表所占的扇区数。对于 FAT32，此域为 0， 在 BPB_FATSz32 中分组含 FAT32 的分区数。
BPB_SecPerTrk	24	2	每道扇区数。对于非磁头、柱面、扇区结构的介质，此域可为零
BPB_NumHeads	26	2	磁头数。对于非磁头、柱面、扇区结构的介质，此域可为零
BPB_HiddSec	28	4	此 FAT 表所在的分区前面的隐藏扇区数。对于非分区的介质，此值可为零，与操作系统有关。
BPB_TotSec32	32	4	对于 FAT32，此域非零；对于 FAT12/FAT16，如果扇区总数超过 0x10000，则此域为扇区总数。
BS_DrvNum	36	1	操作系统有关参数，软盘使用 0x00，硬盘使用 0x80。
BS_Reserved1	37	1	保留（供 NT 使用）。必须为 0。
BS_BootSig	38	1	扩展引导标记(0x29)。此标记用来指示其后的三个域可用。
BS_VolID	39	4	卷的序列号。此域与 BS_VolLab 一起可支持对可移动磁盘的跟踪，用来判断是否是正确的磁盘。FAT 文件系统可据此判断是否将错误的软盘插到了软驱中。此域常用当前日期和时间来组成。
BS_VolLab	43	11	11 个字节长的卷标，此域需与要目录中的卷标一致

BS_FilSysType	54	8	“FAT12”，“FAT16”或“FAT”之一。此域仅仅是一个标志，操作系统并不关心它，也不用它来确实文件系统的类型。
Executable Code	62		如果是引导分区，它接过系统控制权后，可以执行一列引导操作
Executable Marker	*510*		55 AA

下表是 FAT 的结构

表项	代码示例	含义
0	FFF8	磁盘标识字，必须为 FFF8，注意高位在后
1	FFFF	第一簇已经被占用
2	0003	0000h 可用簇 0002h-FFEFh 已占用，代码代表存放文件的簇链中的下一个簇的簇号 FFF0h-FFF6h 保留簇 FFF7h 坏簇 FFF8h-FFFF 文件的最后一簇
3	0004	
4	0005	
5	FFFF	
6	0000	

FAT 的项数与磁盘上的总簇数相关（因为每一个项要代表一个簇，簇越多当然需要的 FAT 表项越多），每一项占用的字节数也与总簇数有关（因为其中需要存放簇号，簇号越大当然每项占用的字节数就大）。FAT 的格式有多种，最为常见是 FAT16 和 FAT32，其中 FAT16 是指文件分配表使用 16 位，由于 16 位分配表最多能管理 65536（即 2 的 16 次方）个簇，又由于每个簇的存储空间最大只有 32KB，所以在使用 FAT16 管理磁盘时，每个分区的最大存储容量只有（65536×32 KB）即 2048MB，也就是我们常说的 2G。现在的硬盘容量是越来越大，由于 FAT16 对硬盘分区的容量限制，所以当硬盘容量超过 2G 之后，用户只能将硬盘划分成多个 2G 的分区后才能正常使用。

由于 FAT 对于文件管理的重要性，所以 FAT 有一个备份，即在原 FAT 的后面再建一个同样的 FAT。

由于 FAT32 在 64M 的小容量移动存储器中无法发挥其优势，反而表现出处理效率低，占用更多有效存储空间等缺点，所以在 MP3 中完全可以不考虑 FAT32。本学习板只支持 FAT16，U 盘在格式化的时候必须选择 FAT16。

3）根目录区（Root Directory Region）

紧接着第二个 FAT 表的后面一个扇区，就是根目录区了。根目录区中存放目录项，每个目录项为 32 个字节，记录一个文件或目录的信息 (长文件名例外) 。

目录项所占的扇区数与有多少个目录项有关，它将占用（目录项＊32 / 512）个扇区。

4）文件和目录数据区

目录项的所占的最后一个扇区之后，便是真正存放文件数据或是目录的位置了。

目录项格式：

偏移	长度	说明	格式	备注
00H	8	文件名	ASCII 字符，当首字母如下时为特殊代码： 00H=未用名称 05H=当文件的第一个字符为 E5H 时，必须换成 05H，因为 E5H 在首字母时另有含义。 E5H=文件已使用，但已经删除 2EH=本项为目录	不足八个字节时，必须以空格填满
08H	3	文件类型	ASCII 字符	不足三个字节时必须填满
0BH	1	文件属性	7 6 5 4 3 2 1 0 未定义 未定义 存档 目录 卷标 系统 隐藏 只读	
0CH	10	保留		
16H	2	上次更新时间	须经编码 :(unsigned 16 bit-bit integer) time=Hr*2048+Min*32+Sec+2	*:高位在后，低位在前 LSB
18H	2	上次更新日期	须经编码 :(unsigned 16 bit-bit integer) time=(Yr-1980)*512+Mon*32+Day	*:高位在后，低位在前 LSB
1AH	2	起始簇号	此文件开始的簇号，如果文件有多簇，根据 FAT 中与此对应项的信息可得下一簇簇号	
1CH	4	文件大小	文件长度	

@有关 MP3 文件结构的知识

MP3 文件是由帧(frame)构成的，帧是MP3 文件最小的组成单位。MP3 的全称应为MPEG1 Layer-3 音频文件，MPEG(Moving Picture Experts Group)在汉语中译为活动图像专家组，特指活动影音压缩标准，MPEG音频文件是MPEG1 标准中的声音部分，也叫MPEG 音频层，它根据压缩质量和编码复杂程度划分为三层，即Layer-1、Layer2、Layer3，且分别对应MP1、MP2、MP3 这三种声音文件，并根据不同的用途，使用不同层次的编码。MPEG 音频编码的层次越高，编码器越复杂，压缩率也越高，MP1 和MP2 的压缩率分别为4：1 和6：1-8：1，而MP3 的压缩率则高达10：1-12：1，也就是说，一分钟CD 音质的音乐，未经压缩需要10MB的存储空间，而经过MP3 压缩编码后只有1MB 左右。不过MP3 对音频信号采用的是有损压缩方式，为了降低声音失真度，MP3 采取了“感官编码技术”，即编码时先对音频文件进行频谱分析，然后用过滤器滤掉噪音电平，接着通过量化的方式将剩下的每一位打散排列，最后形成具有较高压缩比的MP3 文件，并使压缩后的文件在回放时能够达到比较接近原音源的声音效果。

MP3 文件大体分为三部分：TAG_V2(ID3V2)，Frame，TAG_V1(ID3V1)

ID3V2	包含了作者，作曲，专辑等信息，长度不固定，扩展了ID3V1 的信息量。
Frame . . Frame	一系列的帧，个数由文件大小和帧长决定 每个FRAME 的长度可能不固定，也可能固定，由位率bitrate 决定 每个FRAME 又分为帧头和数据实体两部分 帧头记录了mp3 的位率，采样率，版本等信息，每个帧之间相互独立
ID3V1	包含了作者，作曲，专辑等信息，长度为128BYTE。

每个FRAME 都有一个帧头FRAMEHEADER，长度是4BYTE（32bit）,帧头后面可能有两个字节的CRC 校验，这两个字节的是否存在决定于FRAMEHEADER 信息的第16bit，为0 则帧头后面无校验，为1 则有校验,校验值长度为2 个字节，紧跟在FRAMEHEADER 后面，接着就是帧的实体数据了，格式如下：

FRAMEHEADER	CRC（free）	MAIN_DATA
4 BYTE	0 OR 2 BYTE	长度由帧头计算得出

在帧头中包含着MP3解码器所需的重要信息，有采样率、位率、声道数等。

1）每帧的播放时间：无论帧长是多少，每帧的播放时间都是26ms；

2）数据帧大小：

```
FrameSize = (((MpegVersion == MPEG1 ? 144 : 72) * Bitrate) / SamplingRate) +
             PaddingBit
```

例如: Bitrate ＝ 128000, a SamplingRate ＝44100, and PaddingBit ＝ 1

```
FrameSize = (144 * 128000) / 44100 + 1 = 417 bytes
```

MAIN_DATA 部分长度是否变化决定于FRAMEHEADER 的bitrate 是否变化，一首MP3 歌曲，它有三个版本：96Kbps（96 千比特位每秒）、128Kbps 和192Kbps。Kbps（比特位速率），表明了音乐每秒的数据量，Kbps 值越高，音质越好，文件也越大，MP3 标准规定，不变的bitrate 的MP3 文件称作CBR，大多数MP3文件都是CBR 的，而变化的bitrate 的MP3 文件称作VBR，每个FRAME 的长度都可能是变化的。

每个ID3V2.3 的标签都一个标签头和若干个标签帧或一个扩展标签头组成。关于曲目的信息如标题、作者等都存放在不同的标签帧中，扩展标签头和标签帧并不是必要的，但每个标签至少要有一个标签帧。标签头和标签帧一起顺序存放在MP3 文件的首部。在文件的首部顺序记录10 个字节的ID3V2.3 的头部。数据结构如下：

```
char Header[3]; /*必须为"ID3"否则认为标签不存在*/
char Ver; /*版本号ID3V2.3 就记录3*/
char Revision; /*副版本号此版本记录为0*/
char Flag; /*存放标志的字节，这个版本只定义了三位，稍后详细解说*/
char Size[4]; /*标签大小，包括标签头的10 个字节和所有的标签帧的大小*/
```

在我们的程序中没必要关心标签帧的内容，只需判断是否有标签头，以及获得标签大小，全部跳过即可。

回到 JMBIE MP3 学习板

MP3 播放的主控程序如下：

```
MP3_Init();
KeyBoardInit();
MP3InitFlag = 1;
NowPlaying=0;   //指示当前播放的歌曲编号
Init_FAT_Info();
NumofSong = GetMP3List();  //获取 MP3 列表
while(1)   //MP3 播放功能的循环
{   for(i=0;i<8;i++)
        PlayingSong[i]= SONG[8 * NowPlaying + i];
    if(NumofSong != 0)
    { if(MP3InitFlag)  //如果要开始播放某 MP3 文件，则初始化该歌曲采样率等信息
        { PlayInit(PlayingSong)
          MP3InitFlag = 0;  //初始播放标志清零
        }
      PlayMP3(PlayingSong);  //进入该歌曲播放状态
    }
}
```

程序中首先用下列函数进行和 MP3 播放有关的时钟设置，MP3 解码器初始化和音频部件初始化。请阅读 AT89C51SND1 的文档了解相关寄存器的设置。

```
void MP3_Init(void)
{
    PllInit();
    MP3Init();
    AudioInit();
}
```

