目 录

第 1 章 ADS 集成开发环境及 EasyJTAG 仿真器应用 2
1.1 ADS 1.2 集成开发环境的组成 2
1.1.1 CodeWarrior IDE 简介 2
1.1.2 AXD 调试器简介 3
1.2 工程的编辑 4
1.2.1 建立工程 4
1.2.2 建立文件 5
1.2.3 添加文件到工程 5
1.2.4 编译连接工程 6
1.2.5 打开旧工程 8
1.3 工程的调试 9
1.3.1 选择调试目标 9
1.3.2 调试工具条 9
1.4 LPC2200 系列 ARM7 微控制器工程模板 10
1.4.1 为 ADS1.2 增加 LPC2200 专用工程模板 11
1.4.2 使用 LPC2200 专用工程模板建立工程 11
1.4.3 模板适用范围 13
1.5 EasyJTAG 仿真器的安装与应用 16
1.5.1 安装 EasyJTAG 仿真器 17
1.5.2 使用 EasyJTAG 仿真器 17
1.6 固化程序 19
1.6.1 片内 FLASH 的固化 19
1.6.2 片外 FLASH 的固化 23
第 2 章 基础实验 26
2.1 外部中断实验 2 26
2.2 外部存储器接口实验 2 28
2.3 定时器实验 2 32
2.4 UART 实验 2 35
2.5 I^2C 接口实验 2 40
2.6 SPI 接口实验（选做） 44
2.7 RTC 实验 2 47
2.8 低功耗实验 2 51
第 3 章 基于 μC/OS-II 的基础实验 55
3.1 SPI 总线的 LED 控制应用 55
3.2 I^2C 总线的 EEPROM 应用 58
3.3 I^2C 总线的 ZLG7290 应用 62
3.4 LPC2000 系列微控制器 MODEM 接口软件包 65
3.4.1 概述 65
3.4.2 软件包的使用 66
3.4.3 设计原理 69
第 4 章 ZLG/SD 软件包使用手册 78

4.1 SD/MMC 卡的外部物理接口 78
4.1.1 SD 模式 79
4.1.2 SPI 模式 80
4.2 访问 SD/MMC 卡的 SPI 模式硬件电路设计 81
4.2.1 SPI 总线 82
4.2.2 卡供电控制 82
4.2.3 卡检测电路 82
4.3 ZLG/SD 软件包的文件结构及整体构架 82
4.3.1 ZLG/SD 软件包的文件组成 82
4.3.2 ZLG/SD 软件包整体框架 83
4.4 ZLG/SD 软件包的使用说明 83
4.4.1 ZLG/SD 软件包的硬件配置 83
4.4.2 ZLG/SD 软件包提供的 API 函数 87
第 5 章 ISP1161A1 和 ISP1181B 简介及硬件电路设计 90
5.1 ISP1161A1 简介及硬件电路设计 90
5.1.1 ISP1161A1 管脚描述 91
5.1.2 ISP1161A1 与 LPC2200 的硬件电路设计 93
5.2 ISP1181B 简介及硬件电路设计 94
5.2.1 ISP1181B 管脚描述 95
5.2.2 ISP1181B 与 LPC2200 的硬件电路设计 98
5.2.3 ISP1181B PACK 简介 99
第 6 章 ZLG/USBD13 软件包 101
6.1 ZLG/USBD13 软件包结构 101
6.2 ZLG/USBD13 软件包的使用方法 101
6.2.1 ZLG/USBD13 软件包的硬件配置方法 102
6.2.2 软件包的其它配置 105
6.3 ZLG/USBD13 软件包 API 函数 107
6.4 EasyUSBD13.dll 动态库 110
第 7 章 ZLG 驱动中间件使用说明 111
1.1 ZLG/CF 驱动中间件 111
1.1.1 概述 111
1.1.2 ZLG/CF 驱动的结构视图 111
1.1.3 中间件配置选项 112
1.1.4 API 函数手册 113
第 8 章 读写 U 盘模块驱动的使用 120
8.1 读写 U 盘模块简介 120
8.2 固件驱动说明 120
8.2.1 初始化配置 120
8.2.2 用户编写的接口函数 121
8.2.3 API 函数 124
第 9 章 MiniGUI 图形界面实验 127
9.1 MiniGUI for uC/OS-II 移植实验 127
9.2 MiniGUI 消息处理实验 147
9.3 对话框应用编程实验 153

9.4 控件应用编程实验.....162
9.5 自定义控件实验.....170
9.6 简易编辑器实验.....175
9.7 位图显示实验.....181
9.8 GDI 绘图实验.....186
9.9 桌面主题实验.....191
第 10 章 μCLinux 开发平台构建.....202
10.1 宿主机和目标机.....202
10.2 建立交叉开发环境.....202
10.2.1 建立开发环境.....202
10.2.2 安装μCLinux.....203
10.2.3 配置μCLinux.....203
10.3 下载μCLinux 内核到开发板.....206
10.3.1 文件结构.....206
10.3.2 下载 ZLG_BOOT.....208
10.3.3 下载μCLinux 内核和文件系统.....213
10.4 启动μCLinux.....221
10.4.1 在 Widows 下通过超级终端进行.....221
10.4.2 在 Linux 下通过 Minicom 进行.....223
10.5 NFS 设置.....224
10.6 GDB 调试.....228
第 11 章 基于μCLinux 基础实验.....229
11.1 CAT1025 读写实验.....229
11.2 任务定时器.....231
11.3 使用信号量作定时器.....234
11.4 多进程实验.....236
11.5 WebServer 实验.....239

μC/OS-II 篇

第1章 ADS 集成开发环境及 EasyJTAG 仿真器应用

ADS 集成开发环境是 ARM 公司推出的 ARM 核微控制器集成开发工具，英文全称为 ARM Developer Suite，成熟版本为 ADS1.2。ADS1.2 支持 ARM10 之前的所有 ARM 系列微控制器，支持软件调试及 JTAG 硬件仿真调试，支持汇编、C、C++源程序，具有编译效率高、系统库功能强等特点，可以在 Windows98、Windows XP、Windows2000 以及 RedHat Linux 上运行。

这里将简单介绍使用 ADS1.2 建立工程，编译连接设置，调试操作等等。最后还介绍了基于 LPC2200 系列 ARM7 微控制器的工程模板的使用，EasyJTAG 仿真器的安装与使用。

1.1 ADS 1.2 集成开发环境的组成

ADS 1.2 由 6 个部分组成，如表 1.1 所示。

表 1.1 ADS 1.2 的组成部分

名称	描述	使用方式
代码生成工具	ARM 汇编器， ARM 的 C、C++编译器， Thumb 的 C、C++编译器， ARM 连接器	由 CodeWarrior IDE 调用
集成开发环境	CodeWarrior IDE	工程管理，编译连接
调试器	AXD， ADW/ADU， armsd	仿真调试
指令模拟器	ARMulator	由 AXD 调用
ARM 开发包	一些底层的例程， 实用程序(如 fromELF)	一些实用程序由 CodeWarrior IDE 调用
ARM 应用库	C、C++函数库等	用户程序使用

由于用户一般直接操作的是 CodeWarrior IDE 集成开发环境和 AXD 调试器，所以这一章我们只介绍这两部分软件的使用，其它部分的详细说明参考 ADS 1.2 的在线帮助文档或相关资料。

1.1.1 CodeWarrior IDE 简介

ADS 1.2 使用了 CodeWarrior IDE 集成开发环境，并集成了 ARM 汇编器、ARM 的 C/C++编译器、Thumb 的 C/C++编译器、ARM 连接器，包含工程管理器、代码生成接口、语法敏感(对关键字以不同颜色显示)编辑器、源文件和类浏览器等等。CodeWarrior IDE 主窗口如图 1.1 所示。

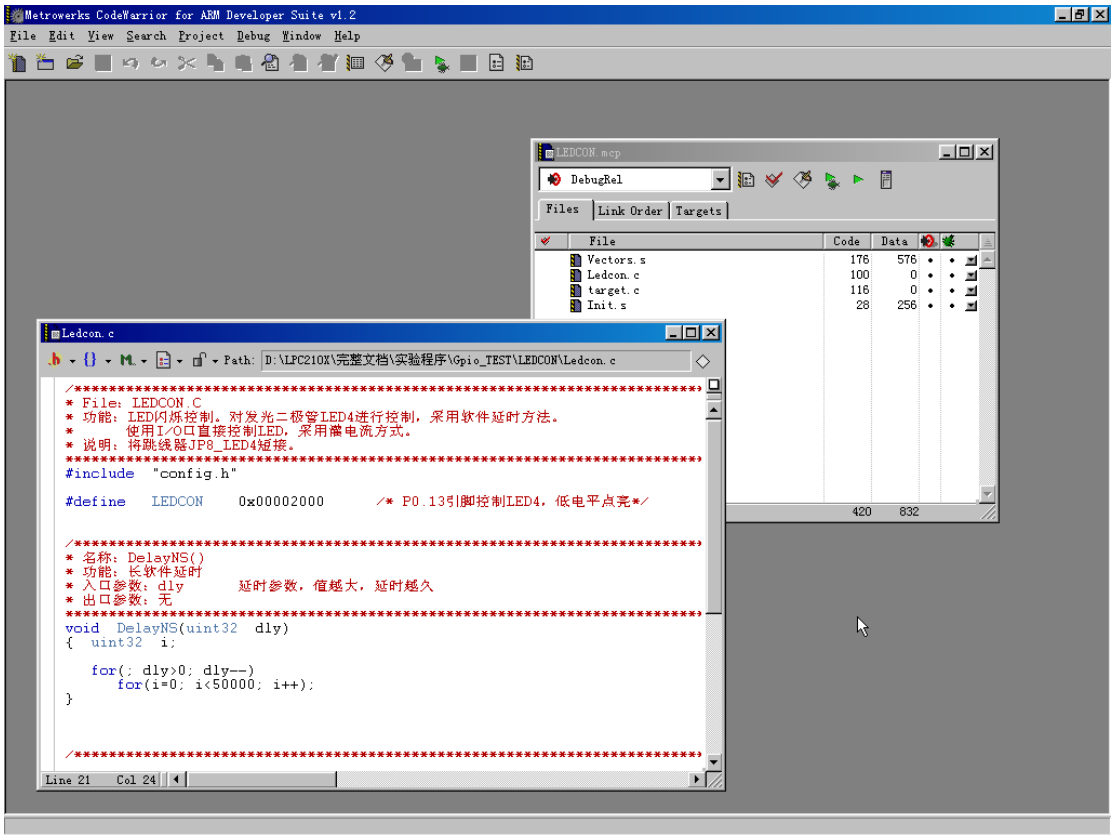


图 1.1 CodeWarrior 开发环境

1.1.2 AXD 调试器简介

AXD 调试器为 ARM 扩展调试器(即 ARM eXtended Debugger)，包括 ADW/ADU 的所有特性，支持硬件仿真和软件仿真(ARMulator)。AXD 能够装载映像文件到目标内存，具有单步、全速和断点等调试功能，可以观察变量、寄存器和内存的数据等等。AXD 调试器主窗口如图 1.2 所示。

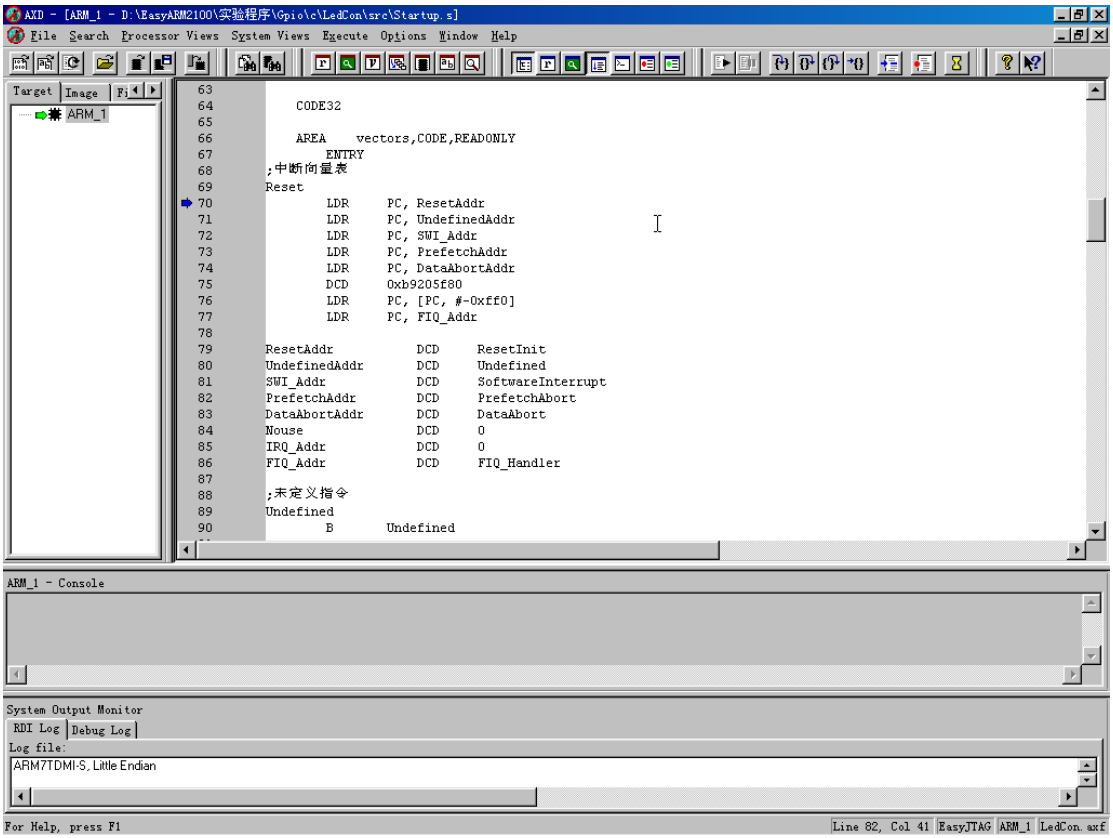


图 1.2 AXD 调试器

1.2 工程的编辑

1.2.1 建立工程

点击 WINDOWS 操作系统的【开始】->【程序】->【ARM Developer Suite v1.2】->【CodeWarrior for ARM Developer Suite】起动 Metrowerks CodeWarrior，或双击“CodeWarrior for ARM Developer Suite”快捷方式起动。启动 ADS1.2 IDE 如图 1.3 所示。

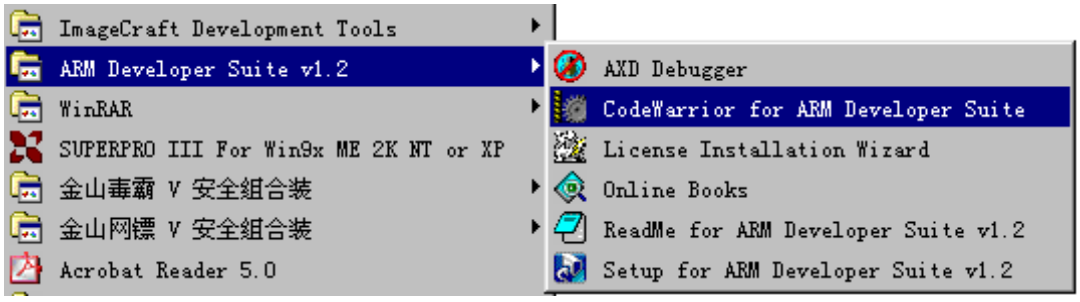


图 1.3 启动 ADS1.2 IDE

点击【File】菜单，选择【New…】即弹出 New 对话框，如图 1.4 所示。

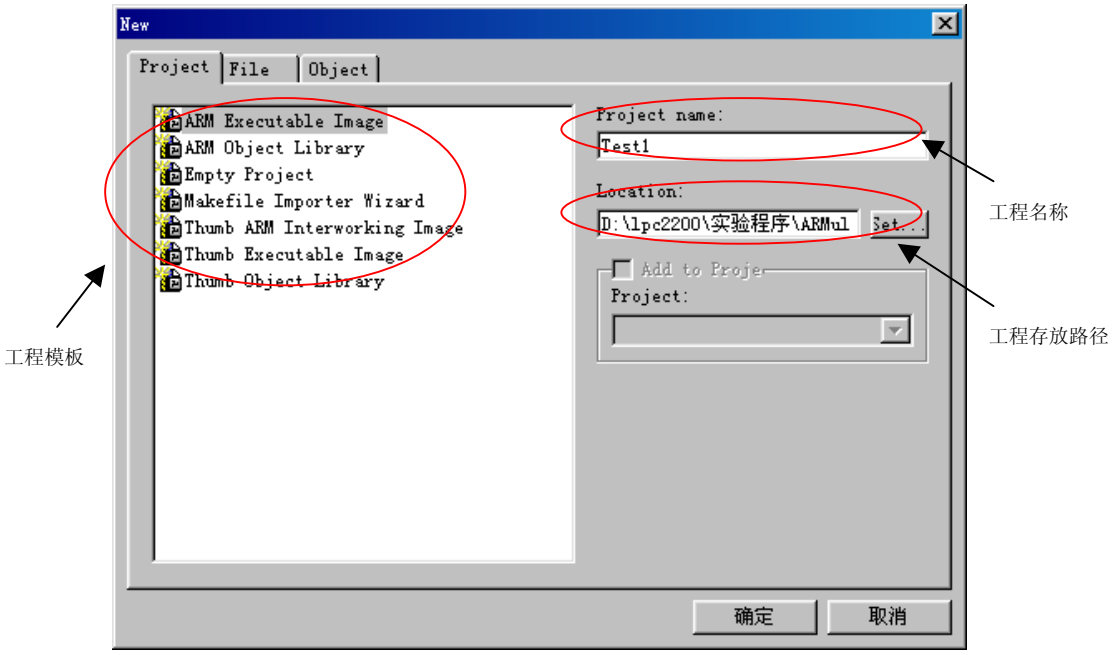


图 1.4 New 对话框

选择工程模板为 ARM 可执行映象（ARM Executable Image）或 Thumb 可执行映象(Thumb Executable Image)，或 Thumb、ARM 交织映象(Thumb ARM Interworking Image)，然后在【Location】项选择工程存放路径，并在【Project name】项输入工程名称，点击【确定】按钮即可建立相应工程，工程文件名后缀为 mcp(下文有时也把工程称为项目) 。

1.2.2 建立文件

建立一个文本文件，以便输入用户程序。点击“New Text File”图标按钮，如图 1.5 所示。

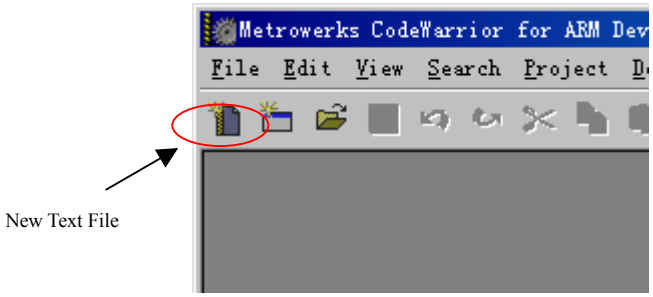


图 1.5 “New Text File”图标按钮

然后在新建的文件中编写程序，点击“Save”图标按钮将文件存盘(或从【File】菜单选择【Save】)，输入文件全名，如 TEST1.S。注意，请将文件保存到相应工程的目录下，以便于管理和查找。

当然，您也可以 New 对话框选择【File】页来建立源文件，如图 1.4 所示，或使用其它文本编辑器建立或编辑源文件。

1.2.3 添加文件到工程

如图 1.6 所示，在工程窗口中【Files】页空白处点击鼠标右键，弹出浮动菜单，选择“Add Files…”即可弹出“Select files to add…”对话框，选择相应的源文件(可按着 Ctrl 键一次选择多个文件)，点击【打开】按钮即可。

另外，用户也可以在【Project】菜单中选择【Add Files…】来添加源文件，或使用 New 对话框选择【File】页来建立源文件时选择加入工程(即选中“Add to Project”项)。添加文件操作如图 1.6、图 1.7 所示。

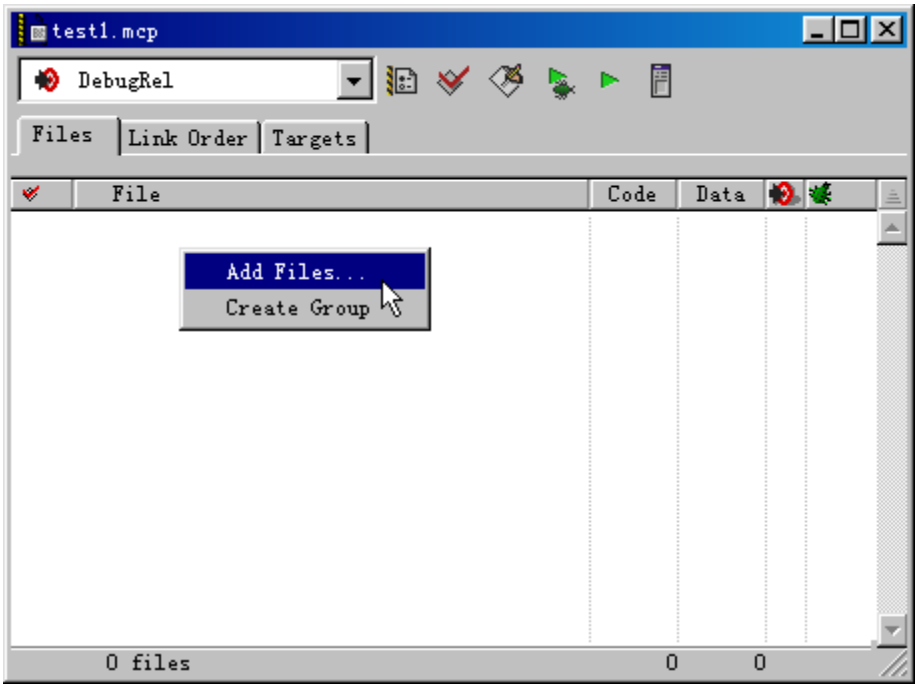


图 1.6　在工程窗口中添加源文件

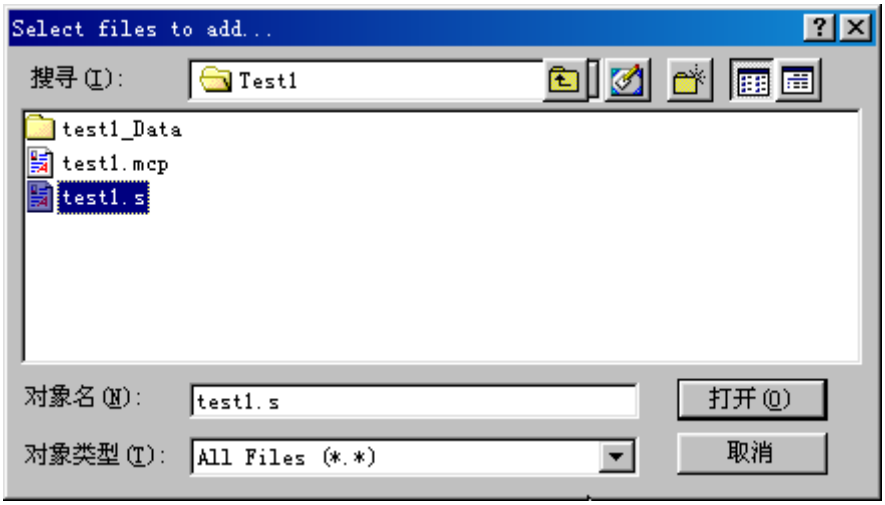


图 1.7　Select files to add…对话框

1.2.4　编译连接工程

如图 1.8 所示为工程窗口中的图标按钮，通过这些图标按钮，您可以快速的进行工程设置、编译连接、启动调试等等(在不同的菜单项上可以分别找到对应的菜单命令)。它们从左至右分别为：

DebugRel Settings…　　**工程设置，如地址设置、输出文件设置、编译选项等，其中 DebugRel 为当前的生成目标(target system)。**

Synchronize Modification Dates	同步修改日期，检查工程中每个文件的修改日期，若发现有更新(如使用其它编辑器编辑源文件)，则在 Touch 栏标记“√”。
Make	**编译连接(快捷键为 F7)。**
Debug	**启动 AXD 进行调试(快捷键为 F5)。**
Run	启动 AXD 进行调试，并直接运行程序。
Project Inspector	工程检查，查看和配置工程中源文件的信息。

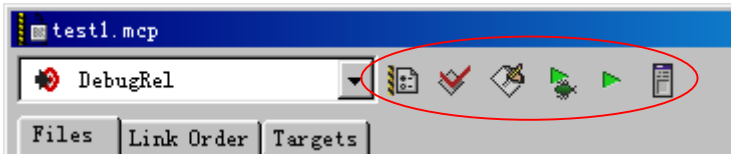


图 1.8 工程窗口中的图标按钮

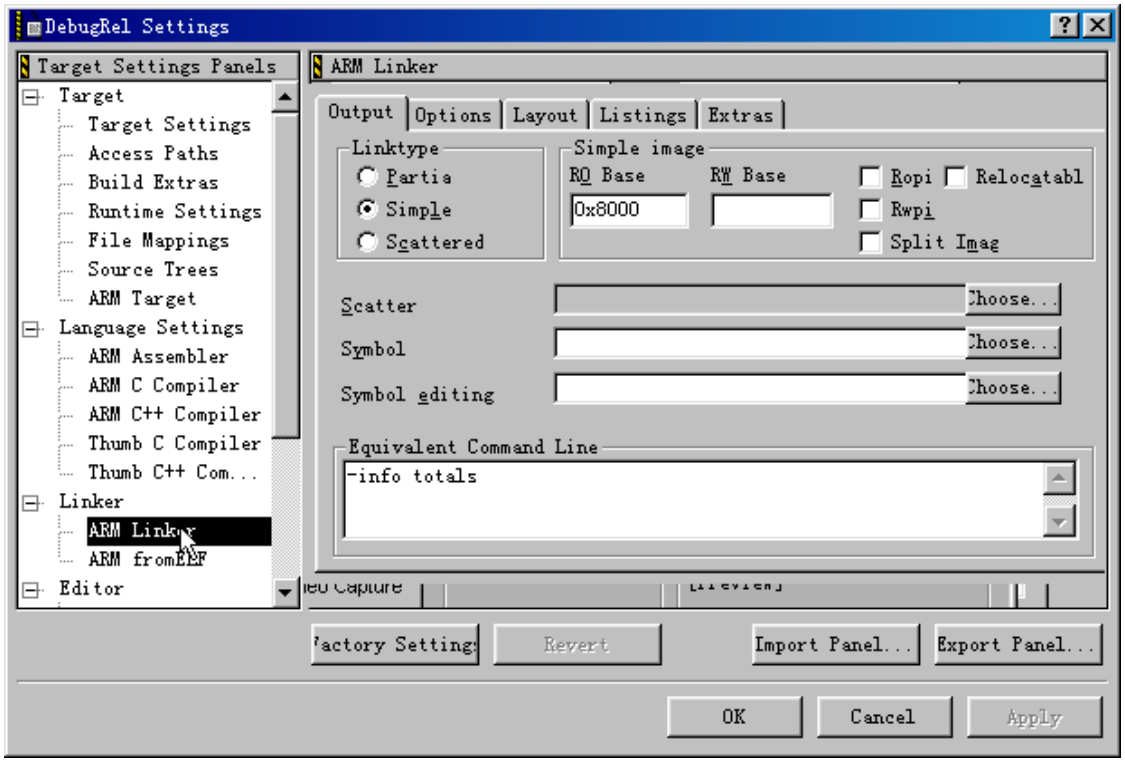


图 1.9 DebugRel Settings 窗口

点击“DebugRel Settings…”图标按钮，即可进行工程的地址设置、输出文件设置、编译选项等，如图 1.9 所示。在“ARM Linker”对话框设置连接地址，在“Language Settings”中设置各编译器的编译选项。

对于简单的软件调试，可以不进行连接地址的设置，直接点击工程窗口的“Make”图标按钮，即可完成编译连接。**若编译出错，会有相应的出错提示，双击出错提示行信息，编辑窗即会使用光标指出当前出错的源代码行**，编译连接输出窗口如图 1.10 所示。同样，您可以在【Project】菜单中找到相应的命令。

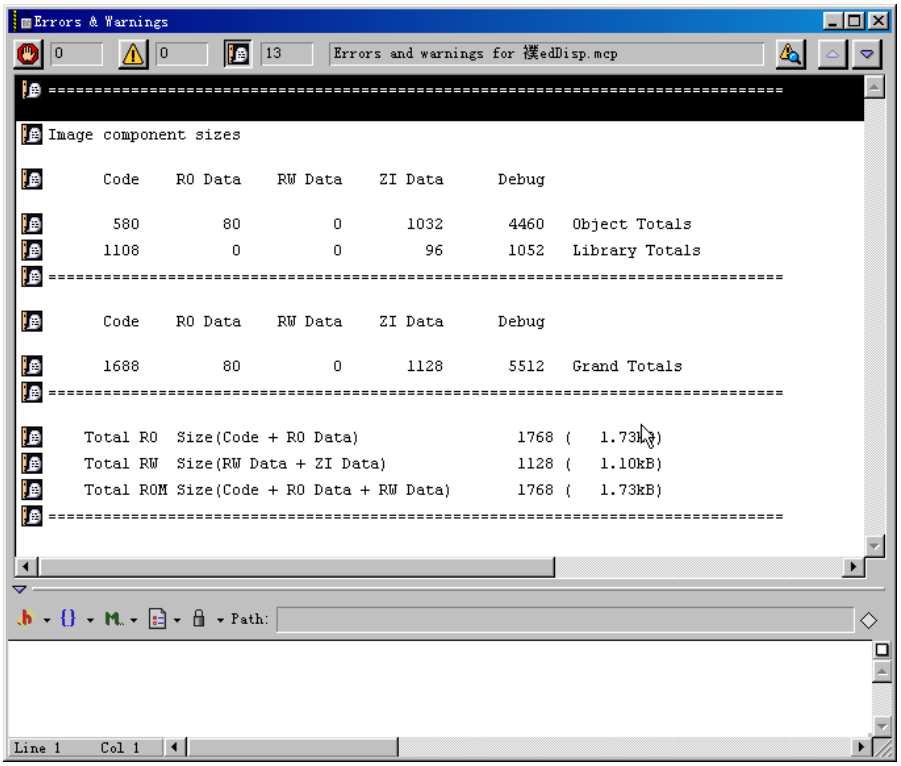


图 1.10 编译连接输出窗口

如图 1.11 所示，Touch 栏用于标记文件是否已编译，若打上“√”则表明对应文件需要重新编译。Touch 栏用于标记文件是否已编译，若打上“√”则表明对应文件需要重新编译。可以通过单击该栏位置来设置/取消符号“√”，或将工程目录下的*.tdt 文件删除也可以使整个工程源文件均打上“√”。