然后利用函数 Init_FAT_Info()初始化 FAT 文件系统，获取必要的参数信息，这里比较重要的有 SecPerClus，BytesPerSec，FatSize16，RootEntCnt，RootStartSec，FirstDataSec 等。

GetMP3List()的功能是扫描目录区，获取扩展名为 MP3 的文件清单，实际上程序最多能保存 24 首歌曲的列表，所定义的 SONG[]数组只有 24 个单元。

在每首 MP3 歌曲开始播放的时候，会调用 PlayInit()函数对该 MP3 歌曲进行初始化，该函数的功能是解析 MP3 文件头，获取采用率等信息并正确设置 MP3 解码器的时钟参数。如果开始检测到有标签 ID3V2.3 的标签头，则获得标签大小后将这些标签帧全部跳过，程序中并不关心标签中的信息。

```
    if (Page_Buf[0] == 0x49)
        if ((Page_Buf[1] == 0x44) && (Page_Buf[2] == 0x33))
            {//检测到标签 ID3V2.3 的标签头
              total_size = (Page_Buf[6] & 0x7F) * 0x200000 + (Page_Buf[7] & 0x7F)
* 0x4000 + (Page_Buf[8] & 0x7F) * 0x80 + (Page_Buf[9] & 0x7F);
              //求得标签的大小
             while (total_size > 512)
```



```
            {   ReadSector(SongName, Page_Buf);
                total_size -=512;
            } //读出所有的标签帧数据，但并不使用这些数据

             i = total_size;
            }
```

下面这行程序的目的是跳过扩展标签头，使指针直接后移 10 个字节即可。

```
if (Page_Buf[i] != 0xFF)        i += 10;
```

然后将帧头读取到 MP3_Framehead[]数组中，根据 MP3 帧头的格式，分离出 MPEG 版本和采样率标识来，调用 MP3FsInit()函数设置 PLL 时钟参数来满足 MP3 解码器和音频输出部件的采用率时钟要求。

```
if (MP3_Framehead[1] & 0x08) // MPEG Version 1
    { switch ((MP3_Framehead[2] & 0x0C) >> 2) //分别处理不同的采样率标识
      {
       case 0x00 : MP3FsInit(24, 126, 3, 5); break;          //Fs=44.1kHz
       case 0x01 : MP3FsInit(124, 575, 3, 4); break;         //Fs=48kHz
       case 0x02 : MP3FsInit(124, 511, 3, 9); AUDCON0 = 0x76; break;//Fs=32kHz
       default : break;
       }
    }
  else   // MPEG Version 2
    {switch ((MP3_Framehead[2] & 0x0C) >> 2)
      { case 0x00 : MP3FsInit(24, 126, 3, 11); break;         //Fs=22.05kHz
       case 0x01 : MP3FsInit(124, 575, 3, 9); break;         //Fs=24kHz
       case 0x02 : MP3FsInit(124, 511,3,19); AUDCON0 = 0x76; break; //Fs=16kHz
       default : break;
       }
    }
```

MP3 歌曲的采样率设置好以后，就调用 PlayMP3()函数播放该歌曲，如果没有控制按键动作的话，该函数在播放完歌曲后才退出，主控程序里会继续播放下一首歌曲。

```
void PlayMP3(unsigned char *SongName)
{ int i =0,m =0;
  while (1)      //歌曲播放主要在这个循环里
   { RDCOUNT = ReadSector(SongName,0,Page_Buf); //读取一扇区 MP3 数据
     for (i=0; i<RDCOUNT; i++)      //此循环将最多 512 字节的数据送 MP3 解码器
       {while (!(MP3STA1 & MPBREQ)) //等待 MP3 解码器握手信号......
           while (!PlayState);       //此间可以干一些其他事情，判断是否暂停
        MP3DAT = Page_Buf[i];        //传送一个字节到 MP3 解码器
       }
     if (ChangeSong)
       { ChangeSong = 0;
         m =1;
        }
```

```
      if (RDCOUNT < 512)        //该歌曲播放完毕，最后读取扇区数据不足 512 字节
        { NowPlaying++;      //自动切换到下一曲
          if(NowPlaying == NumofSong)
               NowPlaying= 0;
          m =1;
        }
      if(m)
         {MP3InitFlag = 1;    //换曲后重新设置歌曲初始化标志
          while(MP3STA1 & MPFREQ) //当切换歌曲或者播放完毕后，
             MP3DAT = 0x00;         //补充 MP3 解码器的数据直至缓冲区满
          return;
         }
    }
}
```

在这段程序中，将有大量的数据从 Flash 芯片读取到 SRAM 的缓冲区中，然后再填充到 MP3 解码器的缓冲区中。但这里存在一个需要设计者清醒认识到的问题，就是程序运行过程中的时间、空间、数据之间的关系，也就是程序的效率问题。实际上此类程序的分析属于按数据流程思考问题的例子。具体的说，不能出现 MP3 解码器数据填充的间断，否则将出现断音现象，这是很多程序设计不当而导致声音杂乱（听起来失真，慢拍，实际是均匀连续的断音）的原因所在。为了保证不出现断音的问题，需要确保程序填充 MP3 解码缓冲区的速度大于 MP3 解码速度。具体到程序中，从 SRAM 的缓冲区复制一个字节到 MP3 解码器后，需要等待 MP3 解码器的握手信号，不能立即复制下一个字节。在这段等待时间里可以处理其他事情，比如键盘中断服务程序设置的标志位的查询和响应，本程序中判断是否暂停。但在此处理的事情不能太多太费时间，比如大量的计算就不适合。如果这段程序没有处理好的话，每复制一个字节超时一点点，传送完 512 字节积累起来的超时时间就可能导致 MP3 解码器入不敷出。有的时候需要统计文件进度等，可以在这 512 字节都复制完之后进行，每读一个扇区才计算一次，还是不容易出问题的。另外一点就是，要先准备好一个扇区再判断 MP3 解码器握手信号，不要等查询到解码器需要数据了才去 Flash 里读取。

程序里用函数 ReadSector()读取 MP3 文件中的一个扇区数据：

为了保护作者的权益，在 64M 版本中不再免费公开提供 ReadSector()函数的源代码，仅提供包含该函数的库。但为了分析问题，这里给出 32M 版本的函数源码，仅供参考。

```
int ReadSector(unsigned char *Name, unsigned char *databuff)
  { int i, k, Page;
    unsigned  long CurrentSector;
    if (DataRead == 0) //该变量在初次读取该文件前必须为 0
      { Page = BootSector + RsdSector + 2 * SectorofFatSize;//计算目录区位置
        ReadPage(0 + Page / 32, Page % 32, databuff); //读取目录区到缓冲区
        while (databuff[0] != 0)
        { for (i=0; i<16; i++)    //扫描目录区
          {if (!memcmp(Name, &databuff[i * 32], 11))    //查找与歌曲名字相符的目录项
            {Current_Cluster = databuff[32 * i + 27] * 256 + databuff[32 * i + 26];
             //计算该文件在磁盘数据区的第一个簇的位置
             for (k=31; k>27; k--)   //计算该文件的大小
                DataRemain = (DataRemain << 8) | databuff[i * 32 + k];
```

```
                CurrentSector = (Current_Cluster - 2) * SecPerClus + FirstDataSec;
                //计算该文件第一个扇区的位置
                ReadPage(CurrentSector / 32, CurrentSector % 32, databuff);
                DataRead += 512;
                DataRemain -= 512;
                if (DataRemain < 0) // 已经读到文件的最后一个扇区
                     { DataRead = 0;   //该变量在此清 0
                       return (DataRemain + 512);  //返回最后扇区有效数据的大小
                     }
                   else
                     return (512);
                }
              }
              Page++;    //读取目录区的下一个扇区
              ReadPage(0 + Page / 32, Page % 32, databuff);
            }
            return (0);
          }
        else   //如果变量 DataRead 不等于 0 则不再搜索目录区
          { Current_Cluster++;  //当前簇线性加 1
            CurrentSector = (Current_Cluster - 2) * SecPerClus + FirstDataSec;
            ReadPage(CurrentSector / 32, CurrentSector % 32, databuff);
            DataRead += 512;
            DataRemain -= 512;
            if (DataRemain < 0)
                {   DataRead = 0;
                    return (DataRemain + 512);
                }
            else return (512);
           }
}
```

如果用户仔细阅读了和 FAT 文件分配表相关的文档并对文件系统有个比较清楚的认识，就发现这段程序并非完全按照文件系统的规定去写的。这里有两个假设：第一，一个 MP3 歌曲文件在 Flash 中是线性递增连续存放的；第二，一个簇只有一个扇区。在程序中，DataRead 变量用来表示已经读取的字节数，如果该变量为 0 则表示刚开始读取文件，需要在目录区中查找该文件的起始扇区。之后便认为上述两个假设成立，表示当前簇的变量 Current_Cluster 连续加 1，再用 ReadPage 函数从 Flash 中读取一个 Page。如果每个簇不只有 1 个扇区，也就是 SecPerClus 不等于 1 的话，就必须依次读取完本簇内的扇区之后才能使 Current_Cluster 加 1。另外，既然是 FAT 链式文件存储结构，就应该回到 FAT 中查找下一簇的簇号，而非当前簇号简单加 1。当然，更符合一般文件系统软件的做法是定义文件控制块，开辟文件缓冲区，编写打开、关闭、定位、读写等函数，这样就支持按文件名同时打开多个文件，随机读取数据了，有兴趣的用户可以自己改写这段程序。

实际情况中，32M 版本中 FAT16 下 SecPerClus 就是 1，而一般情况下歌曲文件是连续存放的，所以一般可以正常播放，但不能保证完全正确。

在 64M 版本中，ReadSector()函数已经完全正确支持 FAT16，可以处理 SecPerClus 不等于 1 的情况，并能正确从 FAT 中获取当前簇的下一簇地址。所以无论 U 盘中的文件是否连续都可以正确读取该文件的全部内容。实际上，FAT16 管理 64M 空间时，SecPerClus = 2。有能力的用户可以尝试编写正确处理 FAT16 的 ReadSector()函数，注意在 FAT 中查找下一簇的算法。