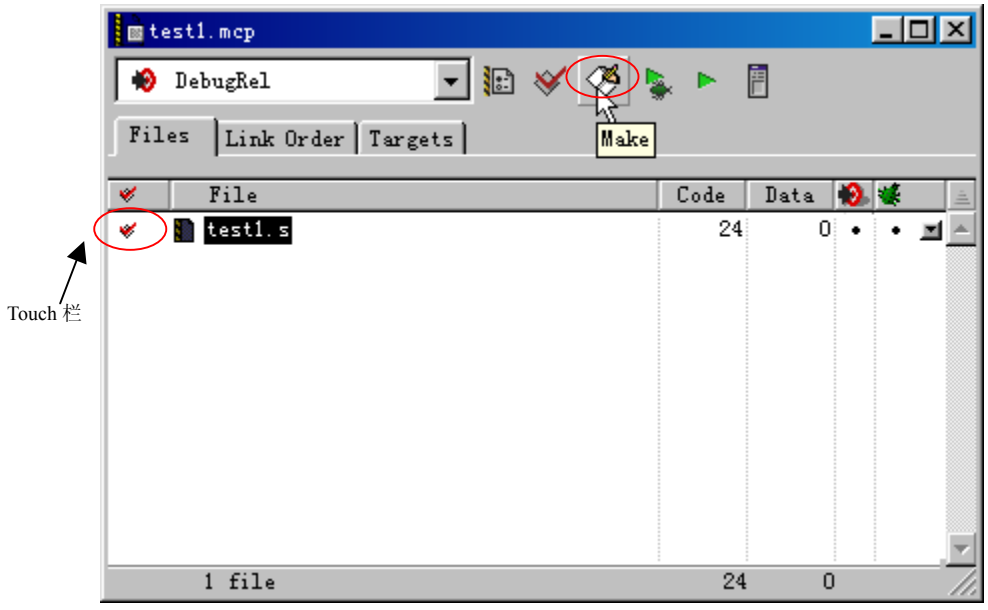


图 1.11 工程窗口中 Make 操作

1.2.5 打开旧工程

点击【File】菜单，选择【Open…】即弹出“打开”对话框，找到相应的工程文件(*.mcp)，

单击【打开】即可。在工程窗口的【Files】页中，双击源程序的文件名即可打开该文件进行编辑。

1.3 工程的调试

1.3.1 选择调试目标

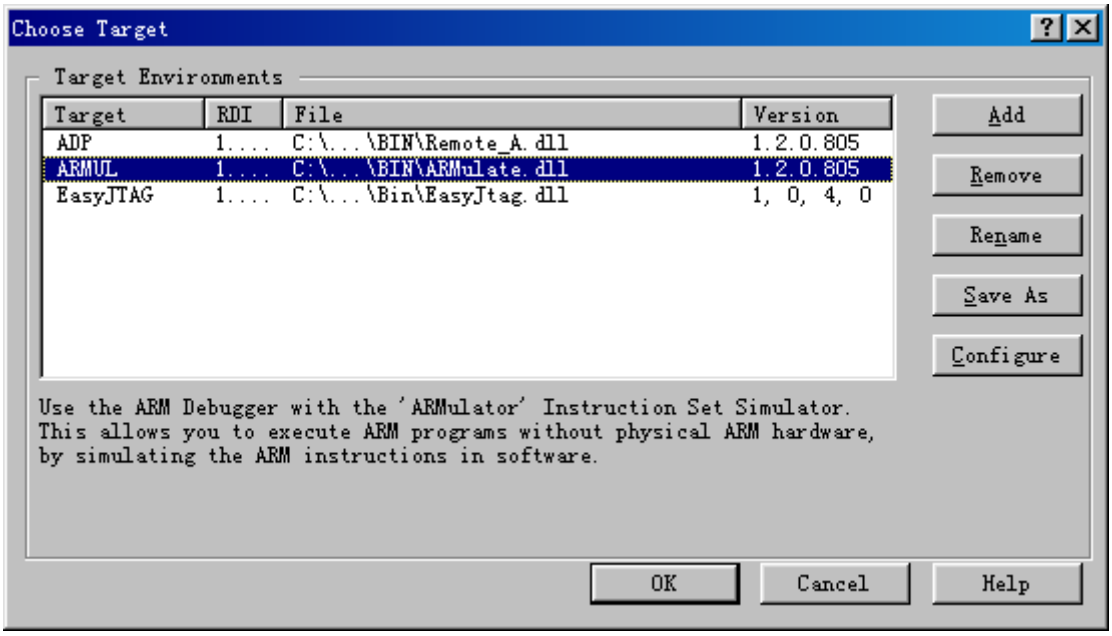


图 1.12 Choose Target 窗口

当工程编译连接通过后，在工程窗口中点击“Debug”图标按钮，即可启动 AXD 进行调试(也可以通过【开始】菜单起动 AXD)。点击菜单【Options】选择【Configure Target…】，即弹出 Choose Target 窗口，如图 1.12 所示。在没有添加其它仿真驱动程序前，Target 项中只有两项，分别为 ADP(JTAG 硬件仿真)和 ARMUL(软件仿真)。

选择仿真驱动程序后，点击【File】选择【Load Image…】加载 ELF 格式的可执行文件，即*.axf 文件。说明：当工程编译连接通过后，在“工程名\工程名_Data\当前的生成目标”目录下就会生成一个*.axf 调试文件。比如工程 TEST，当前的生成目标 Debug，编译连接通过后，则在…\TEST\TEST_Data\Debug 目录下生成 TEST.axf 文件。

1.3.2 调试工具条

AXD 运行调试工具条如图 1.13 所示，调试观察窗口工具条如图 1.14 所示，文件操作工具条如图 1.15 所示。



图 1.13 运行调试工具条

 全速运行(Go)

 停止运行(Stop)

 单步运行(Step In)，与 Step 命令不同之处在于对函数调用语句，Step In 命令将进入

该函数。



单步运行(Step)，每次执行一条语句，这时函数调用将被作为一条语句执行。



单步运行(Step Out)，执行完当前被调用的函数，停止在函数调用的下一条语句。



运行到光标(Run To Cursor)，运行程序直到当前光标所在行时停止。



设置断点(Toggle BreakPoint)



图 1.14 调试观察窗口工具条



打开寄存器窗口(Processor Registers)



打开观察窗口(Processor Watch)



打开变量观察窗口(Context Variable)



打开存储器观察窗口(Memory)



打开反汇编窗口(Disassembly)



图 1.15 文件操作工具条



加载调试文件(Load Image)



重新加载文件(Reload Current Image)。由于 AXD 没有复位命令，所以通常使用 Reload 实现复位(直接更改 PC 寄存器为零也能实现复位)。

1.4 LPC2200 系列 ARM7 微控制器工程模板

在第 1.2 节介绍新建立工程时，我们已经接触了 ADS1.2 提供的几个标准工程模板，使用各个模板建立的工程，它们的各项设置均有不同之处，方便生成不同结构的代码，如 ARM 可执行映象（生成 ARM 指令的代码）或 Thumb 可执行映象(生成 Thumb 指令的代码)，或 Thumb、ARM 交织映象(生成 Thumb、ARM 指令交织的代码)。

针对 LPC2200 系列 ARM7 微控制器，我们定义了 6 个工程模板，这些模板一般包含的设置信息有 FLASH 起始地址 0x00000000、片内 RAM 起始地址 0x40000000、片外 RAM 起始地址为 0x80000000、编译连接选项及编译优化级别等等；模板中包含了 LPC2200 系列 ARM7 微控制器的起动文件，包括 STARTUP.S、TARGET.C；模板还包含了 LPC2200 系列 ARM7 微控制器的头文件(如：LPC2294.h 和 LPC2294.inc，LPC2294 的寄存器是向下兼容的)，分散加载描述文件(如：mem_a.scf、mem_b.scf、mem_c.scf)等等。

1.4.1 为 ADS1.2 增加 LPC2200 专用工程模板

将“lpc2200 project module”目录下的所有文件和目录拷贝到“<ADS1.2 安装目录>\Stationery\”即可，操作如图 1.16 和图 1.17 所示。这个步骤只需 1 次，以后就可以直接使用工程模板了。

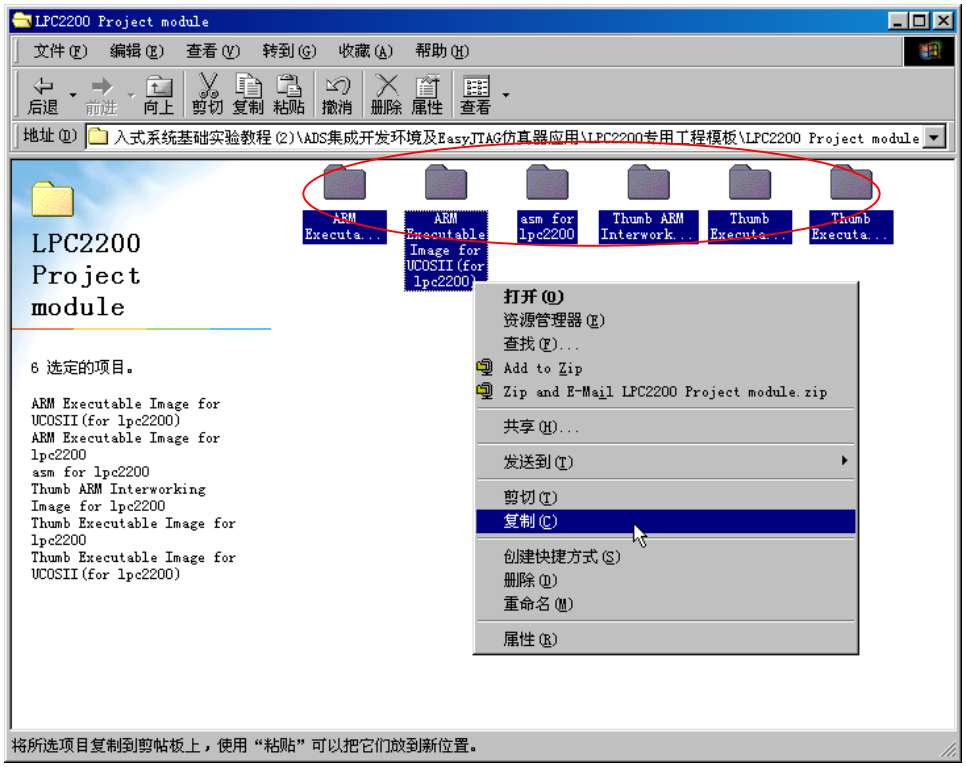


图 1.16 选择拷贝的文件和目录

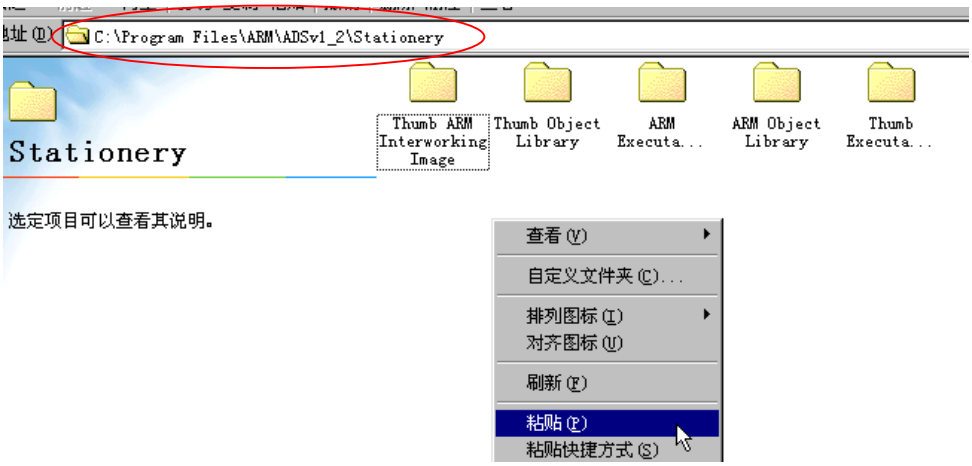


图 1.17 复制文件目录

1.4.2 使用 LPC2200 专用工程模板建立工程

启动 ADS1.2 IDE，点击【File】菜单，选择【New...】即弹出 New 对话框，如图 1.18 所示。由于事先增加了 LPC2200 专用工程模板，所以在工程模板栏中多出几项工程模板选项。

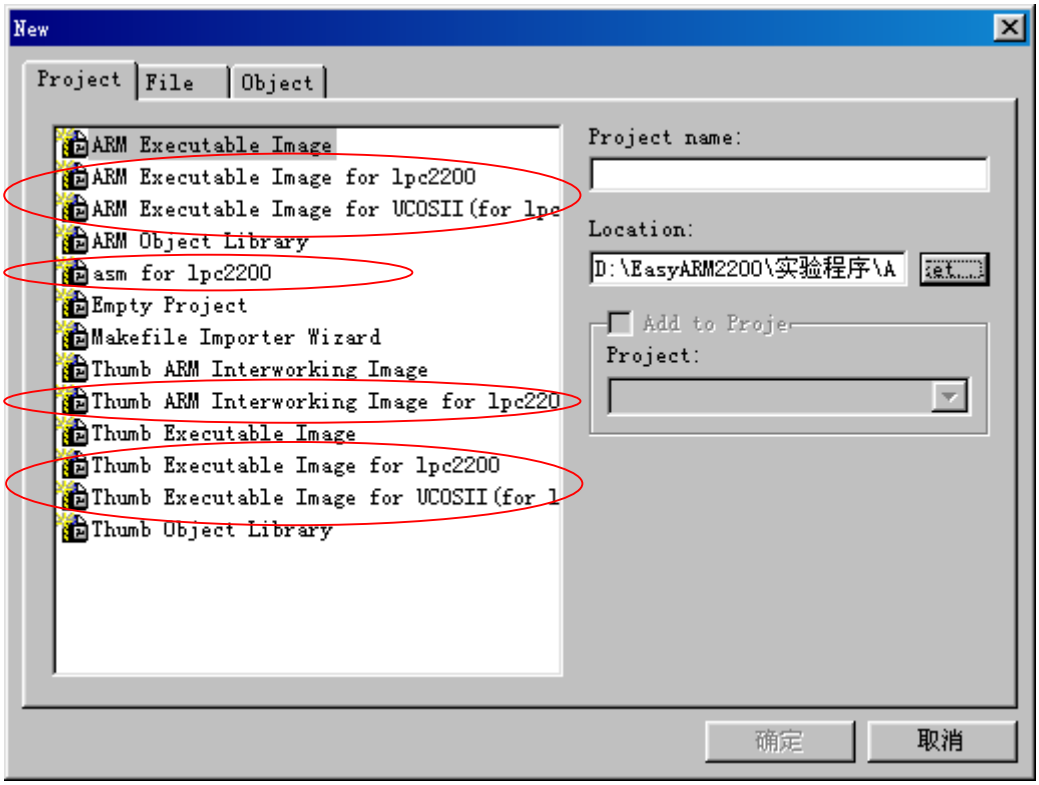


图 1.18 增加的工程模板

LPC2200 专用工程模板说明如下：

ARM Executable Image for lpc2200 ： 无操作系统时所有 C 代码均编译成 ARM 指令的工程模板。

asm for lpc2200 ： 汇编程序工程模板。

Thumb ARM Interworking Image for lpc2200 ： 无操作系统时部分 C 代码编译为 ARM 指令，部分 C 代码编译为 Thumb 指令的工程模板。

Thumb Executable Image for lpc2200 ： 无操作系统时所有 C 编译成 Thumb 指令的工程模板。

ARM Executable Image for UCOSII(for lpc2200) ： 所有 C 代码均编译为 ARM 指令的 μC/OS-II 工程模板

Thumb Executable Image for UCOSII(for lpc2200) ： 部分 C 代码编译为 ARM 指令，部分 C 代码编译为 Thumb 指令的μC/OS-II 工程模板（使用μC/OS-II 时，不可能所有代码均编译成 Thumb 指令）。

用户选择相应的工程模板建立工程，如图 1.19 所示为使用 ARM Executable Image for lpc2200 工程模板建立的一个工程。工程有四个生成目标(target system)：DebugInExram、DebugInChipFlash、RelInChip 和 RelOutChip，它们的配置如表 1.2 所示。工程模板已经将相应的编译参数设置好了，可以直接使用即可。

注意：选用 RelInChip 目标时，将会对 LPC2200 芯片进行加密（没有片内 FLASH 的芯片不能加密）。加密的芯片只能使用 ISP 进行芯片全局擦除后，才能恢复 JTAG 调试及 ISP 读/写操作。

表 1.2 LPC2200 专用工程模板各生成目标的配置

生成目标	分散加载描述文件	调试入口点地址	C 优化等级	应用说明

DebugInExram	mem_b.scf	0x80000000	Most	片外 RAM 调试模式，程序在片外 RAM 中
DebugInChipFlash	mem_c.scf	0x00000000	Most	片内 FLASH 调试模式，程序在片内 FLASH 中
RelInChip	mem_c.scf	0x00000000	Most	片内 FLASH 工作模式，程序在片内 FLASH 中。**程序写入芯片后芯片即被加密**
RelOutChip	mem_a.scf	0x80000000	Most	片外 FLASH 工作模式，程序在片外 FLASH 中

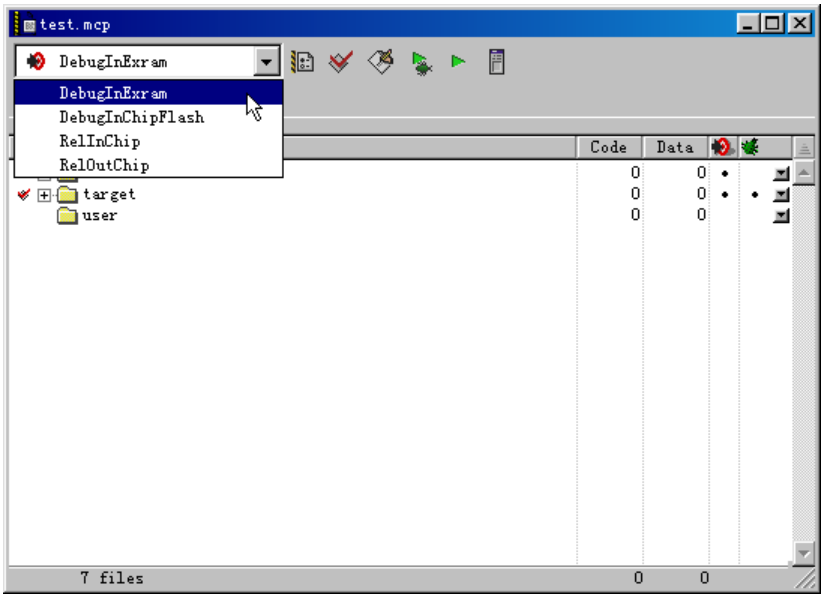


图 1.19 用 LPC2200 专用工程模板建立的工程

1.4.3 模板适用范围

(1) 本模板假设用户系统使用片外存储器。如果用户不使用片外存储器，可以使用 LPC2100 的工程模板，下载地址为 http://www.zlgmcu.com/tools/kaifaban/EasyARM2100.asp 的 EasyARM2100 开发套件快速入门和 LPC210…。

(2) 本模板假设用户系统片外存储器使用 16 位总线，且不使用 ETM 功能。如果用户的片外存储器不是使用 16 位总线，和 / 或使用 ETM 功能，需要修改 Startup.s 这个文件，修改点见程序清单 1.1。如何修改请参考 LPC2200 芯片的使用手册，下载地址为：http://www.zlgmcu.com/philips/philips-arm.asp。

注意：各个工程模板中的 Startup. s 不完全相同，可根据需要分别修改。

程序清单 1.1 Startup.s 文件需要更改的代码

```
        ……
ResetInit
;初始化外部总线控制器，根据目标板决定配置
        LDR     R0, =PINSEL2
    IF :DEF: EN_CRP
        LDR     R1, =0x0f814910
```

```
; 0x0f814910 改成需要的数值，注意最低 4 位必须为 0
    ELSE
        LDR     R1, =0x0f814914
; 0x0f814914 改成需要的数值，如果使用 ETM，最后的 4 需要修改为 6
    ENDIF
        STR     R1, [R0]

        LDR     R0, =BCFG0
        LDR     R1, =0x1000ffef
; 0x1000ffef 改成需要的数值
        STR     R1, [R0]

        LDR     R0, =BCFG1
        LDR     R1, =0x1000ffef
; 0x1000ffef 改成需要的数值
        STR     R1, [R0]
         ……
```

(3) 生成目标 DebugInExRam。假设用户系统在调试时片外 RAM 使用 bank0（即起始地址为 0x8000 0000），这一条不可修改。如果用户系统不是这样，就不能使用 DebugInExRam 这个生成目标调试程序。

(4) 生成目标 DebugInExRam。假设用户系统在调试时片外 RAM 大小为 512K 字节，此条仅影响 DebugInExRam 这个生成目标。如果不是，则需要修改 mem_b.scf 这个文件，修改点见程序清单 1.2。

注意：Windows 会隐藏这种文件的扩展名，仅显示为 mem_b。

程序清单 1.2 mem_b.scf 文件需要修改的代码

```
  ……
    ERAM 0x80040000
/*从这个地址开始存储程序的可读写的变量，根据实际情况修改*/
    {
        * (+RW,+ZI)
    }
  ……
    HEAP_BOTTOM 0x80080000   UNINIT
/*这个地址是片外 RAM 的结束地址，根据实际情况修改*/
    {
        Startup.o (HeapTop)
    }
```

(5) 生成目标 RelOutChip。假设用户系统在使用外部启动时，片外 FLASH 起始地址必须为 0x8000 0000(这是 LPC2200 芯片的要求)，片外 RAM 使用 Bank1（即起始地址为 0x8100 0000）。如果没有片外 FLASH，则不要使用 RelOutChip 这个生成目标。如果片外 RAM 起始地址不为 0x8100 0000，则需要修改 mem_a.scf 文件，修改点见程序清单 1.3。

注意：Windows 会隐藏这种文件的扩展名，仅显示为 mem_a。

程序清单 1.3 mem_a.scf 文件需要修改的代码—片外 RAM

```
    ……
      ERAM 0x81000000
/*从这个地址开始存储程序的可读写的变量，改成实际的片外 RAM 起始地址*/
     {
          * (+RW,+ZI)
      }
      ……
     HEAP_BOTTOM 0x81080000   UNINIT
/*这个地址是片外 RAM 的结束地址，根据实际情况修改*/
     {
          Startup.o (HeapTop)
     }
```

(6) 生成目标 DebugInChipFlash 和 RelInChip。假设用户系统片外 RAM 使用 Bank0（即起始地址为 0x8000 0000）。如果片外 RAM 起始地址不为 0x8000 0000，则需要修改 mem_c.scf 文件，修改点见程序清单 1.4。

程序清单 1.4 mem_c.scf 文件需要修改的代码—片外 RAM

```
    ……
      ERAM 0x80000000
/*从这个地址开始存储程序的可读写的变量，改成实际的片外 RAM 起始地址*/
     {
          * (+RW,+ZI)
      }
      ……
     HEAP_BOTTOM 0x80080000   UNINIT
/*这个地址是片外 RAM 的结束地址，根据实际情况修改*/
     {
          Startup.o (HeapTop)
     }
```

注意：用户还可以修改 mem_a.scf、mem_b.scf、mem_c.scf 这几个文件对内存的使用进行更多的控制。

(7) 为了适应不同速度的存储器，工程模板默认配置 4 个 Bank 存储器接口为最慢的访问速度。用户可以根据实际使用的存储器重新配置访问速度，以获得最好的系统性能，参考程序清单 1.5。

程序清单 1.5 在 target.c 文件中配置存储器接口访问速度

```
void   TargetResetInit(void)
{
```

```
#ifdef __DEBUG
    MEMMAP = 0x3;                       //remap
    /* 重新配置 Bank0 的访问速度 */
    BCFG0   = 0x10000400;
#endif

#ifdef __OUT_CHIP
    MEMMAP = 0x3;                       //remap
    /* 重新配置 Bank0 的访问速度 */
    BCFG0   = 0x10000400;
#endif

#ifdef __IN_CHIP
    MEMMAP = 0x1;                       //remap
    /* 重新配置 Bank0 的访问速度 */
    BCFG0   = 0x10000400;
#endif
    ……
}
```

注意：用户还可以修改 target.c 的 TargetResetInit()函数在进入 main 函数前初始化更多的东西（使用汇编程序工程模板的工程除外）。

1.5 EasyJTAG 仿真器的安装与应用

EasyJTAG 仿真器是广州周立功单片机发展有限公司开发的 LPC2000 系列 ARM7 微控制器的 JTAG 仿真器，支持 ADS1.2 集成开发环境，支持单步、全速及断点等调试功能，支持下载程序到片内 FLASH 和特定型号的片外 FLASH，采用 ARM 公司提出的标准 20 脚 JTAG 仿真调试接口。其主要特点如下：

- 采用 RDI 通讯接口，无缝嵌接 ADS1.2 和其它采用 RDI 接口的 IDE 调试环境。
- 高达 1M 速率的 JTAG 时钟驱动。
- 采用同步 Flash 刷新技术（synFLASH），同步下载用户代码到 Flash 中，即下即调。
- 采用同步时序控制技术（synTIME），仿真可靠稳定。
- 支持 32 位 ARM 指令/16 位 THUMB 指令的混合调试。
- 增加映射寄存器窗口，方便用户查看/修改寄存器数值。
- 微型体积设计，方便用户灵活使用。

EasyJTAG 仿真器外观如图 1.20 所示，其驱动程序可在 http://www.zlgmcu.com/tools/kaifaban/EasyARM2200.asp 网页下载获得，或在产品光盘上获得(其目录名为 EasyJTAG_drive，该目录下有一个 readme.txt 的文件说明)。



图 1.20 EasyJTAG 仿真器实物外观

1.5.1 安装 EasyJTAG 仿真器

首先，将 EasyJTAG 仿真器的驱动程序(比如产品光盘 EasyJTAG_drive 目录下的所有文件)复制到 ADS 的 BIN 目录，如 C:\Program Files\ARM\ADSv1_2\BIN。

接着，将 EasyJTAG 仿真器的 25 针接口通过并口延长线与 PC 机的并口连接，将 EasyJTAG 仿真器的 20 针接口通过 20 PIN 连接电缆接到 SmartARM2200 开发板的 J2 上，**再使用配套的变压器(9V)给开发板供电。**

然后，进入 AXD 调试环境，打开【Options】->【Configure Target...】，弹出 Choose Target 窗口，如图 1.12 所示。点击“ADD”添加仿真器的驱动程序，在添加文件窗口选择如 C:\Program Files\ARM\ADSv1_2\BIN 目录下的 EasyJTAG.dll，点击“打开”即可。

说明：点击 Windows 系统的【开始】->【程序】->【ARM Developer Suite v1.2】->【AXD Debugger】可以直接运行 AXD 软件。

注意：若在添加文件窗口中没有显示 DLL 文件，请设置 WINDOWS 文件浏览窗口的“文件夹选项(O)...”，将查看页中的“隐藏文件”项选用“显示所有文件”。

1.5.2 使用 EasyJTAG 仿真器

将计算机并口与 EasyJTAG 仿真器连接，并将仿真器 JTAG 口接头插入 SmartARM2200 开发板的 J2，通过 AXD 软件的设置即可进行仿真调试。

1. 仿真器设置

在 AXD 调试环境, 打开【Options】->【Configure Target...】，弹出 Choose Target 窗口，在“Target Environments”框中选择“EasyJTAG...”项。

点击“Configure”按钮，进入“EasyJTAG Setup”设置窗口，见图 1.21。在"ARMcore" 项中选取 CPU 类型，在“Options”项中选择 Halt program。然后点击“OK”，再点击“OK”，此时 EasyJTAG 将会进行连接(开发板)的操作。**若连接成功，则开发板上的 LPC2210 芯片由 EasyJTAG 控制，原先运行的程序被停止。**

注意：有时，AXD 会弹出如图 1.23 所示的错误对话框，或者类似的对话框，此时可以点击“Connect mode...”，然后选择“ATTACH...”项确定，再点击“Restart”。若 EasyJTAG 正确连接开发板，AXD 代码窗口将显示空白，接下来就可以使用【File】->【Load Image...】加载调试文件，进行 JTAG 调试。

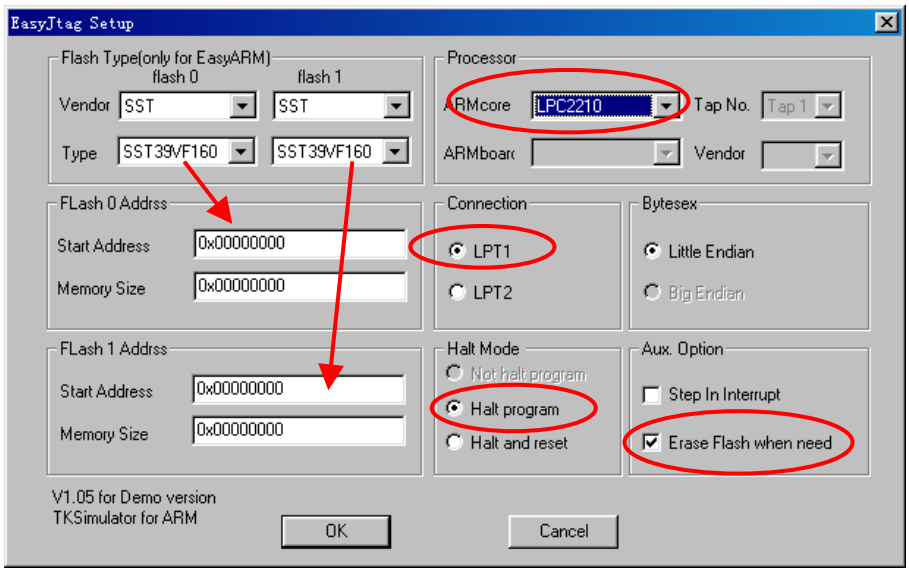


图 1.21 “EasyJTAG Setup”设置窗口

EasyJTAG 设置选项说明：

ARMcore 项，选择 CPU 型号；

Tap No.项，当 CPU 为 LPC2106/2105/2104 时，选择主/从 JTAG 调试口，Tap1 为主，Tap2 为从；

Connection，硬件连接接口选择；

Halt Mode，停机模式选择，包含 Halt program(停止 CPU)和 Halt and reset(复位然后停止 CPU)两项；

Aux. Option，辅助选项，包含 Step In Interrupt(允许单步运行进入中断)和 Erase Flash when need(允许 EasyJTAG 擦除 Flash)两项；

Flash Type，片外 FLASH 型号选择，可支持两块 FLASH 芯片，当 ARMcore 选择 LPC2200 系列 CPU 时此项才有效。当程序需要下载到片外 FLASH 时，EasyJTAG 仿真器会按所选芯片型号进行擦除/编程。

Flash 0 Addrss，第一块 Flash 的地址设置，包含 Start Address(Flash 的起始地址，比如 Bank0 时为 0x80000000)和 Memory Size(存储器容量，按实际芯片容量填写，比如 SST39VF160 的容量为 0x200000)。**当程序不需要下载到片外 FLASH，或系统没有片外 FLASH 时，Start Address 和 Memory Size 均要设置为 0。**

Flash 1 Addrss，同 Flash 0 Addrss。

2. 仿真器的应用问题

在 ADS1.2 IDE 环境中按 F5 键或 Debug 图标按钮即可直接进入 AXD，但有时会出现如图 1.22 所示的提示，处理方法是点击“确定”，然后在弹出的 Load Session 窗口中点击“取消”。若进入 AXD 后，主调试窗口没有任何代码，且【File】->【Load Image…】菜单项无效时，此时需要重新打开【Options】->【Configure Target...】点击“OK”，再点击【File】选择【Load Image…】加载调试文件。

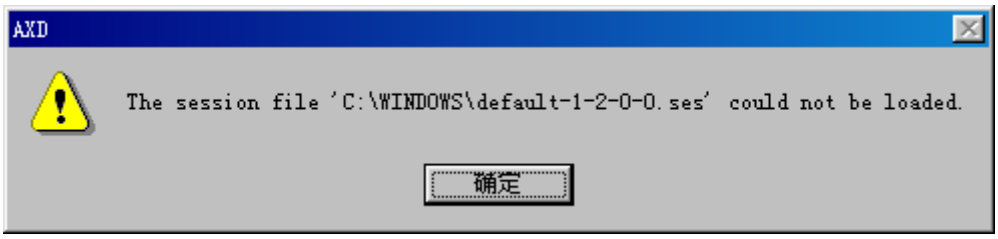


图 1.22 session 文件错误提示

在进入 AXD 调试环境后，有时会弹出 Fatal AXD Error 窗口，如图 1.23 所示，此时可以点击“Connect mode...”，然后选择“ATTACH ...”项确定，再点击“Restart”。接下来就可以使用【File】->【Load Image...】加载调试文件，进行 JTAG 调试。

注意：对于有的 PC 机，EasyJTAG 不能正确连接开发板，总是弹出错误对话框，这时可以检查并口连接是否可靠，检查并口上是否接有软件狗，或者重新给开发板上下电。另外，在 PC 机的 CMOS 设置里将并口模式设为 SPP 模式，设置并口的资源为 378H～37FH。

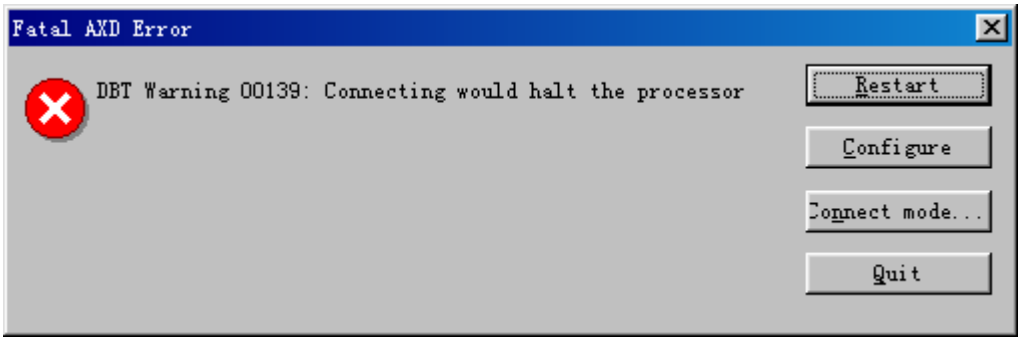


图 1.23 Fatal AXD 错误提示

片内外设的寄存器观察。在【System Views】->【Debugger Internals】即可打开 LPC2000 系列 ARM7 微控制器的片内外设寄存器窗口。有些寄存器是不能读出显示或读操作会影响其它寄存器的值，所以在片内外设寄存器窗口中不能找到，如果需要观察这些寄存器，可以使用存储器观察窗口(Memory)来实现。

使用 JTAG 下载程序到 FLASH。进入 AXD 调试环境，打开【Options】->【Configure Target...】，弹出 Choose Target 窗口，点击“Configure”按钮，进入“EasyJTAG Setup”设置窗口，在“FLASH”项中选择"Erase Flash when need"，然后确定退出。这样，每次装载 FLASH 地址的调试文件时，将会擦除 FLASH 并下载代码到 FLASH 中。

1.6 固化程序

在 JTAG 仿真调试通过后，要将程序下载到片内的 FLASH 或外部的 FLASH 中(即固化程序)，才可脱机运行。

1.6.1 片内 FLASH 的固化

对于 LPC2200 系列 ARM7 微控制器芯片来说，固化程序到片内 FLASH 可通过两种方式实现：JTAG 接口下载和使用 ISP 功能下载。 不管使用哪一种方式，用户均要先设置编译链接的地址，即代码地址从 0x00000000 地址开始，比如使用 LPC2200 专用工程模板时，在生成目标选用 RelInChip，其分散加载描述文件 mem_c.scf 如程序清单 1.6 所示。