在 MP3 的播放过程中，可以使用按键对歌曲进行控制，这依赖于 AT89C51SND1 的键盘中断。本学习板打开了 4 个按键的键盘中断，提供如下的键盘中断服务程序：

```
void key_interrupt() interrupt 11
{       unsigned char i,j,k;
        EA = 0;
        IEN1 &= (~EKB);           //关闭键盘中断
        k=KBSTA & 0x0f;           //读取键值
        for(j=0;j<50;j++)          //延时 20ms，用于去抖
          for(i=0;i<200;i++);
      if(k==(~P1 & 0x0f))   //如果 20ms 后 P1 口键盘电平和初始寄存器的值相同则认为键有效
        switch (k)
        {   case 1 : Func(); break;         //调用相应的函数
            case 2 : Next(); break;
            case 4 : Previous(); break;
            case 8 : PlayPause(); break;
            default : break;
        }
        IEN1 |= EKB;             //重新打开键盘中断
        EA = 1;
        k=KBSTA;       //再次读取，清空键盘状态
        return;
}
```

这段程序中依据按键的不同调用不同的功能函数，执行具体功能。程序中用 CurrentFun 变量表示当前的键盘状态，按 FUNC 按键时该变量循环改变。NEXT 和 PREV 按键对应的函数则根据 CurrentFun 变量决定具体功能，设置有关寄存器或者修改变量，这些函数的执行结果将被 PlayMP3()函数查询到，并改变 MP3 播放状态。

这个函数也是在单片机 C51 语言中编写中断服务程序 ISR 的例子，函数定义中

```
void key_interrupt() interrupt 11
```

设定该函数 key_interrupt()是 11 号中断的 ISR。

5. UART 相关代码分析

在程序调试的时候，需要从串口输出一些信息，帮助设计者了解程序运行的状况。这里编写了几个简单的串口输出函数。

在串口初始化函数里，要注意设置 CKCON 寄存器使得 UART 的时钟不受 X2 位影响。

```
void init_uart(int fre)
{
 if (fre == 1200)      //初始化程序只支持 1200 的波特率
 {   CKCON |=0x04;     //设置时钟
     TMOD = 0x20;      //设置定时器 1 作为 UART 的波特率发生器
     TH1  = 0xa9;
     TL1  = 0xa9;
     SCON = 0x50;
     PCON = PCON|0x80;
     TR1  = 1;
 }
}
```

由于 C51 程序的限制以及简单的功能定义，这里提供的 printu()函数只能显示固定的字符串常量或变量，不支持变参功能。而 printuf()也不同于 PC 机中 C 语言的同名函数，这里的格式字符串里只能包含一个“%x”或“%d”或“%c”，也就是函数的第 2 个参数可以按 16 进制、10 进制或字符来出现在最终显示的字符串中，不支持其他格式定义，也不支持变参。

6. LCD 相关代码分析

LCD 程序是针对外接的 12232 系列点阵图形 LCD 编写的驱动程序，其效果直接和 LCD 硬件有关系，所以不能保证这些函数能支持所有型号为 12232 的 LCD。学习板开发过程中用到的 CM12232 LCD 的硬件资料和电路连接图都可以在光盘里找到。

在 LCD.H 文件中定义了硬件接口：

```
sbit LCD_E1 =P1^4;   //E1,LCD Pin 5,H active
sbit LCD_E2 =P1^5;   //E2,LCD Pin 6,H active
sbit LCD_RW =P1^6;   //RW,LCD Pin 7,H=Read,L=Write
sbit LCD_DI =P1^7;   //A0,LCD Pin 8,H=Data,L=Instruction

#define  LCD_BUS P4   //LCD DATA BUS
```

结合 LCD 硬件的时序和命令定义，编写了发送命令函数 SendCommand()和发送数据函数 SendData()。

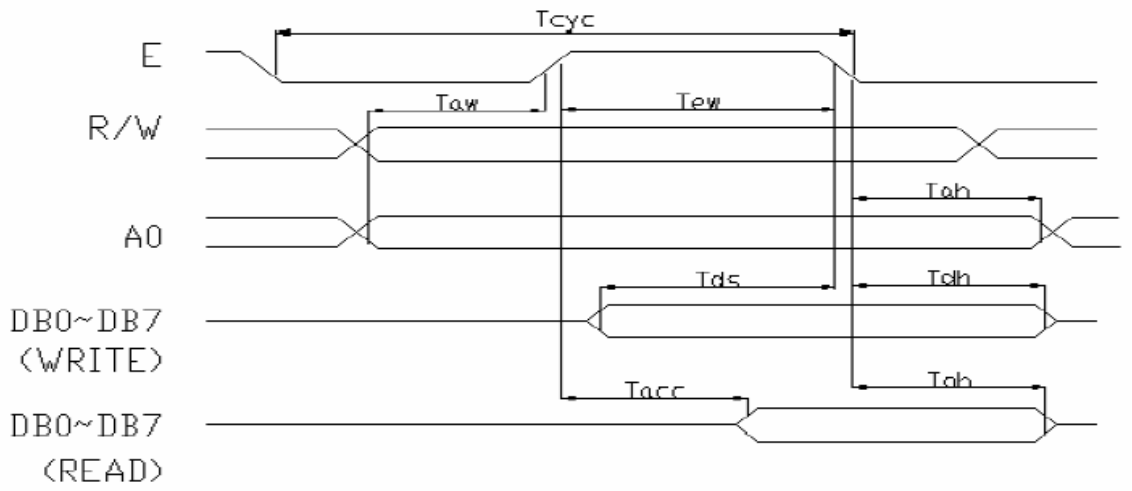
LCD 的读写时序图如图 34 所示。

```
void SendCommand(unsigned char cmd)
{LCD_E1 =0;
 LCD_E2 =0;
 LCD_RW =0;
 LCD_DI =0;
 LCD_BUS =cmd;
 LCD_E1 =1;
 LCD_E2 =1;
 LCDdelay();
```

```
    LCD_E1 =0;
    LCD_E2 =0;
    LCDdelay();
    LCD_BUS =0xff;
  }
```

这款 LCD 内部含有 MASTER 和 SLAVE 两个控制器，实际上 MASTER 给 SLAVE 提供脉冲使两个控制器协调工作，共同完成 122×32 点阵的显示。光盘里提供了一个“SED1520”的芯片文档，可以通过该文档了解这种主从结构的 LCD 工作原理。LCD_E1 这个信号实际上控制的是 MASTER，LCD_E2 则控制 SLAVE，其他信号线二者并联共用。SendCommand()函数里将命令参数同时打入两个控制器的。

MASTER 和 SLAVE 两个控制器共同构成整个 LCD 点阵画面的行列驱动，点阵的实际分布如图 35 所示。在发送数据函数中，参数 seg 认为是从左到右为 0 到 121 的水平象素编号，而参数 page 则认为从上到下为 0 到 3 的垂直象素组编号。所谓垂直象素组，指的是在每一列象素中 32 个点分为 4 个组，每组 8 个点正好一个字节数据，每次只能一起写入这 8 个点的数据而不能寻址到单个象素。数据字节中 bit 值为 1 的对应点显示，为 0 则不显示。详细象素和数据对应关系见图 35，图中的象素编号顺序和上述的写数据函数参数正好相反，函数内部自动转换。



Bus Read /Write Operaion Sequence

图 34　CM12232　LCD 的读写时序

Page	Data		Com No	Drive
3	D7 : D0	: : 122 X 8 PLXELS	31 : :	Slave
2	D7 : D0	: : 122 X 8 PLXELS	16	
1	D7 : D0	: : 122 X 8 PLXELS	15 : :	Master
0	D7 : D0	: : 122 X 8 PLXELS	0	
Column Addr	ADC=0	3CH ····00H	3CH ····00H	
	Seg No	121 ···· 61	60 ···· 0	
	Drive	Slave	Master	

图 35　LCD 的象素和主从控制器、数据位的对应关系

```
void SendData(unsigned char seg,unsigned char page,unsigned char dots)
{   //seg is from 0 to 121 and page is from 0 to 3.
 page=3-page;          //反转编址顺序，将习惯顺序转换为 LCD 实际顺序
 SendCommand(LCDCOMD_PAGEADDR|page);  //写入 PAGE 地址

 if (seg<61)
     {   seg=60-seg;  //反转水平顺序
         SendCommand(LCDCOMD_SEGADDR|seg);  //写入水平 SEG 地址
         LCD_DI =1;
         LCD_RW =0;
         LCD_BUS =dots;
         LCD_E2 =1;       //当 seg<61 时对应 LCD 左侧区域，数据应打入 SLAVE 驱动器
         LCDdelay();
         LCD_E2 =0;
         LCDdelay();
     }
  else
     {   seg=121-seg;
         SendCommand(LCDCOMD_SEGADDR|seg);
         LCD_DI =1;
         LCD_RW =0;
         LCD_BUS =dots;
         LCD_E1 =1;     //当 seg>61 时对应 LCD 右侧区域，数据应打入 MASTER 驱动器
         LCDdelay();
```

```
            LCD_E1 =0;
            LCDdelay();
        }
    LCD_BUS=0xff;
   }
```

然后是上层的字符和点阵图的显示函数了。

```
void DisplayBMP(unsigned char seg,unsigned char page,unsigned char *bmp)
{unsigned char x,y;
 for(x=0;x<bmp[0];x++)
    for(y=0;y<bmp[1];y++)
       SendData(seg+y,page+x,bmp[2+x*bmp[1]+y]);
}
```

这个函数用来显示简单的位图，它的参数 bmp 指向一个特定的数据结构，程序中定义了 JMBIE_BMP[]一维数组来显示 JMBIE 的 LOGO。在该数据结构中，JMBIE_BMP[0]是后续数据表示的图形的垂直象素组的数量值，而 JMBIE_BMP[1]则是水平象素的数量值，DisplayBMP()函数根据这两个值决定循环次数、seg 和 page 的偏移、计算并取得正确位置的字节数据。

这是用来显示 5×8 点阵的小 ASCII 字符的函数，每个字模水平 5 个点，垂直 8 个点。字模数据定义在 CharTab[]数组中，最后编译到可执行程序中。

```
void LCD_printen(unsigned char seg,unsigned char page,unsigned char *str)
{unsigned char i=0,j=0,x,buffer;
  while(str[i]!='\0')   //检索字符串
   {x=seg+i*6;   //实际上每个字符占 6 个水平象素宽度，两个字符之间留下一个点的间距
    if(x>116)
      break;    //如果字符串太长则放弃显示超出 LCD 水平范围的字符
    SendData(x++,page,0x00);  //将空的间距送到 LCD
    for(j=0;j<5;j++)
     {buffer=CharTab[(str[i]-0x20)*5+j]; //通过字符的 ASCII 码定位其字模
      SendData(x+j,page,buffer);
       }
    i++;

    }
}
```