其中，ROM_LOAD 为加载区的名称，其后面的 0x00000000 表示加载区的起始地址（存放程序代码的起始地址），也可以在后面添加其空间大小，如“ROM_LOAD 0x00000000

0x20000”表示加载区起始地址为 0x00000000，大小为 128K 字节；ROM_EXEC 描述了执行区的地址，放在第一块位置定义，其起始地址、空间大小与加载区起始地址、空间大小要一致。从起始地址开始放置向量表(即 Startup.o(vectors, +First)，其中 Startup.o 为 Startup.s 的目标文件)，接着放置其它代码(即* (+RO))；变量区 IRAM 的起始地址为 0x40000000，放置 Startup.o (MyStacks)；变量区 ERAM 的起始地址为 0x80000000，放置除 Startup.o 文件之外的其它文件的变量(即* (+RW,+ZI))；紧靠 ERAM 变量区之后的是系统堆空间(HEAP)，放置描述为 Startup.o (Heap)；堆栈区 STACKS 使用片内 RAM，由于 ARM 的堆栈一般采用满递减堆栈，所以堆栈区起始地址设置为 0x40004000，放置描述为 Startup.o (Stacks)。

程序清单 1.6 用于固化程序的分散加载描述文件 mem_c.scf

```
ROM_LOAD 0x00000000
{
    ROM_EXEC 0x00000000
    {
        Startup.o (vectors, +First)
        * (+RO)
    }

    IRAM 0x40000000
    {
        Startup.o (MyStacks)
    }

    STACKS_BOTTOM +0 UNINIT
    {
        Startup.o (StackBottom)
    }

    STACKS 0x40004000 UNINIT
    {
        Startup.o (Stacks)
    }

    ERAM 0x80000000
    {
        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        Startup.o (Heap)
    }

    HEAP_BOTTOM 0x80080000 UNINIT
```

```
    {
        Startup.o (HeapTop)
    }
}
```

1. 使用 JTAG 接口下载

使用 JTAG 接口下载程序到 FLASH 是需要 JTAG 仿真器的支持。EasyJTAG 仿真器可支持 LPC2000 系列 ARM7 微控制器的片内 FLASH 下载，这样就可以使用这一功能将程序下载到 FLASH 中，以便脱机运行。

首先设置 EasyJTAG 仿真器，参见图 1.24，注意 ARMcore 项一定要选择正确的 CPU 型号，否则可能会导致编程出错。

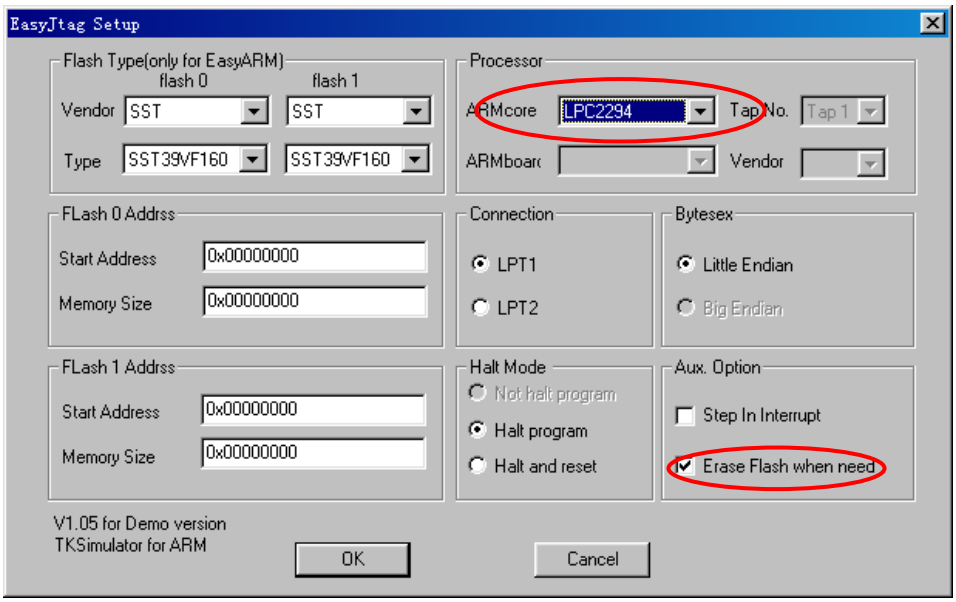


图 1.24 下载片内 FLASH 的 EasyJTAG 设置

然后将工程的生成目标选用 RelInChip，编译链接，再按 F5 键进入 AXD 调试环境，在加载调试映像文件时即会下载程序到 FLASH 中。

实际上，只要你加载调试映像文件，且代码的地址设置为 FLASH 的地址，EasyJTAG 仿真器即把程序下载到指定的 FLASH 空间。

2. 使用 ISP 下载

LPC2200 系列 ARM7 微控制器芯片具有 ISP 功能(LPC2210 无片内 FLASH，不能进行 ISP 编程)，可以通过串口进行程序下载。

首先把当前工程编译生成 HEX 文件，打开工程的 DebugRel Settings 窗口，在 Target Settings 项中设置 Post-linker 选取 ARM fromELF（如图 1.25 所示）。

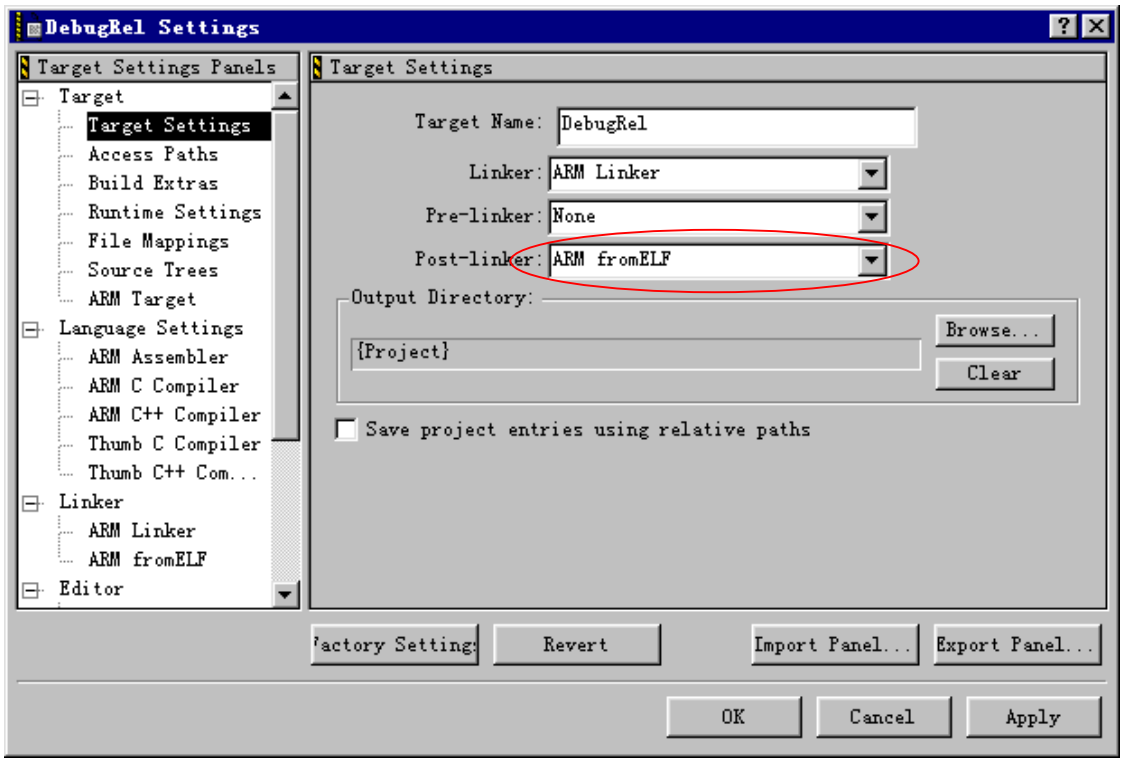


图 1.25 设置 Post-linker

在 ARM formELF 项中设置输出文件类型，如设置为 Intel 32 bit Hex，然后设置输出文件名，也可指定目录，若不指定目录，则生成文件存放在当前工程的目录中（如图 1.26 所示）。重新编译连接，编译通过即会生成指定的输出文件。

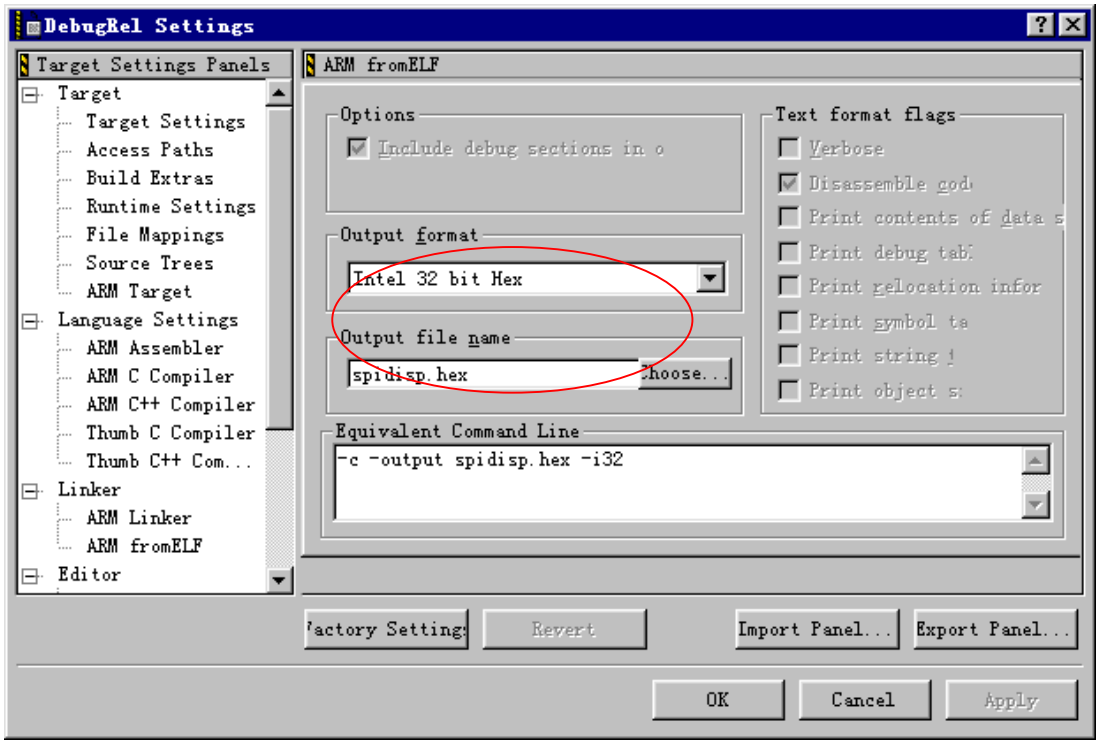


图 1.26 生成文件设置

生成 HEX 文件后，接下来使用串口延长线连接 PC 串口(如 COM1)和 SmartARM2200 实验板(UART0)，并将实验板上的 ISP(JP1)跳线短接。打开 LPC2000 Flash Utility 软件，并设置串口、波特率、系统晶振(注意，晶振频率项单位为 kHz)等，如图 1.27 所示。

设置好参数后，点击 Read Device ID 按钮，读取芯片 ID 号，若读取成功(状态栏显示“Read Part ID Successfully! ”)，则表明 ISP 连接成功。否则，当出错提示为复位 LPC2000 信息时，首先按 SmartARM2200 开发板上的 RST 键复位，然后再确定提示，如图 1.28 所示。

连接成功后，先使用 Erase 按钮擦除选定扇区的 FLASH，然后在 Filename 项中输入要下载的 HEX 文件全名，点击 Upload to Flash 按钮即开始下载程序。程序固化后，将 ISP(JP1)跳线断开，重新复位系统即可运行程序。说明，LPC2200 系列 ARM7 微控制器要求向量表中所有的数据(即 0x00000000～0x0000001c 地址上的指令机器码)32 位累加和为零，才能启动用户程序。通过设置保留异常向量地址 0x14 上的数据实现。

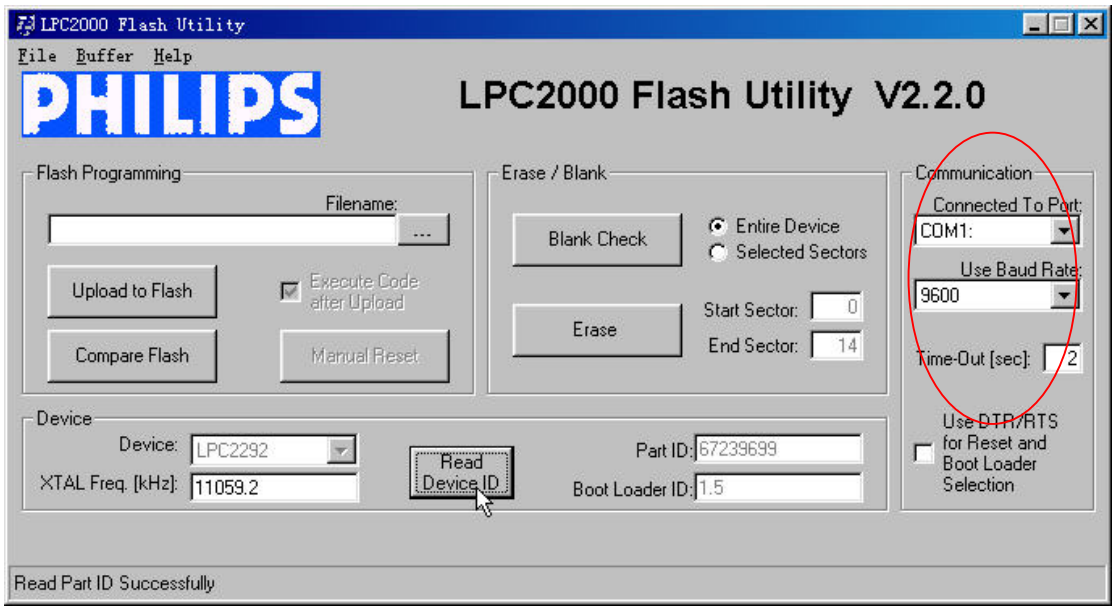


图 1.27 LPC2000 Flash Utility 软件设置



图 1.28 复位 LPC2000 提示

3. 脱机运行

- 将 JP9 跳线选择 INSIDE，将 JP1 断开禁止 ISP；
- 将 JP10 跳线器选择 RAM 为 BANK0 地址，FLASH 为 BANK1 地址；
- 复位系统，即可启动片内 FLASH 中的程序。

1.6.2 片外 FLASH 的固化

EasyJTAG 仿真器支持对特定的片外 FLASH 编程。用户要先设置编译链接的地址，即

代码地址从 0x80000000 地址开始，比如使用 LPC2200 专用工程模板时，在 target system 选用 RelOutChip，其分散加载描述文件 mem_a.scf 如程序清单 1.7 所示。

其中，ROM_LOAD 为加载区的名称，其后面的 0x80000000 表示加载区的起始地址(因为片外 FLASH 分配为 Bank0)；ROM_EXEC 描述了执行区的地址，放在第一块位置定义，其起始地址、空间大小与加载区起始地址、空间大小要一致，从起始地址开始放置向量表(即 Startup.o(vectors, +First)，其中 Startup.o 为 Startup.s 的目标文件)，接着放置其它代码(即* (+RO))；变量区 IRAM 的起始地址为 0x40000000，放置 Startup.o (MyStacks)；堆栈区 STACKS 使用片内 RAM，由于 ARM 的堆栈一般采用满递减堆栈，所以堆栈区起始地址设置为 0x40004000，放置描述为 Startup.o (Stacks)；变量区 ERAM 的起始地址为 0x81000000(因为片外 RAM 分配为 BANK1)，放置除 Startup.o 文件之外的其它文件的变量(即* (+RW,+ZI))；紧靠 ERAM 变量区之后的是系统堆空间(HEAP)，放置描述为 Startup.o (Heap)；

程序清单 1.7 用于固化程序的分散加载描述文件 mem_a.scf

```
ROM_LOAD 0x80000000
{
    ROM_EXEC 0x80000000
    {
        Startup.o (vectors, +First)
        * (+RO)
    }

    IRAM 0x40000000
    {
        Startup.o (MyStacks)
    }

    STACKS_BOTTOM +0 UNINIT
    {
        Startup.o (StackBottom)
    }

    STACKS 0x40004000 UNINIT
    {
        Startup.o (Stacks)
    }

    ERAM 0x81000000
    {
        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        Startup.o (Heap)
```

```
    }

    HEAP_BOTTOM 0x81080000 UNINIT
    {
        Startup.o (HeapTop)
    }
}
```