这是显示 8×16 的 ASCII 字符和 16×16 的汉字的函数，该函数所使用的字模即汉字库太大而不能在程序中定义，作为文件存放到 Flash 中，进入 U 盘模式后可以和歌曲一起下载到 U 盘。这个函数里涉及到汉字库的逻辑结构，字模内含定义，通过汉字内码提取字模等问题，用户可以在网上搜索相关文章来了解。为了能方便驱动这款 LCD，光盘里提供的汉字库是我们特地制作的，和 UCDOS 下的 HZK16 在汉字顺序上没有区别，汉字内码到字模的映射未变，但字模内部的点顺序按 LCD 的特征重新排序了。另外那个字库文件实际是将 8×16 的 ASCII 字符字模库和 16×16 的汉字字模库分别处理后合并得到的。

```
void LCD_printch(unsigned char seg,unsigned char page,unsigned char *str)
{unsigned long addr;
 unsigned char buffer[32],i,j,xx=0,t;

 if(!CHFontInitOK) return;
 for(t=0;str[t]!=0;t++)
  {if(str[t]&0x80) //该字符是汉字，汉字内码占俩字节，每个字节的最高位为 1
    {
     addr=(long)((str[t]-0xa1)*94+str[t+1]-0xa1)*32+256*16; //求得字模偏移地址
     ReadFontLib(addr,buffer); //获取字模，汉字的字模占用 32 字节
     t++;  //字符串指针后移一个，跳过该汉字
     for(i=0;i<2;i++)
      for(j=0;j<16;j++)
         SendData(seg+xx+j,page+i,buffer[i*16+j]); //将字模数据送 LCD
      xx+=16; //水平后移 16 个点
     }
    else
     {//否则为 ASCII 字符，每个 ASCII 码一个字节
     addr=(long)str[t]*16; //在 ASCII 字库里计算偏移很简单
     ReadFontLib(addr,buffer);
      for(i=0;i<2;i++)
      for(j=0;j<8;j++)
         SendData(seg+xx+j,page+i,buffer[i*8+j]);
      xx+=8;
     }
  }
}
```

在LCD汉字显示函数LCD_printch()中调用ReadFontLib()函数根据字模数据在汉字库中的字节偏移量从汉字库文件中读取 32 字节到给定缓冲区中。

为保护作者权益，并不免费公开提供 ReadFontLib()函数的源代码，仅提供包含该函数的库文件。该文件同样能正确支持 FAT16，从 Flash 中的汉字库文件读取所需的字模数据。在 LCD_init()函数初始化 LCD 时，调用 InitCHFontLib()函数根据汉字库文件名初始化 ReadFontLib()函数所需的参数。

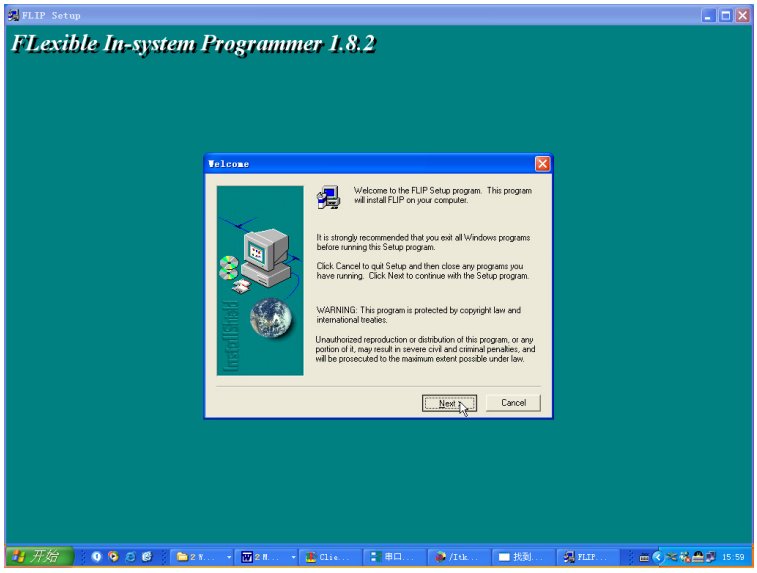
三. 开发工具软件使用说明

在 AT89C51SND1 的 64Kflash 中，其中地址从 F000～FFFF 的区域为 4K 字节的 Boot Flash，这部分程序是出厂时已经固化的 Boot Loader 代码，它使用 DFU（Device Fireware Update：设备固件升级）协议来更新芯片 FLASH 存储器中的用户程序。有两种方式可以使得芯片在上电复位后执行 Boot Flash 中的这段程序，一是当芯片中的 BLJB 位为 1 时，上电即执行这段代码。当 BLJB 位不为 1 时，如果在复位后芯片检测到引脚 ISP 为低时，也会执行这段程序。芯片出厂时为第一种状态，上电后即执行 Boot Loader，通过 USB 接口，可对 FLASH 进行在线编程（ISP：In-System Programming）。这样，不需要硬件烧写器便可下载目标代码到芯片，降低了芯片开发的投资。

@FLIP 软件安装

步骤 1、启动安装向导。

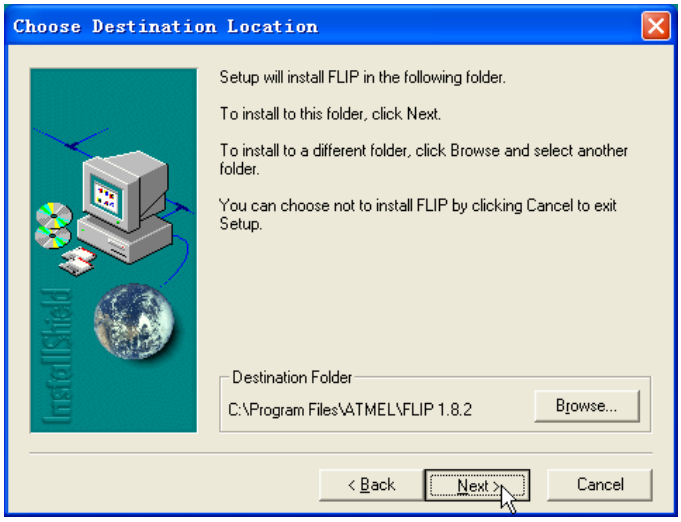
ATMEL 公司为了配合通过 USB 进行在线编程下载，提供了 PC 端的 USB 驱动程序 USB_DFU.INF 和应用程序 FLIP。从 ATMEL 公司的主页上下载 FLIP 软件包后，将其解压至磁盘上，双击其中的 SETUP.EXE 程序进行安装，FLIP 的安装和普通 Windows 应用程序的安装类似。光盘里提供了 FLIP 软件的安装版和 USB 驱动程序。



安装 FLIP

步骤 2、选择安装路径。

可以将 FLIP 安装在任意位置，但请记下这个位置，因为安装向导同时将 USB 驱动程序也安装到了 FLIP 软件的目录下，这个驱动程序在后面会用到。



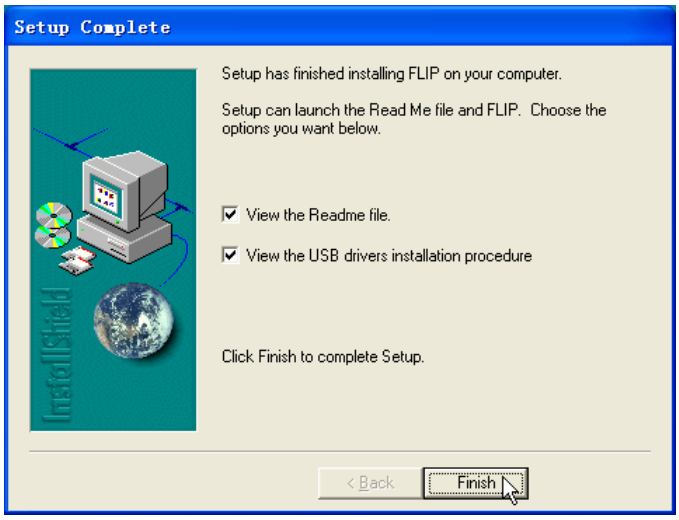
驱动程序及应用程序的安装路径

步骤 3、完成安装。

安装过程顺利完成，需要提醒的是，结束画面上对于 USB 驱动程序的安装过程进行了描述，如果选择“View the USB drivers installation procedure”选项，则单击“完成”后，会显示驱动程序安装的注意事项：

（1）运行程序组中 ATMEL 程序组中的“Install USB Drivers”项。

（2）插入 USB 设备进行驱动程序安装。

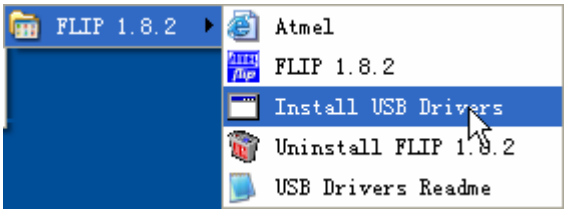


完成软件安装

@USB－DFU 驱动安装

步骤 1、运行“Install USB Drivers”。

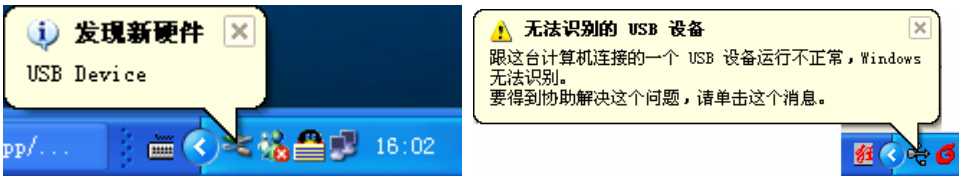
按照 FLIP 安装完毕时的提示，首先在程序组中找到 ATMEL\FLIP 1.8.2，执行其中的 Install USB Drivers。此步骤不是必须的。



运行“Install USB Drivers”