1. 使用 JTAG 接口下载

使用 JTAG 接口下载程序到片外 FLASH 是需要 JTAG 仿真器的支持。EasyJTAG 仿真器支持对特定的片外 FLASH 下载程序，这样就可以使用这一功能将程序下载到片外 FLASH 中，以便脱机运行。

首先将 JP10 跳线选择 Bank0-Flash，Bank1-Ram；

然后设置 EasyJTAG 仿真器，参见图 1.29；

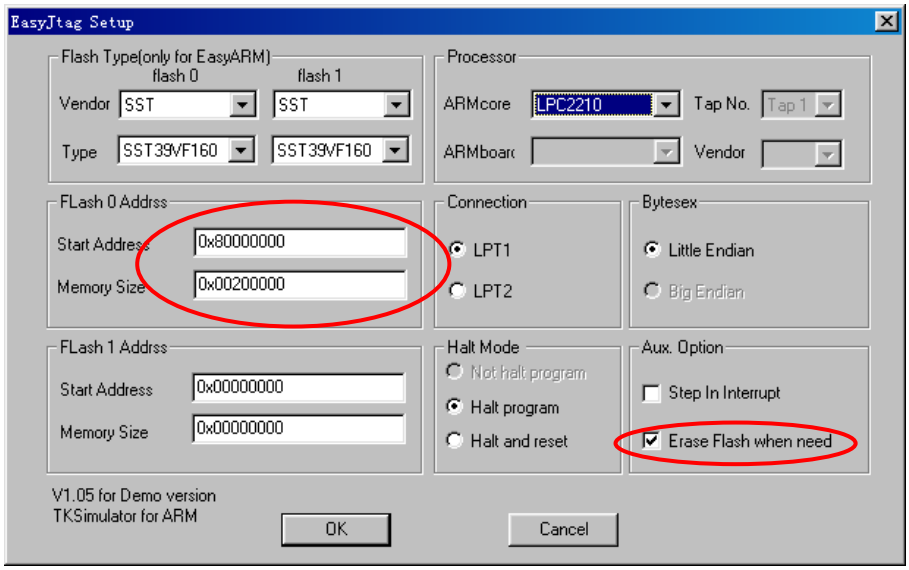


图 1.29　下载片外 FLASH 的 EasyJTAG 设置

最后将工程的生成目标选用 RelOutChip，编译链接，再按 F5 键进入 AXD 调试环境，在加载调试映像文件时即会下载程序到片外 FLASH 中。

实际上，只要你加载调试映像文件，且代码的地址是设置为片外 FLASH 的地址，EasyJTAG 仿真器即把程序下载到指定的 FLASH 空间。

2. 脱机运行

- 将 JP9 跳线选择 OUTSIDE，将 JP1 断开禁止 ISP；
- 将 JP10 跳线选择 Bank0-Flash，Bank1-Ram；
- 复位系统，即可启动片外 FLASH 中的程序。

第2章 基础实验

2.1 外部中断实验 2

1. 实验目的

（1）掌握向量 IRQ 中断的设置及应用；
（2）掌握外部中断引脚功能设置及外部中断的工作模式设置；

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境

3. 实验内容

设置 P0.20 脚为 EINT3 功能，初始化为向量中断，并设置为下降沿触发模式，然后等待外部中断。中断服务程序将蜂鸣器控制输出信号取反，然后清除中断标志并退出中断。

4. 实验预习要求

仔细阅读《ARM 嵌入式系统基础教程》第 5.4.6 节外部中断输入的说明，第 5.8 节向量中断控制器的说明。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 VICVect_C。
（2）在 user 组中的 main.c 中编写主程序代码。
（3）在 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈处的代码为“MSR CPSR_c, #0x5f”，即使能 IRQ 中断。
（4）选用 DebugInExram 生成目标，然后编译连接工程。
（5）将 SmartARM2200 教学实验开发平台上的 JP2、JP4 跳线短接，JP7 断开。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。
（6）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。
（7）在中断服务程序中设置断点，全速运行程序，使 EINT3 为低/高电平，即反复按下与释放 KEY1。
（8）单步/全速运行程序，观察程序是否正确运行，蜂鸣器是否蜂鸣。
（9）一直按下 KEY1，观察是否会不断产生中断。
注意：KEY1 操作中的抖动，可能引起多次中断。

6. 实验参考程序

外部中断实验 2 的参考程序见程序清单 2.1。

程序清单 2.1 外部中断实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
```

```
* 功    能：使用外部中断 3 进行 B1 的控制，每当有一次中断时，即取反 B1 控制口，以便指示中断输入。
*           使用向量中断方式，EINT3 下降沿有效。
* 说    明：将跳线器 JP2、JP4 短接，JP7 断开，然后反复按下与释放 KEY1。
*********************************************************************************/
#include   "config.h"

#define       BEEPCON    1<<7                  // P0.7 引脚控制 B1，低电平蜂鸣，1<<7 等价于 0x80

/*********************************************************************************
* 名    称：IRQ_Eint3()
* 功    能：外部中断 EINT3 服务函数，取反 B1 控制口。
* 入口参数：无
* 出口参数：无
*********************************************************************************/
void    __irq IRQ_Eint3(void)
{
    uint32   i;

    i = IO0SET;                               // 读取当前 B1 控制值
    if( (i&BEEPCON)==0 )                      // 控制 B1 输出取反
    {
        IO0SET = BEEPCON;
    }
    else
    {
        IO0CLR = BEEPCON;
    }

   EXTINT = 1<<3;                             // 清除 EINT3 中断标志，1<<3 等价于 0x08
   VICVectAddr = 0;                           // 向量中断结束
}

/*********************************************************************************
* 名    称：main()
* 功    能：初始化外部中断 3(EINT3)为向量中断，并设置为下降沿触发模式，然后等待外部中断。
* 说    明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)。
*********************************************************************************/
int   main(void)
{
    PINSEL1 = 3<<8;                 // 设置管脚连接，P0.20 设置为 EINT3
                                    // 3<<8 等价于 0x00000180
    IO0DIR = BEEPCON;               // 设置 B1 控制口为输出，其它 I/O 为输入
    EXTMODE = 1<<3;                 // 设置 EINT3 中断为边沿触发模式
                                    // 1<<3 等价于 0x08
```

```
    EXTPOLAR = 0x00;                // 设置 EINT3 中断为下降沿触发

    /* 打开 EINT3 中断(设置向量控制器，即使用向量 IRQ) */
    VICIntSelect = 0x00000000;      // 设置所有中断分配为 IRQ 中断
    VICVectCntl0 = 0x20|17;         // 分配 EINT3 中断到向量中断 0
                                    // 0x20 表示向量 IRQ 使能，1<<17 表示 EINT3 在 VIC 通道 17 号
    VICVectAddr0 = (int)IRQ_Eint3;  // 设置中断服务程序地址
    EXTINT = 1<<3;                  // 清除 EINT3 中断标志
    VICIntEnable = 1<<17;           // 使能 EINT3 中断，EINT3 在 VIC 通道 17 号

    while(1);                       // 等待中断
    return(0);
}
```

7. 思考

（1）产生 IRQ 中断后，CPSR 寄存器的 I 位是 0 还是 1？
（2）如何正确理解 VIC 的中断优先级？
（3）将实验参考程序的 EINT3 中断设置为 Slot10 向量中断。

2.2 外部存储器接口实验 2

1. 实验目的

通过实验掌握外部存储器控制器(EMC)的设置，使外部存储器的访问速度最优化，提高外部程序的运行速度。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件

3. 实验内容

在外部 RAM 运行 LED 流水灯显示控制程序(使用软件延时)，并使用定时器 0 来测量每一轮循环所需要的时间，将定时器的值(即程序运行的时间)通过串口向上位机发送。通过更改 EMC 存储器组的配置，控制外部 RAM 的访问速度，然后观察程序的运行速度。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》第 5.6 节外部存储器控制器的说明。
（2）仔细阅读本书第 1 章的内容，了解 SmartARM2200 教学实验开发平台的硬件结构，注意系统存储器电路。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 Speed_C。

（2）在 user 组中的 main.c 中编写主程序代码，在项目中的 config.h 文件中加入#include <stdio.h>。

（3）在 Startup.s 文件的 ResetInit 子程序中，观察 BCFG0 和 BCFG1 寄存器的设置值，可知工程模板默认配置外部存储器接口为最慢的访问速度。

（4）选用 DebugInExram 生成目标，然后编译连接工程。

（5）将 SmartARM2200 教学实验开发平台上的跳线 JP12 全部短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（6）使用串口延长线把 SmartARM2200 教学实验开发平台的 CZ2(UART0)与 PC 机的 COM1 连接。PC 机运行 EasyARM 软件，设置串口为 COM1，波特率为 115200，然后选择【设置】->【发送数据】，在弹出的发送数据窗口中点击“高级”即可打开接收窗口。

（7）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（8）全速运行程序，观察流水灯显示的变化速度及 EasyARM 软件上显示的运行时间值。

（9）停止 JTAG 仿真调试，关闭 AXD 软件。在 Startup.s 文件的 ResetInit 子函数中重新配置 Bank0 的访问速度，参考程序清单 2.2。

注意：如果进入 AXD 软件不成功，请检查 BCFG0 和 BCFG1 参数是否设置总线过快。

程序清单 2.2 重新配置存储器接口访问速度

```
ResetInit
;Initial extenal bus controller.
;初始化外部总线控制器，根据目标板决定配置

        LDR     R0, =PINSEL2
    IF :DEF: EN_CRP
        LDR     R1, =0x0f814910
    ELSE
        LDR     R1, =0x0f814914
    ENDIF
        STR     R1, [R0]

        LDR     R0, =BCFG0
        LDR     R1, =0x1000ffef         ;  16 位总线最慢操作参数
        STR     R1, [R0]
….
```

（10）重新编译连接工程，再次启 AXD 进行 JTAG 仿真调试。

（11）全速运行程序，观察流水灯显示的变化速度及 EasyARM 软件上显示的运行时间值。

6. 实验参考程序

外部存储器接口实验 2 的参考程序见程序清单 2.3。

程序清单 2.3 外部存储器接口实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功    能：通过 I/O 控制 LED1~LED4，产生流水灯效果，并测量程序运行时间，
*           然后将运行时间值向 UART0 发送。
* 说    明：将跳线器 JP12 全部短接
*           通讯波特率 115200，8 位数据位，1 位停止位，无奇偶校验。
*           请修改 BCFG0 数值达到测试目的，注意 PSRAM 的极限参数，否则可能调试失败
*****************************************************************************/
#include   "config.h"
#define    LEDCON         0xf0000000
#define    UART_BPS       115200          // 定义通讯波特率

const uint32   DISP_TAB[8] = {  0x1fffffff, 0x2fffffff, 0x4fffffff, 0x8fffffff,
                                0xffffffff, 0x0fffffff, 0xffffffff, 0x0fffffff };

/*****************************************************************************
* 名    称：DelayNS()
* 功    能：长软件延时
* 入口参数：dly           延时参数，值越大，延时越久
* 出口参数：无
*****************************************************************************/
void   DelayNS(uint32   dly)
{    uint32   i;

     for(; dly>0; dly--)
     {
          for(i=0; i<5000; i++);
     }

}

/*****************************************************************************
* 名    称：UART0_Ini()
* 功    能：初始化串口 0。设置为 8 位数据位，1 位停止位，无奇偶校验，波特率为 115200
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void   UART0_Init(void)
{    uint16 Fdiv;
```

```
    U0LCR = 0x83;                           // DLAB = 1，可设置波特率
    Fdiv = (Fpclk / 16) / UART_BPS;         // 设置波特率
    U0DLM = Fdiv / 256;
    U0DLL = Fdiv % 256;
    U0LCR = 0x03;
}

/*****************************************************************************
* 名    称：UART0_SendByte()
* 功    能：向串口发送字节数据，并等待发送完毕。
* 入口参数：data            要发送的数据
* 出口参数：无
*****************************************************************************/
void  UART0_SendByte(uint8 data)
{
    U0THR = data;                           // 发送数据
    while( (U0LSR&0x40)==0 );               // 等待数据发送完毕
}

/*****************************************************************************
* 名    称：UART0_SendStr()
* 功    能：向串口发送一字符串
* 入口参数：srt             要发送的字符串的指针
* 出口参数：无
*****************************************************************************/
void  UART0_SendStr(uint8 const *str)
{
    while(1)
    {
        if( *str == '\0' ) break;
        UART0_SendByte(*str++);             // 发送数据
    }
}

/*****************************************************************************
* 名称：main()
* 功能：根据表 DISP_TAB 来控制 LED 显示。
*****************************************************************************/
int  main(void)
{   uint8   i;
    char disp_buf[30];

    PINSEL0 = 0x00000005;                   // 设置 I/O 连接到 UART0
    UART0_Init();
```

```
    /* PINSEL2 使用启动代码的默认配置，切勿任意配置 PINSEL2，否则总线会受到干扰 */
    IO2DIR = LEDCON;
    T0PR = 0;
    while(1)
    {
        T0TC = 0;
        T0TCR = 0x01;

        for(i=0; i<8; i++)
        {
            IO2SET = DISP_TAB[i];       // 输出 LED 显示数据
            DelayNS(10);                // 延时
            IO2CLR = 0xffffffff;
        }
        T0TCR = 0x00;
        sprintf(disp_buf, "Run time is:  %d \r\n", (uint32)T0TC);
        UART0_SendStr((uint8 *)disp_buf);
    }

    return(0);
}
```

7. 思考

（1）除了 EMC 的配置外，还有哪些系统设置会影响外部程序的访问速度？
（2）若程序放在片内 FLASH 运行，运行速度会提高吗？为什么？

2.3 定时器实验 2

1. 实验目的

熟悉 LPC2000 系列 ARM7 微控制器的定时器 0 的基本设置及定时中断应用。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境

3. 实验内容

使用定时器 0 实现 1 秒定时，控制蜂鸣器蜂鸣。采用中断方式实现定时控制。

4. 实验预习要求

仔细阅读《ARM 嵌入式系统基础教程》第 5.14 节定时器 0 和定时器 1 的说明，第 5.8

节向量中断控制器的说明。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 TimeOut_C。

（2）在 user 组中的 main.c 中编写主程序代码。

（3）在 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈处的代码为 MSR CPSR_c, #0x5f，即使能 IRQ 中断。

（4）选用 DebugInExram 生成目标，然后编译连接工程。

（5）将 SmartARM2200 教学实验开发平台上的 JP4 跳线短接，JP7 跳线断开。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（6）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（6）可以全速运行程序，蜂鸣器会响一秒，停一秒，然后再响一秒……依次循环。

6. 实验参考程序

定时器实验 2 的参考程序见程序清单 2.4。

程序清单 2.4 定时器实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功    能：使用定时器 0 实现 1 秒定时,控制蜂鸣器蜂鸣。(中断方式)
* 说    明：JP4 跳线短接，JP7 跳线断开。
*****************************************************************************/
#include   "config.h"
#define        BEEPCON   1<<7                // P0.7 引脚控制 B1，低电平蜂鸣

/*****************************************************************************
* 名    称：IRQ_Time0()
* 功    能：定时器 0 中断服务程序，取反 BEEPCON 控制口。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void __irq   IRQ_Time0(void)
{
    if( (IO0SET&BEEPCON) == 0 )
    {
        IO0SET = BEEPCON;
    }
    else
    {
        IO0CLR = BEEPCON;
    }
```

```
    T0IR = 0x01;                        // 清除中断标志
    VICVectAddr = 0x00;                 // 通知 VIC 中断处理结束
}

/*****************************************************************************
* 名    称：Time0Init()
* 功    能：初始化定时器 0，定时时间为 1S，并使能中断。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  Time0Init(void)
{   /* Fcclk = Fosc*4 = 11.0592MHz*4 = 44.2368MHz
       Fpclk = Fcclk/4 = 44.2368MHz/4 = 11.0592MHz
    */
    T0PR = 99;                          // 设置定时器 0 分频为 100 分频，得 110592Hz
    T0MCR = 0x03;                       // 匹配通道 0 匹配中断并复位 T0TC
    T0MR0 = 110592;                     // 比较值(1S 定时值)
    T0TCR = 0x03;                       // 启动并复位 T0TC
    T0TCR = 0x01;

    /* 设置定时器 0 中断 IRQ */
    VICIntSelect = 0x00;                // 所有中断通道设置为 IRQ 中断
    VICVectCntl0 = 0x24;                // 定时器 0 中断通道分配最高优先级(向量控制器 0)
    VICVectAddr0 = (uint32)IRQ_Time0;   // 设置中断服务程序地址向量
    VICIntEnable = 0x00000010;          // 使能定时器 0 中断
}