步骤 2、插入设备。

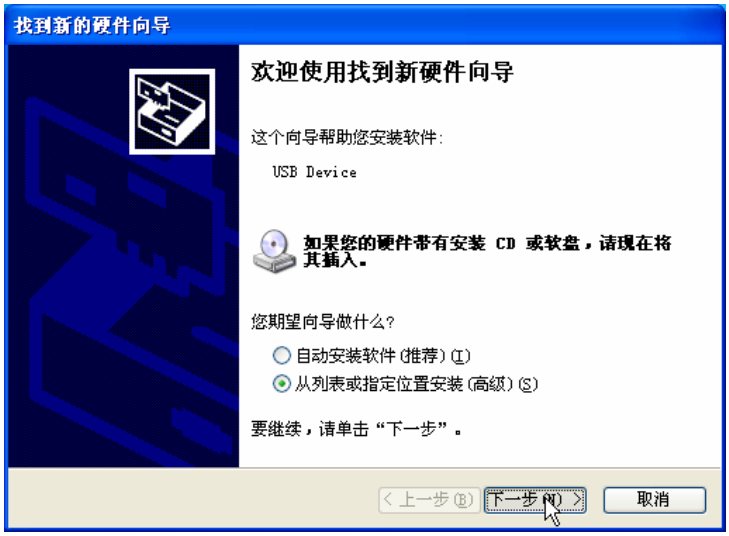
将焊接完的硬件电路板的 USB 接口插入计算机上的扁平 USB 接口中（通过 USB 延长线或直接插入）。如果 Boot Flash 中的内容得到了正确执行，PC 桌面的右下角系统状态栏中将会出现下图左图所示的提示，表明有新设备插入 USB 总线。如果 Boot Flash 中的代码没有正确执行，则出现下图右图的提示将是“无法识别的 USB 设备”。这时，应该立即将 MP3 从 USB 接口上拔下进行必要检查。没有发热现象可以带电，请测试晶振是否起振，电压等是否正常。



插入 USB 口

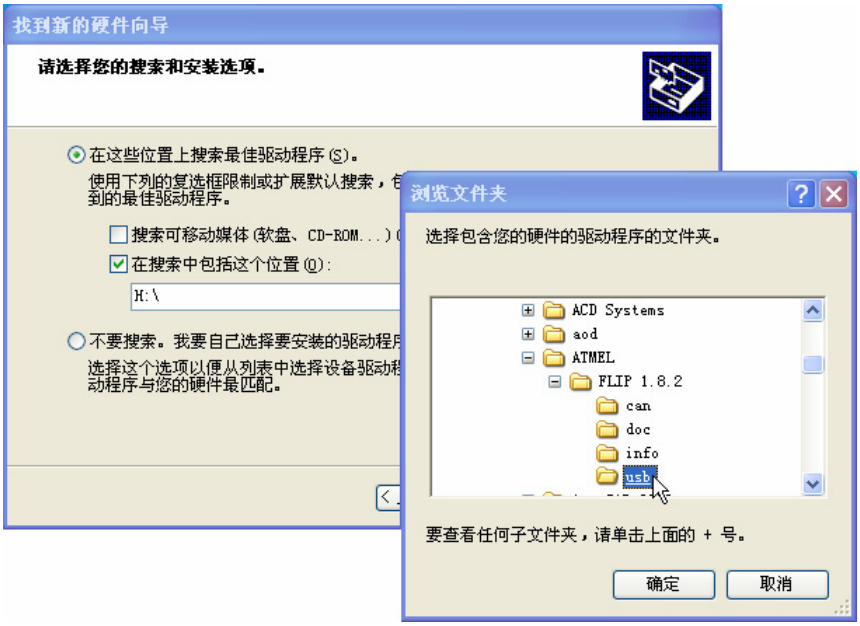
步骤 3、安装驱动程序。

过数秒后，会出现下图所示的提示，PC 无法识别此设备，要求安装设备的 USB 驱动程序。以下的安装驱动程序的方式跟我们在电脑上安装其他类型的硬件设备的方法一致，比如新装的网卡。以下的安装过程也只是一个典型过程，在不同的平台，不同的系统版本上可能会有所不同，但只要按照提示进行安装即可。选择 “从列表或指定位置安装”，并单击“下一步”。



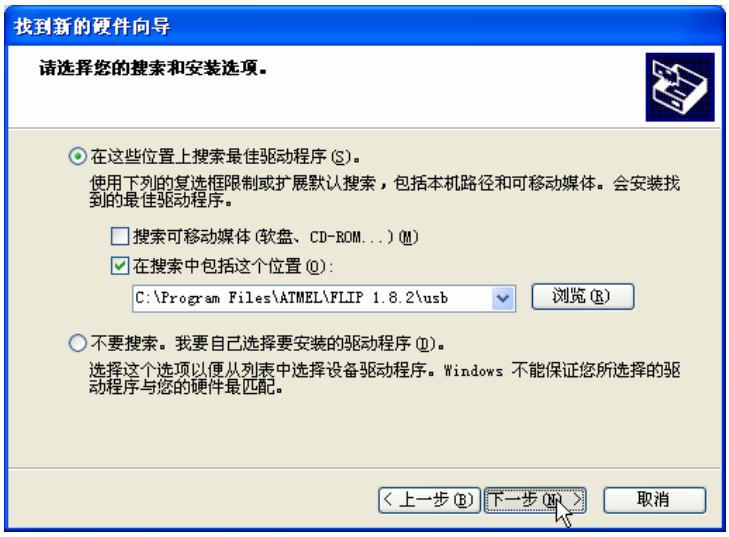
硬件驱动程序安装向导

在下图所示的“选择搜索和安装选项”页面中，选择“在搜索中包括这个位置”，并单击其右方的“浏览”按钮，从“浏览文件夹”对话框中选择刚才安装 FLIP 的那个目录，并选择其下的 USB 目录，然后单击“浏览文件夹”对话框中的“确定”按钮。



选择驱动程序位置

接下来继续单击向导中的“下一步”按钮。



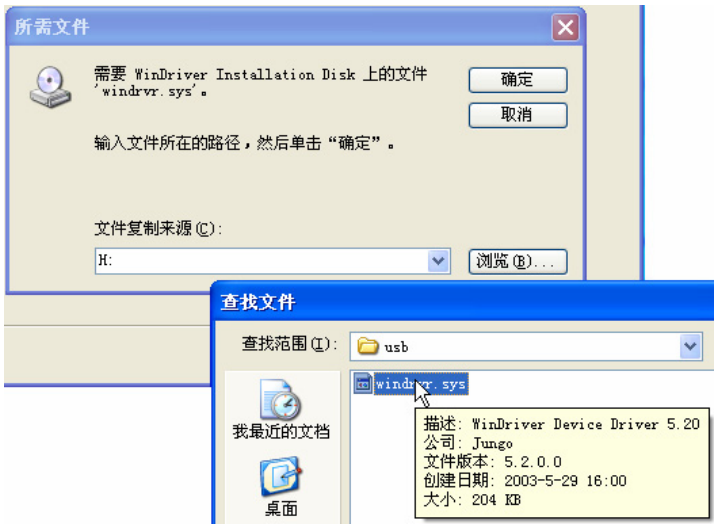
继续安装过程

数秒后，系统会提示已经找到设备 USB_DFU，并开始复制驱动程序文件。

USB_DFU 安装完成后，系统还提示安装与 WinDriver 有关的一个组件，出现下图所示的对话框。此时，单击“确定”，继续安装过程。

提示 WinDriver 组件的安装

在下图所示的对话框中，选择“浏览”按钮，再次找到安装 FLIP 的目录，浏览到其中的 USB，双击其中的 Windriver.sys 文件，再单击“确定”。



查找 windriver.sys

最后，完成驱动程序的安装，单击“完成”结束安装向导。

驱动程序成功安装

@设备连接与固件下载－FLIP 的使用

步骤 1、选择设备类型

运行程序组中的“FLIP 1.8.2”程序项，进入下图所示的 FLIP 主界面。FLIP 支持多个类型的芯片，首先单击图中的“器件选择”图标，在“Device Seletion”对话框中选择器件。

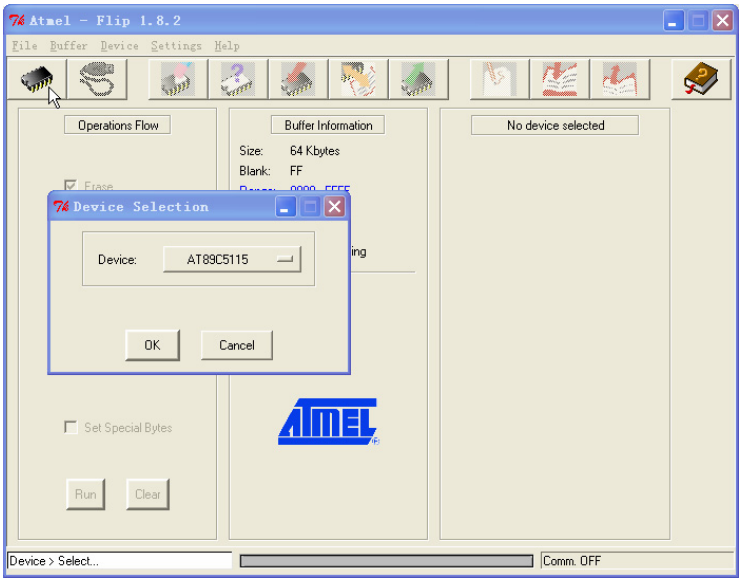
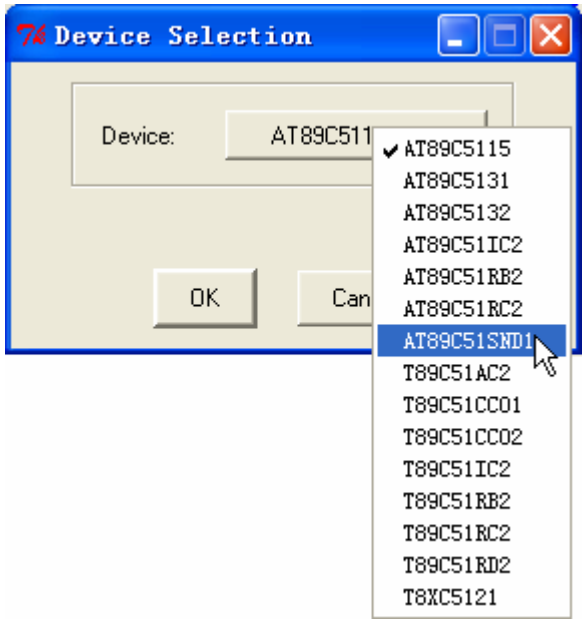