/*****************************************************************************
* 名    称：main()
* 功    能：初始化 I/O 及定时器，然后等待中断。
* 说    明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)。
*****************************************************************************/
int  main(void)
{
    PINSEL0 = 0x00000000;               // 设置管脚连接 GPIO
    IO0DIR = BEEPCON;                   // 设置 I/O 为输出
    Time0Init();                        // 初始化定时器 0 及使能中断
    while(1);                           // 等待定时器 0 中断或定时器 1 匹配输出

    return(0);
}
```

7. 思考

（1）使用定时器 0 和定时器 1 定时中断控制蜂鸣器(实现响 0.5 秒，停 0.5 秒)，两个定

时器均为 1S 定时，定时器 0 中断时控制蜂鸣器蜂鸣，定时器 1 中断时控制蜂鸣停止，请编写程序实现。(提示：定时器 0 启动 0.5S 后，再立即启动定时器 1)

2.4 UART 实验 2

1. 实验目的

通过实验，掌握 UART 中断程序的设计，能够理解发送 FIFO 和接收 FIFO 的功能。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件

3. 实验内容

使用串口 UART0 接收上位机发送的数据，当接收到 8 个连续数据后，将接收计数值加 1 并输出 LED1～LED4 显示，再将接收到的数据原封不动地发送回上位机。使能 UART0 的 FIFO 进行数据发送/接收，接收采用中断处理方式。UART0 设置为通讯波特率 115200，8 位数据位，1 位停止位，无奇偶校验。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》第 5.10 节 UART0 的说明，注意 FIFO 接收情况的特性。

（2）仔细阅读本书第 1 章的内容，了解 SmartARM2200 教学实验开发平台的硬件结构，注意串口部分的电路。

（3）仔细阅读 SP3232E 芯片的数据手册，了解此芯片的作用及应用电路设计。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 DataRet_C。

（2）在 user 组中的 main.c 中编写主程序代码，在项目中的 config.h 文件中加入#include <stdio.h>。

（3）在 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈处的代码为“MSR CPSR_c, #0x5f”，即使能 IRQ 中断。

（4）选用 DebugInExram 生成目标，然后编译连接工程。

（5）将 SmartARM2200 教学实验开发平台上的 JP12 跳线全部短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（6）使用串口延长线把 SmartARM2200 教学实验开发平台的 CZ2(UART0)与 PC 机的 COM1 连接。PC 机运行 EasyARM 软件，设置串口为 COM1，波特率为 115200，然后选择【设置】->【发送数据】，在弹出的发送数据窗口中点击“高级”即可打开接收窗口。

（7）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（8）全速运行程序，在 PC 机上的 EasyARM 软件发送 8 字节数据，LPC2210 接收到数据后会控制板上的 LED1～LED4 显示，并将接收到的数据回发给 PC 机。程序运行结果如

图 2.1 所示。需要注意的是必须连续发送 8 字节数据。



图 2.1　UART 实验 2 运行结果

6. 实验参考程序

UART 实验 2 的参考程序见程序清单 2.5。

程序清单 2.5　UART 实验 2 参考程序

```
/*********************************************************************************
* 文 件名：main.c
* 功    能：使用串口 UART0 接收上位机发送的数据，当接收到 8 个连续数据后，将接收计数值加一后输
*           出 LED1--LED4 显示，并将数据原封不动地发送回上位机。
* 说    明：将跳线器 JP12 全部短接。
*           通讯波特率 115200，8 位数据位，1 位停止位，无奇偶校验。
*           中断服务程序不响应单字节发送累计至 8 Bytes 的情况，所以 PC 机必须连续发送 8 Bytes。
*********************************************************************************/
#include   "config.h"
#define    LEDCON   0xf0000000

/* 定义串口模式设置数据结构 */
typedef   struct   UartMode
{    uint8 datab;                    // 字长度，5/6/7/8
     uint8 stopb;                    // 停止位，1/2
```

```
    uint8 parity;                       // 奇偶校验位，0 为无校验，1 奇数校验，2 为偶数校验
}   UARTMODE;

uint8   rcv_buf[8];                     // UART0 数据接收缓冲区
volatile uint8   rcv_new;               // 接收新数据标志

/*****************************************************************************
* 名    称：IRQ_UART0()
* 功    能：串口 UART0 接收中断。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void    __irq IRQ_UART0(void)
{   uint8   i;

    if( 0x04==(U0IIR&0x0F) ) rcv_new = 1;// 设置接收到新的数据标志
    for(i=0; i<8; i++)
    {
        rcv_buf[i] = U0RBR;             // 读取 FIFO 的数据，并清除中断标志
    }
    VICVectAddr = 0x00;                 // 中断处理结束
}

/*****************************************************************************
* 名    称：SendByte()
* 功    能：向串口 UART0 发送字节数据。
* 入口参数：data                    要发送的数据
* 出口参数：无
*****************************************************************************/
void   SendByte(uint8 data)
{
    U0THR = data;                       // 发送数据
}

/*****************************************************************************
* 名称：ISendBuf()
* 功能：将缓冲区的数据发送回主机(使用 FIFO)，并等待发送完毕。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void   ISendBuf(void)
{   uint8   i;

    for(i=0; i<8; i++) SendByte(rcv_buf[i]);
```

```
    while( (U0LSR&0x20)==0 );               // 等待数据发送
}

/*****************************************************************************
* 名      称：UART0_Init()
* 功      能：初始化串口 0。设置其工作模式及波特率。
* 入口参数：baud                          波特率
*           set                 模式设置(UARTMODE 数据结构)
* 出口参数：返回值为 1 时表示初化成功，为 0 表除参数出错
*****************************************************************************/
uint8   UART0_Init(uint32 baud, UARTMODE set)
{   uint32   bak;

    /* 参数过滤 */
    if( (0==baud)||(baud>115200) )
    {
        return(0);
    }
    if( (set.datab<5)||(set.datab>8) )
    {
        return(0);
    }
    if( (0==set.stopb)||(set.stopb>2) )
    {
        return(0);
    }
    if( set.parity>4 )
    {
        return(0);
    }

    /* 设置串口波特率 */
    U0LCR = 0x80;                           // DLAB 位置 1
    bak = (Fpclk>>4)/baud;
    U0DLM = bak>>8;
    U0DLL = bak&0xff;

    /* 设置串口模式 */
    bak = set.datab-5;                      // 设置字长度
    if(2==set.stopb)
    {
        bak |= 0x04;                        // 判断是否为 2 位停止位
    }
    if(0!=set.parity)
```

```
    {
        set.parity = set.parity-1;
        bak |= 0x08;
    }
    bak |= set.parity<<4;                    // 设置奇偶校验
    U0LCR = bak;

    return(1);
}

/*****************************************************************************
* 名    称：main()
* 功    能：初始化串口，并等待接收到串口数据。
* 说    明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)。
*****************************************************************************/
int  main(void)
{   uint8       rcv_counter;
    UARTMODE    uart0_set;

    PINSEL0 = 0x00000005;                    // 设置 I/O 连接到 UART0
    IO2DIR = LEDCON;                         // 设置连接 LED 的 I/O 为输出
    rcv_new = 0;                             // 接收标志为 0

    uart0_set.datab = 8;                     // 8 位数据位
    uart0_set.stopb = 1;                     // 1 位停止位
    uart0_set.parity = 0;                    // 无奇偶校验
    UART0_Init(115200, uart0_set);           // 初始化串口模式

    U0FCR = 0x81;                            // 使能 FIFO，并设置触发点为 8 字节
    U0IER = 0x01;                            // 允许 RBR 中断，即接收中断

    /* 设置中断允许 */
    VICIntSelect = 0x00000000;               // 设置所有通道为 IRQ 中断
    VICVectCntl0 = 0x26;                     // UART0 中断通道分配到 IRQ slot 0，即优先级最高
    VICVectAddr0 = (int)IRQ_UART0;           // 设置 UART0 向量地址
    VICIntEnable = 0x00000040;               // 使能 UART0 中断

    rcv_counter = 0;
    IO2SET = 0xffffffff;
    while(1)                                 // 等待中断
    {
        if(1==rcv_new)                       // 是否已经接收到 8 Bytes 的数据
        {
            rcv_new = 0;                     // 清除标志
```

```
            ISendBuf();                     // 将接收到的数据发送回主机
            IO2SET = 0xffffffff;            // LED 灯复位
            IO2CLR = (rcv_counter<<28);  // 通过 LED 以二进制显示计数值
            rcv_counter++;                  // 接收计数值加一

        }
    }
    return(0);
}
```

7. 思考

（1）为什么必须连续发送 8 字节数据？（提示：注意硬件 FIFO 接收方式）

（2）如果出现字符超时中断，应该怎样读取所接收到的数据？(提示：通过 U0LSR 寄存器的 RDR 位判断是否还有未读取的数据)

（3）如果需要每接收到一个字节数据就产生接收中断，应如何设计程序？

2.5 I^2C 接口实验 2

1. 实验目的

（1）掌握 LPC2000 系列 ARM7 微控制器的硬件 I^2C 接口的使用。

（2）了解 ZLG7290 键盘的使用。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件

3. 实验内容

使用主模式 I^2C 对 ZLG7290 进行操作，接收键盘输入，根据按键值在串口打印相关提示给 PC 机。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》第 5.12 节 I^2C 接口的说明。

（2）仔细阅读本书第 1 章的内容，了解 SmartARM2200 教学实验开发平台的硬件结构，注意键盘部分的电路。

（3）仔细阅读 ZLG7290 芯片的数据手册，了解此芯片的应用电路设计。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 I2cInt_c。

（2）在 user 组中的 main.c 中编写主程序代码，然后把 zlg7290.c、zlg7290.h 和 I2cInt.c、I2cInt.h 添加到工程的 user 组中，在项目中的 config.h 文件中加入“#include "I2CINT.H"”

和“#include "ZLG7290.H"”。

（3）在 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈处的代码为“MSR CPSR_c, #0x5f”，即使能 IRQ 中断。

（4）选用 DebugInExram 生成目标，然后编译连接工程。

（5）将 SmartARM2200 教学实验开发平台上的 JP6 跳线全部短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（6）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（7）使用串口延长线把 SmartARM2200 教学实验开发平台的 CZ2(UART0)与 PC 机的 COM1 连接。PC 机运行 EasyARM 软件，设置串口为 COM1，波特率为 115200，然后选择【设置】->【发送数据】，在弹出的发送数据窗口中点击“高级”即可打开接收窗口。

（8）全速运行程序，LPC2210 会通过 I^2C 接口控制 ZLG7290。按 S1～S16 键，接收窗口可以得到相关的提示。

6. 实验参考程序

I^2C 接口实验 2 的参考程序见程序清单 2.6。其中 ZLG7290 控制接口函数保存在 zlg7290.c 文件，I^2C 接口函数及中断处理程序在 I2cInt.c 文件(文件代码见产品光盘)。

程序清单 2.6 I^2C 接口实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功    能：使用硬件 I2C 对 ZLG7290 进行操作，利用中断方式操作。
* 说    明：将跳线器 JP6 短接。
*           主程序不做防抖处理
*****************************************************************************/
#include   "config.h"
#define   ZLG7290    0x70                              // 定义器件地址

/*****************************************************************************
* 名    称：UART0_Init()
* 功    能：初始化串口 0。设置为 8 位数据位，1 位停止位，无奇偶校验
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void   UART0_Init(uint32 bps)
{    uint16 Fdiv;

    PINSEL0 = (PINSEL0 & (~0x0F)) | 0x05;          // 不影响其它管脚连接,设置 I/O 连接到 UART0
    U0LCR = 0x83;                                  // DLAB = 1，可设置波特率
    Fdiv = (Fpclk / 16) / bps;                     // 设置波特率
    U0DLM = Fdiv / 256;
    U0DLL = Fdiv % 256;
    U0LCR = 0x03;
}
```

```
/*****************************************************************************
* 名    称：UART0_SendByte()
* 功    能：向串口发送字节数据，并等待发送完毕。
* 入口参数：data          要发送的数据
* 出口参数：无
*****************************************************************************/
void  UART0_SendByte(uint8 data)
{
    U0THR = data;                                   // 发送数据
    while( (U0LSR&0x40)==0 );                       // 等待数据发送完毕
}

/*****************************************************************************
* 名    称：UART0_SendStr()
* 功    能：向串口发送一字符串
* 入口参数：srt           要发送的字符串的指针
* 出口参数：无
*****************************************************************************/
void  UART0_SendStr(char *str)
{
    while(1)
    {
        if( *str == '\0' ) break;
        UART0_SendByte(*str++);                     // 发送数据
    }
}

/*****************************************************************************
* 名    称：I2C_Init()
* 功    能：主模式 I2C 初始化，包括初始化其中断为向量 IRQ 中断。
* 入口参数：fi2c          初始化 I2C 总线速率，最大值为 400K
* 出口参数：无
*****************************************************************************/
void  I2C_Init(uint32 fi2c)
{
    if(fi2c>400000) fi2c = 400000;

    PINSEL0 = (PINSEL0 & (~0xF0)) | 0x50;           // 不影响其它管脚连接,设置 I/O 连接到 I2C
    I2SCLH = (Fpclk/fi2c + 1) / 2;                  // 设置 I2C 时钟为 fi2c
    I2SCLL = (Fpclk/fi2c) / 2;
    I2CONCLR = 0x2C;
    I2CONSET = 0x40;                                // 使能主 I2C

    /* 设置 I2C 中断允许 */
```

```
    VICIntSelect = 0x00000000;                  // 设置所有通道为 IRQ 中断
    VICVectCntl0 = 0x29;                        // I2C 通道分配到 IRQ slot 0，即优先级最高
    VICVectAddr0 = (int)IRQ_I2C;                // 设置 I2C 中断向量地址
    VICIntEnable = 0x0200;                      // 使能 I2C 中断
}

/*****************************************************************************
* 名    称：DelayNS()
* 功    能：长软件延时
* 入口参数：dly             延时参数，值越大，延时越久
* 出口参数：无
*****************************************************************************/
void  DelayNS(uint32  dly)
{   uint32  i;

    for(; dly>0; dly--)
    {
        for(i=0; i<5000; i++);
    }
}

/*****************************************************************************
* 名    称：main()
* 功    能：对 ZLG7290 进行操作
* 说    明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)；
*           在 CONFIG.H 文件中包含 I2CINT.H、ZLG7290.H。
*****************************************************************************/
int  main(void)
{   uint8  key_buf[8];
    char   disp_buf[32];
    uint8  key;

    I2C_Init(30000);                            // I2C 配置初始化
    UART0_Init(115200);                         // UART0 配置初始化

    sprintf(disp_buf,"\r\nKey testing!\r\n");
    UART0_SendStr(disp_buf);

    /* 读取按键，将结果通过 UART0 发给 PC */
    while(1)
    {
        DelayNS(5);
        key = 0;
        IRcvStr(ZLG7290, 0x01, key_buf, 8);
```

```
        if(0 == key_buf[1])                    // 是否有效的按键动作
        {
            key = key_buf[0];                  // 取得键值
        }
        sprintf(disp_buf,"This is key %d!\r\n", key);
        UART0_SendStr(disp_buf);
    }
    return(0);
}
```

7. 思考

（1）要提高 I^2C 总线的速度，总线上拉电阻的阻值是要加大还是要减小？

（2）如果主函数要进行按键抖动处理，如何实现？

2.6 SPI 接口实验（选做）

1. 实验目的

通过实验，掌握 SPI 接口的初始化及数据输出/输入控制。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
  自备 LED 显示 SPI 接口板 一个
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境

3. 实验内容

使用硬件 SPI 接口与 74HC595 进行连接，控制 74HC595 驱动 1 个数码管显示。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》第 5.13 节 SPI 接口的说明。

（2）仔细阅读本书第 1 章的内容，了解 SmartARM2200 教学实验开发平台的硬件结构，找出 SPI1 的引出口。

（3）仔细阅读 74HC595 的数据手册，了解如何控制数据移位，锁存数据输出。

5. 实验原理

（1）见图 2.2，SmartARM2210 提供了一个 GPIO 的连接器，SPI1 的信号可以从 J5 引出至子板上。注意 P.20(SSLE1)必须上拉至高电平（3.3V），否则 SPI1 不能工作。