图 I-13　FLIP 主界面

在下图所示的对话框中，单击 “Device”右边的选项，从弹出的菜单中选择“AT89C51SND1”，然后单击“OK”按钮。



选择设备类型

步骤 2、连接设备。

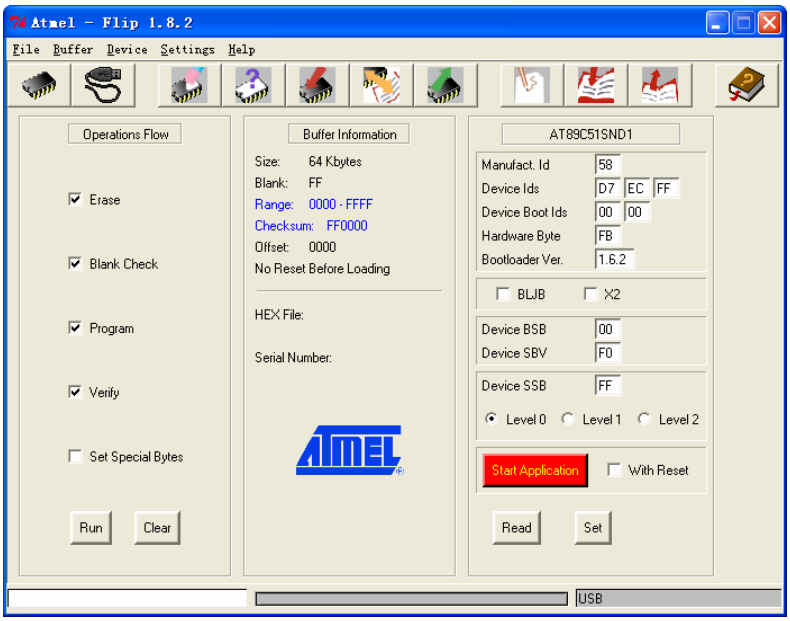
如下图所示，从 Set Communication（设置通讯）中选择“USB”选项。然后单击“USB Port Communication”对话框中的“Open”按钮。

选择设备连接方式

如果出现下图所示的“无法连接设备”警告，请确认驱动程序是否安装正确，检查 USB 电缆，有些机箱前置的 USB 接口并不稳定，USB 电缆太长也可能不稳定。

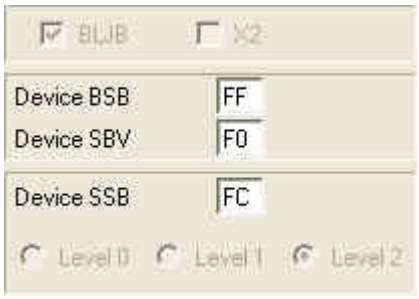
无法连接

如果正确连接，则可以在主界面上看到下图所示情形。主界面右边显示出芯片类型以及与此芯片有关的特殊位设置。



连接正确

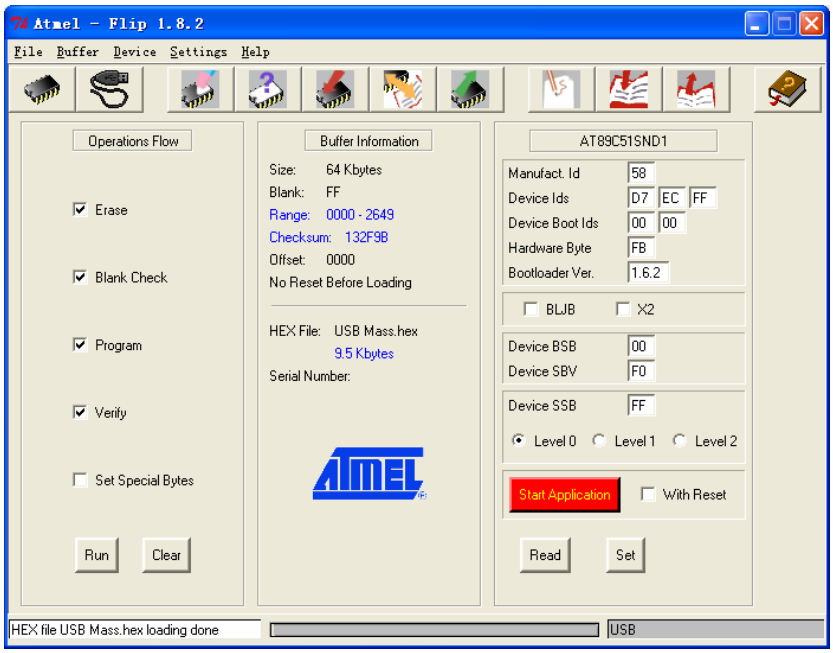
芯片的出厂状态可能随其生产批次而不一样。如果成功连接设备后，出现下图那样的情况，即 BLJB 项为不可选状态时，这种状态下，是因为对芯片进行了加密保护（图中 Level 2 处于选择状态）。必须执行一次 Erase 操作，重置芯片。否则，可能导致代码无法写入芯片。



芯片被设置为 Level 2 加密方式

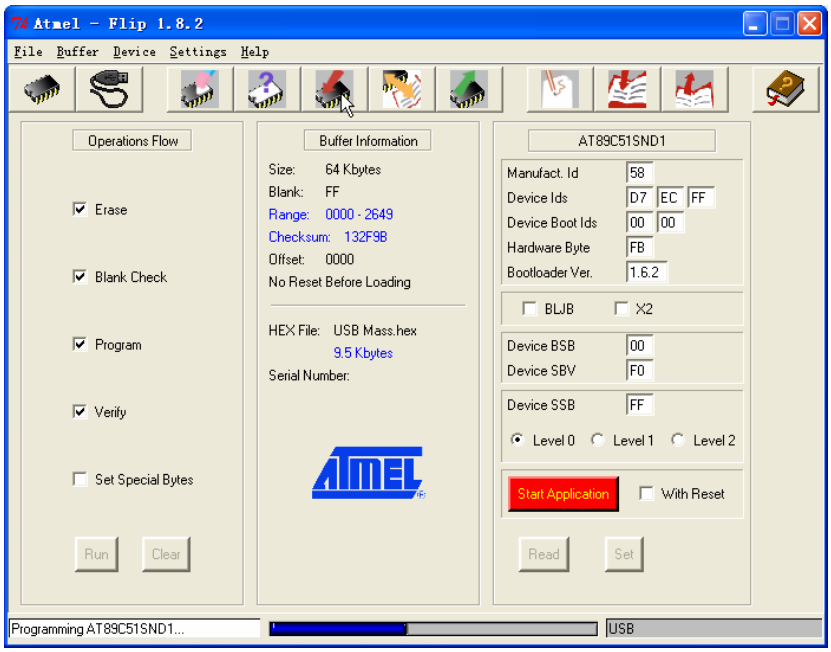
接下来，可以下载由编译器生成的扩展名为“HEX”的目标代码文件了。从 FLIP 主界面上的菜单中选择“FILE / LOAD HEX FILE”，打开自己编译好的 HEX 文件即可。

如果所选择的代码文件格式正确，则 FLIP 主界面的 Buffer Information 中会显示出此文件的长度信息，表明其将占用的 FLASH 存储空间的大小。



完成 HEX 文件载入

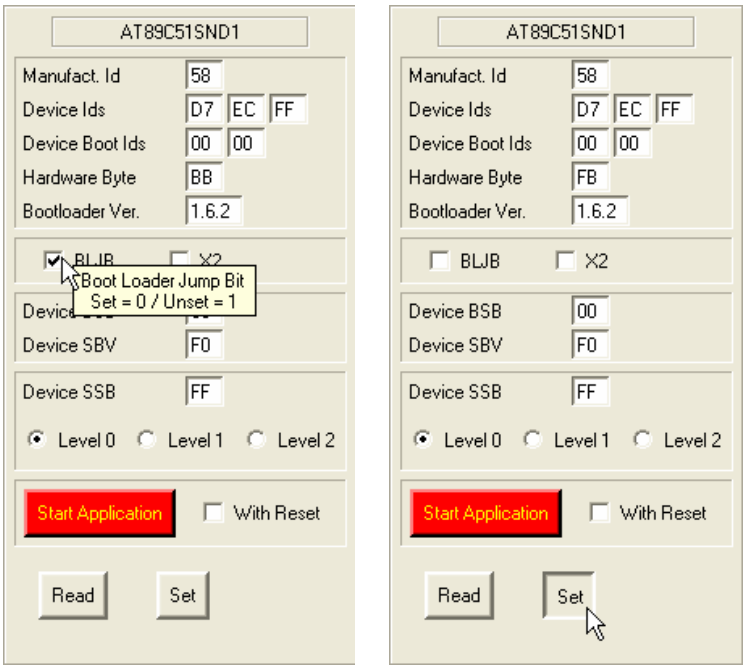
接下来，可以单击工具图标中的“Write（写入）”图标，将刚刚调入的 HEX 数据写入 FLASH 中。也可以点击界面左下角的 Run 按钮，顺序执行上面打了勾的一系列步骤。



将数据定入 FLASH

数据写入完毕以后，可以直接单击主界面右边的醒目的“Start Application”按钮，以使 AT89C51SND1 执行刚刚下载的程序。

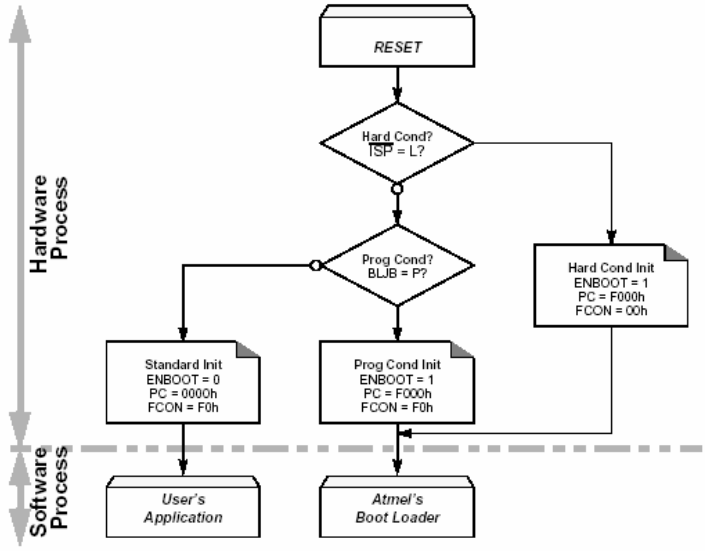
也可以单击 BLJB 选项，将其上的选中标记去掉，然后单击“Set”按钮将其保存到芯片中。



改变 BLJB 位

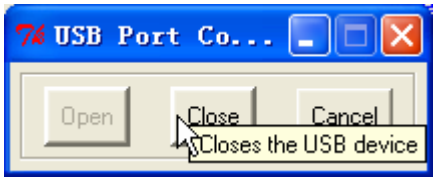
下次设备只要一接通电源，即执行刚刚所下载的用户程序，而不是执行 Boot Loader（因为此时 ISP 没有接地，BLJB 也没有选中）。在这种情况下，如果想再次执行 Boot Flash 进行下载，则需要保证上电复位后芯片的 ISP 引脚为低。关于这点在光盘中其他文档中也有详细叙述。

下图给出 AT89C51SND1 的启动过程，可供参考。或者阅读光盘中 AT89C51SND1 芯片文档，其中有详尽的分析和解释。



硬件启动过程算法流程图

完成对 FLASH 及芯片特殊位的操作后，可以通过下图所示“USB Port Communication”中的“Close”按钮可以安全关闭连接。或者也可以直接断开 USB 连接电缆，但不推荐这种方式，特别是在对 FLASH 进行操作的过程中不要这样做。



断开设备连接

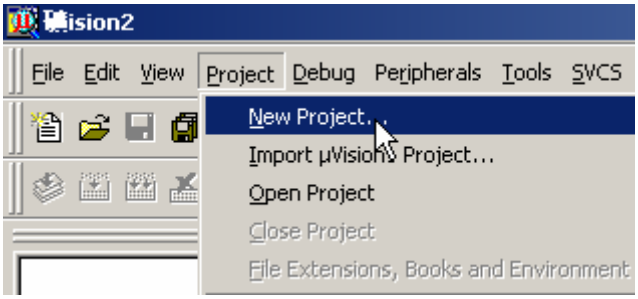
至此，已经可以成功下载代码。代码下载后，有两种方式可以执行之。一种是直接单击 FLIP 界面上的红色“Start Application”按钮。另外一种方式是既不使 ISP 引脚为低，也不选中 BLJB，即直接上电后执行用户程序。结合调试过程中的特点，这两种运行方式分别适用于不同的场合。对于非 USB 通讯程序的调试，一般使用前一种方法，即使 BLJB 选中，这样，每次插入 USB 接口时，总是执行 Boot Loader，于是可以进行程序下载，下载完了以后，单击“Start Application”按钮即可执行。对于 USB 通讯程序，必须一插入 USB 口就执行，因此，采用后一种方法，即 BLJB 不选中，通过按与不按连接在 ISP 上的按钮来决定执行那段程序，按下时，执行 Boot Loader，不按时，执行所下载的 USB 通讯程序。

@Keil 51 的安装

Keil C 软件的安装方法与注意事项，请参考 Keil C 的安装说明文件进行。

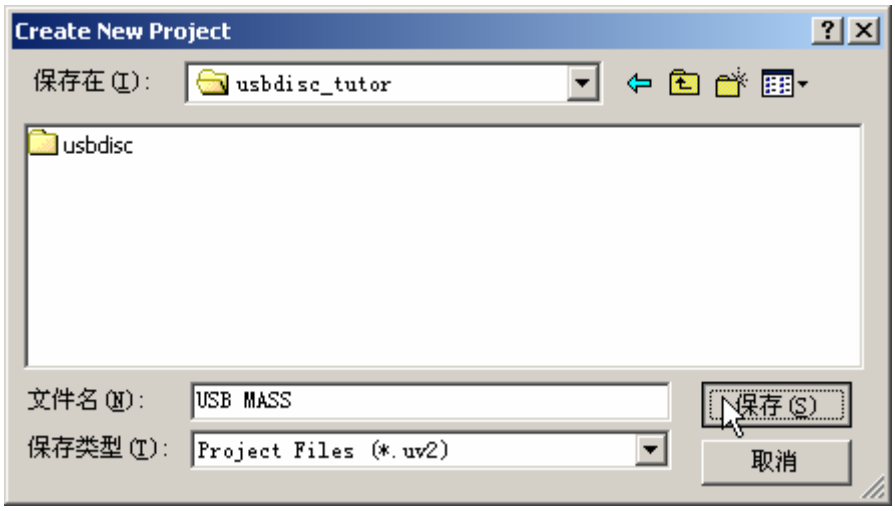
@工程创建与管理

软件安装完成以后，运行程序，如果首次运行，主程序中将不显示任何内容。从菜单中选择“Project / New Project”创建一个工程。



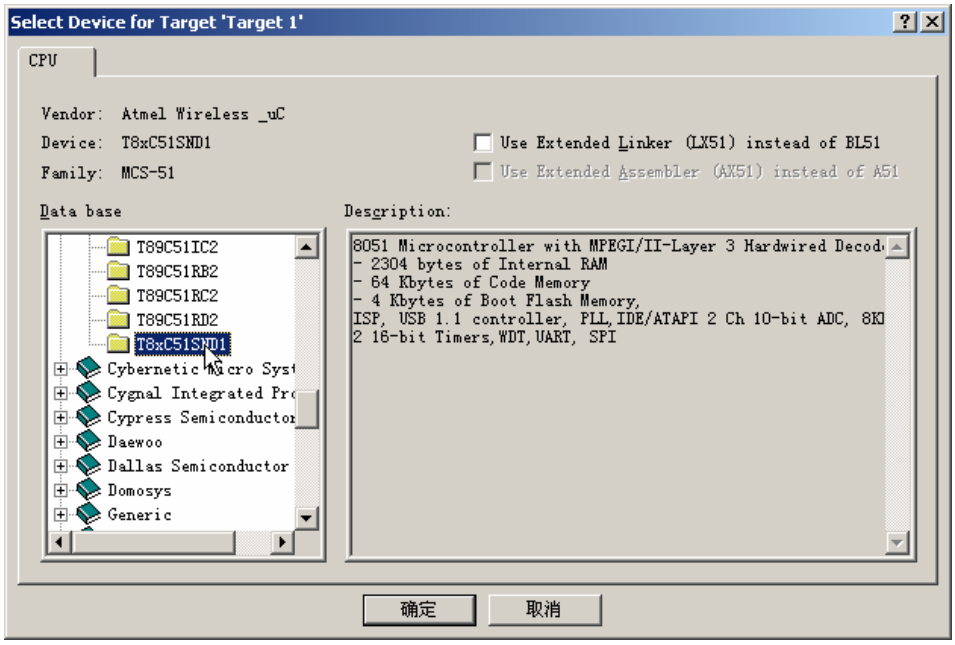
新建工程

在下图的对话框中，为新工程选择文件夹，并为新工程命名。



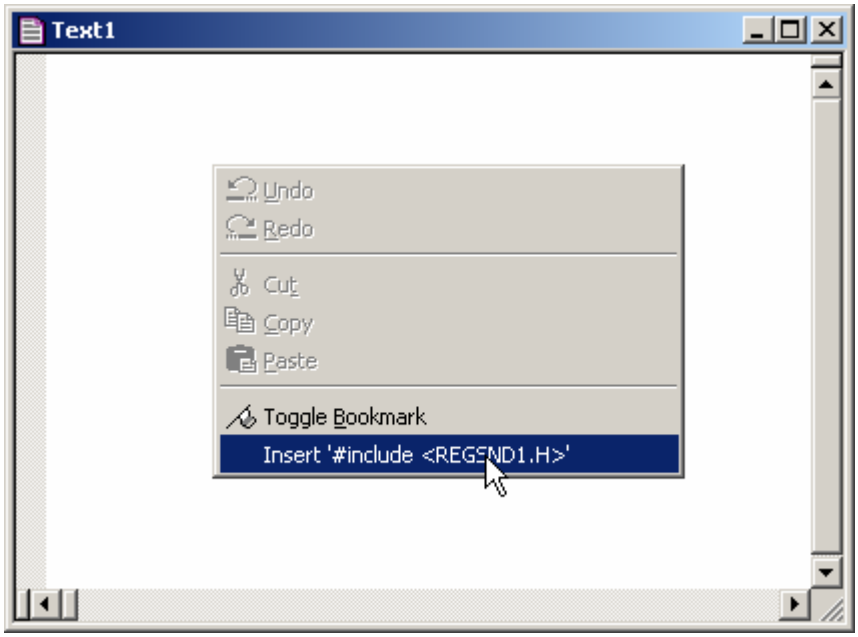
选择新工程路径并命名

Keil 根据不同的器件类型，可以在编译过程中对 RAM 等资源进行控制，因此，需要选择正确的器件类型。单击菜单“Project/Select Deviece for Target…”弹出如下图所示的对话框，在 Data base 中，打开“Atmel Wireless & uC”目录，选择其最后一项“T8xC51SND1”，并单击“确定”。



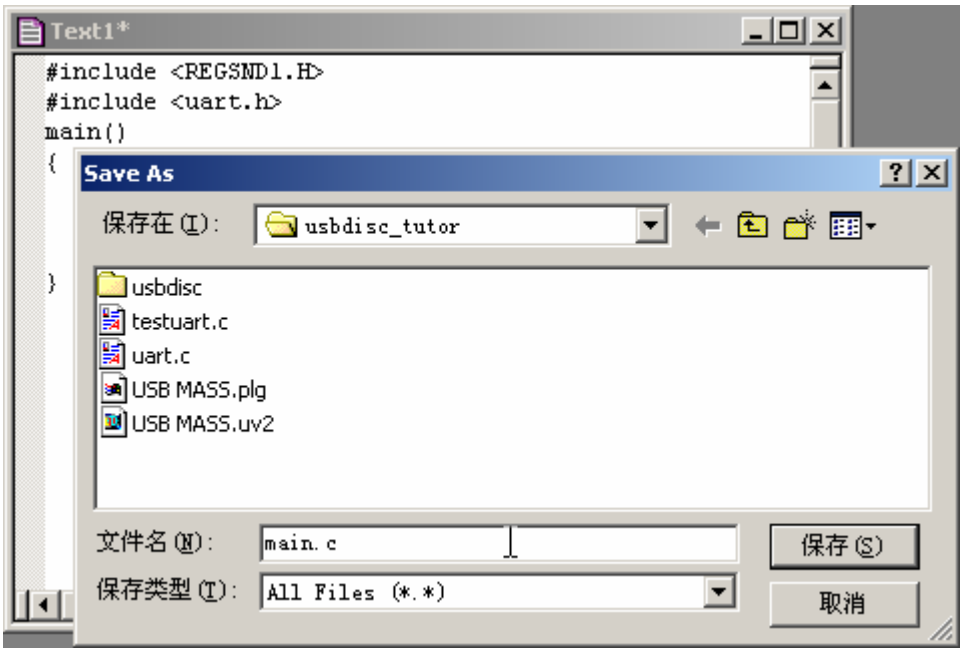
选择器件类型

工程创建完成以后，单击菜单中的“File / New”可以新建文件。如下图所示，在新文件中单击右键，从弹出菜单中选择“Insert ‘#include<REGSND1.H>’”，将与工程中所选器件对应的头文件加入文件中，此头文件中包含了此芯片的的寄存器定义。



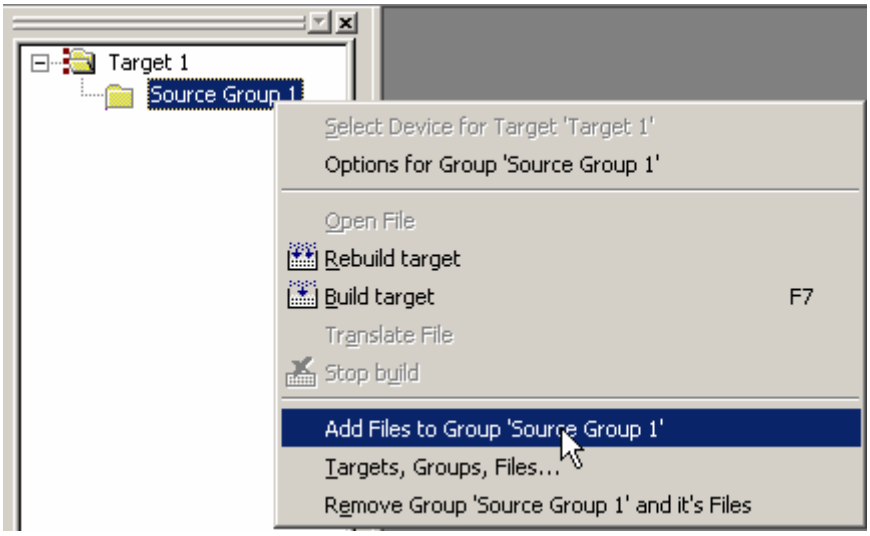
加入头文件

在文件中键入代码后，单击菜单中的“File / Save As”可以保存文件，尽量将文件保存在与创建工程的相同目录下，如图 II-5 所示。



保存文件

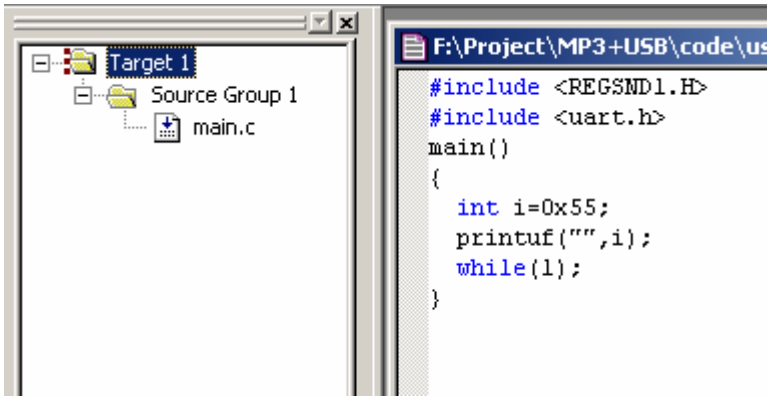
如下图所示，新工程创建后，工程工作区中即有 Target1 和 Sorce Group1，但其中没有文件。在“Project Workspace（工程工作区）”中选中 Source Group1，然后单击鼠标右键，从弹出菜单中选择“Add Files to Group “Source Group 1””，可以向工程中添加源文件。



向工程中添加文件

再从文件浏览窗口选择要添加的文件，双击文件即可将文件加入到 Source Group1 中，也可以选中文件后单击“Add”按钮添加文件。

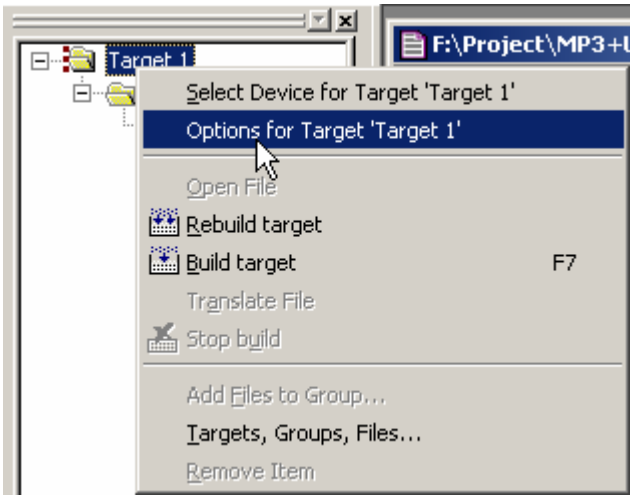
可以多次选择文件，将其添加到工程中，程序中有调用关系的所有文件必须都添加到工程中，Keil 才可以正确编译和链接。



文件添加完毕

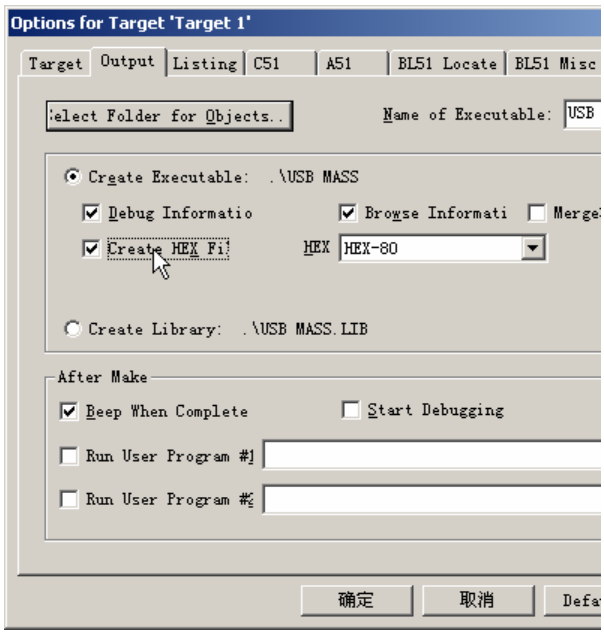
工程创建完毕以后，就可以在工程环境中修改代码，代码修改完毕后，需单击工具栏中的“保存全部”按钮保存代码。然后可以对代码进行编译和链接。

在编译工程之前，还需要对工程进行设置。如下图所示，先用鼠标在工程工作区中选择“Target 1”，然后再单击右键，在弹出菜单中选择“Options for Target ‘Target 1’”，以设置目标选项。



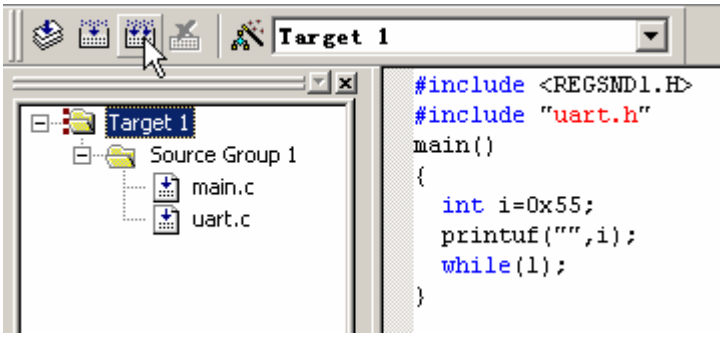
目标选项

在下图所示的 Target 1 目标选项对话框中，切换到“Output”选项卡，选中其中的“Create HEX File”选项，即创建可执行代码。



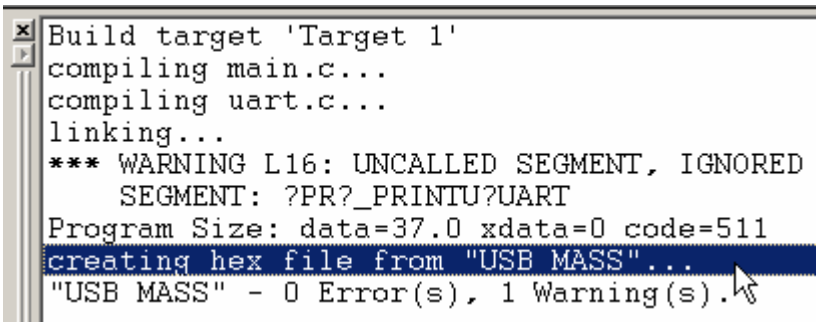
设置选项

设置完成以后，单击 Keil 工具栏上的“Rebuilt All Target File（重新编译所有目标文件）”按钮，对当前目标中的所有文件进行编译。



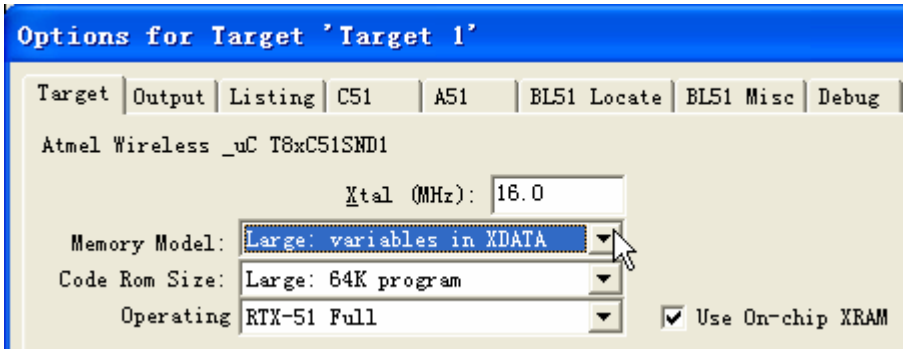
编译目标文件

对于编译结果，如果有错，可根据其提示对工程文件的某行中的错误进行修改。如果代码没有错误，则会生成相应的 HEX 文件。只要显示“creating hex file from”USB MASS”” 信息，即表示 USB MASS.HEX 文件已经成功创建，文件位于工程所在的目录中。



成功创建可执行代码

如果在编译过程中出现“‘DATA’：SEGMENT TOO LARGE”的错误，则说明需要调整存储器模式。按照下图所示的工程选项对话框中设置存储器有关选项。



设置存储器选项

有了 HEX 文件，就可以通过 FLIP 下载工具，将其下载到 AT89C51SND1 芯片的 FLASH 中，再执行此代码，通过向串口输出信息，可以得到固件运行的情况，根据执行结果再次修改代码，直至得到正确结果为至。