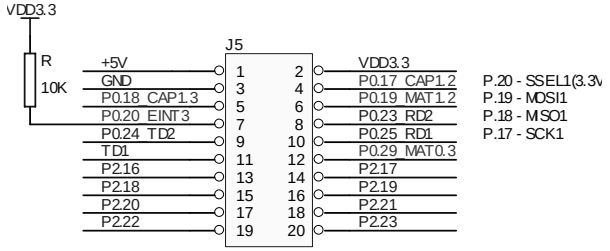


图 2.2 SmartARM2210 连接器 J5

（2）用户可以按照原理图自己准备一个 SPI 接口子板用于 LED 实验。通过 SPI 接口和 LPC2210 相连接，见图 2.3。

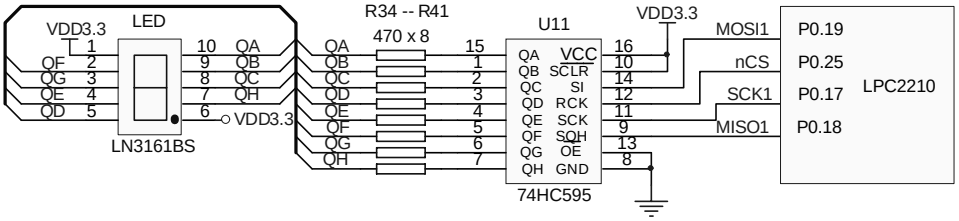


图 2.3 LED 显示 SPI 接口板

6. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 SPIDisp_C。

（2）在 user 组中的 main.c 中编写主程序代码。

（3）选用 DebugInExram 生成目标，然后编译连接工程。

（4）使用杜邦线将 SmartARM2200 教学实验开发平台上的 J5 上 SPI1 信号和 LED 子板的 SPI

接口相连接，并利用教学实验开发平台上的 3.3V 电源给子板供电。

（5）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（6）全速运行程序，观察 LED 的显示变化。

7. 实验参考程序

SPI 接口实验的参考程序见程序清单 2.7。

程序清单 2.7 SPI 接口实验参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功      能：使用硬件 SPI 接口输出控制 LED 显示。(硬件：74HC595 输出控制 LED 显示)
* 说      明：将 SmartARM2200 教学实验开发平台上的 J5 上 SPI1 信号和 LED 子板的 SPI 接口相连接
*****************************************************************************/
#include   "config.h"

#define    HC595_CS          1<<25                  //  P0.25  为片选线

uint8 const DISP_TAB[16] = { //    0      1      2      3      4      5      6      7      8      9
                                  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90,
                             //    A      b      C      d      E      F
                                  0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E};
uint8 rcv_data;

/*****************************************************************************
* 名      称：DelayNS()
```

```
* 功      能：长软件延时
* 入口参数：dly             延时参数，值越大，延时越久
* 出口参数：无
*******************************************************************************/
void   DelayNS(uint32   dly)
{
    uint32   i;
    for(; dly>0; dly--)
    {
        for(i=0; i<5000; i++);
    }
}

/*******************************************************************************
* 名      称：MSpiInit()
* 功      能：初始化 SPI 接口，设置为主机。
* 入口参数：无
* 出口参数：无
*******************************************************************************/
void   MSpiInit(void)
{
    PINSEL1 = (PINSEL1 & ~0x3fc) | 0x2a8;
    S1PCCR = 0x52;                      //  设置 SPI 时钟分频
    S1PCR =   (0 << 3) |                // CPHA = 0,  数据在 SCK  的第一个时钟沿采样
              (1 << 4) |                // CPOL = 1, SCK  为低有效
              (1 << 5) |                // MSTR = 1, SPI  处于主模式
              (0 << 6) |                // LSBF = 0, SPI  数据传输 MSB (位 7)在先
              (0 << 7);                 // SPIE = 0, SPI  中断被禁止
}

/*******************************************************************************
* 名      称：MSendData()
* 功      能：向 SPI 总线发送数据，并接收从机发回的数据。
* 入口参数：data            待发送的数据
* 出口参数：返回值为接收到的数据
*******************************************************************************/
uint8   MSendData(uint8 data)
{
    IO0CLR = HC595_CS;              //  片选
    S1PDR = data;
    while( 0==(S1PSR&0x80) );       //  等待 SPIF 置位，即等待数据发送完毕
    IO0SET = HC595_CS;
    return(S1PDR);
}
```

```
/*****************************************************************************
* 名    称：main()
* 功    能：使用硬件 SPI 接口输出 DISP_TAB 数组的数据，控制 LED 显示。
*****************************************************************************/
int  main(void)
{   uint8  i;

    IO0DIR = HC595_CS;
    MSpiInit();                                    // 初始化 SPI 接口
    while(1)
    {
        for(i=0; i<16; i++)
        {
            rcv_data = MSendData(DISP_TAB[i]);    //  发送显示数据
            DelayNS(10);
        }
    }
    return(0);
}
```

8. 思考

（1）使用 SPI 接口读取从机的数据时，主机为什么要发送数据？
（2）设 SPCR 寄存器的 CPOL=1、CPHA=1，SPI 的数据传输格式是怎样？

2.7 RTC 实验 2

1. 实验目的

掌握 RTC 的在不同系统时钟下的分频设置，掌握 RTC 日期、时间值的设置及读取。

2. 实验设备

- 硬件：PC 机　　一台
  SmartARM2200 教学实验开发平台　　一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件

3. 实验内容

初始化并运行 RTC，然后每隔 1 秒钟读取一次时间值，并通过串口向上位机发送，上位使用 EasyARM 软件的仿真万年历窗口进行显示。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》第 5.17 节实时时钟(RTC)的说明。

（2）仔细阅读本书附录 A 的 EasyARM 软件使用说明，注意通讯协议部分。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 disptimer2。

（2）在 user 组中的 main.c 中编写主程序代码。

（3）选用 DebugInExram 生成目标，然后编译连接工程。

（4）将 SmartARM2200 教学实验开发平台上的 JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（5）使用串口延长线把 SmartARM2200 教学实验开发平台的 CZ2(UART0)与 PC 机的 COM1 连接。PC 机运行 EasyARM 软件，设置串口为 COM1，波特率为 115200，然后选择【功能】->【万年历】，打开仿真万年历窗口。

（6）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（7）全速运行程序，PC 机上的 EasyARM 软件会不断的显示 RTC 的时间值。

6. 实验参考程序

RTC 实验 2 的参考程序见程序清单 2.8。

程序清单 2.8 RTC 实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功    能：运行 RTC 进行计时，并将所时间值不断的通过串口向上位机发送。上位机使用 EasyARM
*           软件，在仿真的万年历显示器上观察结果。
*           通讯波特率 115200，8 位数据位，1 位停止位，无奇偶校验。
* 说    明：
*****************************************************************************/
#include   "config.h"
uint8   const SHOWTABLE[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

/*****************************************************************************
* 名    称：UART0Init()
* 功    能：初始化串口 0。设置为 8 位数据位，1 位停止位，无奇偶校验
* 入口参数：bps 通讯波特率
* 出口参数：无
*****************************************************************************/
void   UART0Init(uint32 bps)
{
    uint16 Fdiv;
    PINSEL0 = (PINSEL0 & (~0x0F)) | 0x05;      // 不影响其它管脚连接,设置 I/O 连接到 UART0
    U0LCR = 0x83;                              // DLAB = 1，可设置波特率
    Fdiv = (Fpclk / 16) / bps;                 // 设置波特率
    U0DLM = Fdiv / 256;
    U0DLL = Fdiv % 256;
    U0LCR = 0x03;
```

```
}

/*****************************************************************************
* 名    称：UART0SendByte()
* 功    能：向串口发送字节数据，并等待发送完毕。
* 入口参数：data  要发送的数据
* 出口参数：无
*****************************************************************************/
void  UART0SendByte(uint8 data)
{
    U0THR = data;                                   // 发送数据
    while( (U0LSR&0x40)==0 );                       // 等待数据发送完毕
}

/*****************************************************************************
* 名    称：PC_DispChar()
* 功    能：向 PC 机发送显示字符。
* 入口参数：no          显示位置
*           chr         显示的字符，不能为 0xff
* 出口参数：无
*****************************************************************************/
void  PC_DispChar(uint8 no, uint8 chr)
{
    UART0SendByte(0xff);
    UART0SendByte(0x81);
    UART0SendByte(no);
    UART0SendByte(chr);
    UART0SendByte(0x00);
}

/*****************************************************************************
* 名    称：SendTimeRtc()
* 功    能：读取 RTC 的时间值，并将读出的时分秒值由串口发送到上位机显示。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  SendTimeRtc(void)
{   uint32  datas;
    uint32  times;
    uint32  bak;

    times = CTIME0;                             // 读取完整时钟寄存器
    datas = CTIME1;
```

```
    bak = (datas>>16)&0xFFF;                // 取得年值
    PC_DispChar(0, SHOWTABLE[bak/1000]);
    bak = bak%1000;
    PC_DispChar(1, SHOWTABLE[bak/100]);
    bak = bak%100;
    PC_DispChar(2, SHOWTABLE[bak/10]);
    PC_DispChar(3, SHOWTABLE[bak%10]);

    bak = (datas>>8)&0x0F;                  // 取得月值
    PC_DispChar(4, SHOWTABLE[bak/10]);
    PC_DispChar(5, SHOWTABLE[bak%10]);

    bak = datas&0x1F;                       // 取得日值
    PC_DispChar(6, SHOWTABLE[bak/10]);
    PC_DispChar(7, SHOWTABLE[bak%10]);

    bak = (times>>24)&0x07;                 // 取得星期值
    PC_DispChar(8, SHOWTABLE[bak]);

    bak = (times>>16)&0x1F;                 // 取得时的值
    PC_DispChar(9, SHOWTABLE[bak/10]);
    PC_DispChar(10, SHOWTABLE[bak%10]);

    bak = (times>>8)&0x3F;                  // 取得分的值
    PC_DispChar(11, SHOWTABLE[bak/10]);
    PC_DispChar(12, SHOWTABLE[bak%10]);

    bak = times&0x3F;                       // 取得秒的值
    PC_DispChar(13, SHOWTABLE[bak/10]);
    PC_DispChar(14, SHOWTABLE[bak%10]);
}

/*****************************************************************************
* 名    称：RTCInit()
* 功    能：初始化实时时钟。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  RTCInit(void)
{
    PREINT = Fpclk / 32768 - 1;             // 设置基准时钟分频器
    PREFRAC = Fpclk - (Fpclk / 32768) * 32768;
    YEAR = 2005;                            // 初化年
    MONTH = 5;                              // 初化月
```

```
    DOM = 01;                           // 初化日
    HOUR = 8;
    MIN = 30;
    SEC = 0;
    CIIR = 0x01;                        // 设置秒值的增量产生一次中断
    CCR = 0x01;                         // 启动 RTC
}

/*****************************************************************************
* 名    称：main()
* 功    能：读取实时时钟的值，并从串口发送出去。
*****************************************************************************/
int  main(void)
{
    RTCInit();                          // 初始化 RTC
    UART0Init(115200);                  // 初始化 UART0

    while(1)
    {
        while( 0==(ILR&0x01) );         // 等待 RTC 增量中断标志
        ILR = 0x01;                     // 清除中断标志
        SendTimeRtc();                  // 读取时钟值,并向 UART0 发送
    }

    return(0);
}
```

7. 思考

若使用 RTC 中断中来读取并发送时钟数据，如何编写程序？

2.8 低功耗实验 2

1. 实验目的

掌握 LPC2000 系列 ARM7 微控制器的低功耗控制。

2. 实验设备

- 硬件：PC 机　　一台
  SmartARM2200 教学实验开发平台　　一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境

3. 实验内容

控制 LPC2210 进入掉电状态，并允许外部中断 1 唤醒。每唤醒一次，LED1～LED4 显

示值加 1，然后再次进入掉电状态。

4. 实验预习要求

仔细阅读《ARM 嵌入式系统基础教程》第 5.4.11 节功率控制的说明。

5. 实验步骤

（1）启动 ADS 1.2，使用 ARM Executable Image for lpc2200 工程模板建立一个工程 PDRun_C。

（2）在 user 组中的 main.c 中编写主程序代码。

（3）在 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈的代码为“MSR CPSR_c, #0x5f”，即使能 IRQ 中断。

（4）选用 DebugInExram 生成目标，然后编译连接工程。

（5）将 SmartARM2200 教学实验开发平台上的 JP12 跳线短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（6）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（7）全速运行程序，LPC2210 进入掉电状态，按下/释放 KEY1，使 EINT3 为低/高电平，即可唤醒 LPC2210。每唤醒一次，LPC2210 控制 LED1～LED4 显示值加 1。

注意：进入掉电状态时调试器会失效。如果需要设定断点，请在中断服务程序或者 while(1)循环中设置断点才能正常调试。如果此程序已经写入 Flash，那么处理器将长期处于掉电状态，如果需要 JTAG 调试，请先将 ISP 短接，复位 MCU 后进行调试。

6. 实验参考程序

低功耗实验 2 的参考程序见程序清单 2.9。

程序清单 2.9 低功耗实验 2 参考程序

```
/*****************************************************************************
* 文 件 名：main.c
* 功     能：系统进入掉电状态，并允许外部中断 3 唤醒。每唤醒一次，LED1--LED4 显示值加 1。
* 说     明：将跳线器 JP12 短接。
*****************************************************************************/
#include   "config.h"

#define    LEDCON      0xf0000000

/*****************************************************************************
* 名     称：IRQ_Eint3()
* 功     能：外部中断 EINT3 服务函数，取反 B1 控制口。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void    __irq IRQ_Eint3(void)
{
    while((EXTINT&1<<3) != 0)
    {
```

```
        EXTINT = 1<<3;                  // 清除 EINT3 中断标志，1<<3 等价于 0x08
    }

    VICVectAddr = 0;                    // 向量中断结束
}

/*****************************************************************************
* 名    称：InitEint3()
* 功    能：初始化外部中断 1，使能 IRQ 中断。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  InitEint3(void)
{
    VICIntSelect = 0x00000000;          // 设置所有中断分配为 IRQ 中断
    VICVectCntl0 = 0x20 | 17;
    VICVectAddr0 = (int)IRQ_Eint3;      // 设置中断服务程序地址
    EXTWAKE = 1<<3;                     // 允许外部中断 3 唤醒掉电的 CPU
    EXTMODE = 0<<3;                     // EINT3 电平触发
    EXTINT = 1<<3;                      // 清除 EINT3 中断标志
    VICIntEnable = 1<<17;               // 使能 EINT3 中断，EINT3 在 Bit17 上
}

/*****************************************************************************
* 名    称：main()
* 功    能：掉电测试。
* 说    明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)。
*****************************************************************************/
int  main(void)
{   uint8  count;

    IO2DIR = LEDCON;

    PINSEL1 = 3<<8;                     // 设置管脚连接，P0.20 设置为 EINT3
    InitEint3();                        // 初始化外部中断 1，使能 IRQ 中断

    PCONP = 1<<11;                      // 关闭片内外设(使用外部程序存储器，PCEMC 为 1)

    count = 0;
    while(1)
    {
        IO2SET = 0xffffffff;            // 在 LED 上显示恢复运行的次数
        IO2CLR = (count&0x0000000f)<<28;
        PCON = 0x02;                    // 系统进入掉电模式
```

```
        count++;
    }
    return(0);
}
```

7. 思考

（1）掉电状态下和空闲状态下，哪一个功耗更低？为什么？

第3章 基于 μC/OS-II 的基础实验

3.1 SPI 总线的 LED 控制应用

1. 实验目的

掌握 SPI 总线驱动中间件在 LED 数码管应用。

2. 实验设备

- 硬件： PC 机 一台
  SmartARM2200 教学实验开发平台 一套
  自备 LED 显示 SPI 接口板 一个
- 软件： Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)，SPI 总线驱动中间件

3. 实验内容

使用硬件 SPI 接口与 74HC595 进行连接，任务 0 和任务 1 同时分别控制 74HC595 驱动 1 个数码管显示。

4. 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》SPI 接口相关说明。
（2）了解 SmartARM2200 教学实验开发平台的硬件结构，找出 SPI1 的引出口。
（3）仔细阅读 74HC595 的数据手册，了解如何控制数据移位，锁存数据输出。
（4）了解 SPI 中间件的函数调用方法。

5. 实验原理

（1）见图 3.1，SmartARM2210 提供了一个 GPIO 的跳线座，SPI1 的信号可以从 J5 引出至子板上。注意 P.20(SSLE1)必须上拉至高电平（3.3V），否则 SPI1 不能工作。

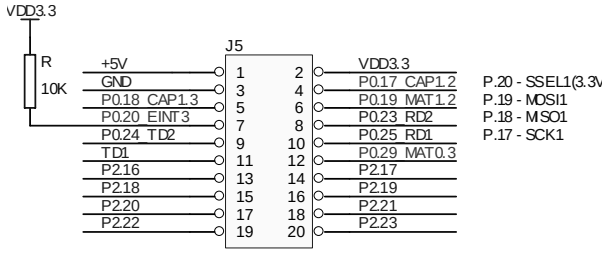


图 3.1 SmartARM2210 跳线座 J5

（2）用户可以按照原理图自己准备一个 SPI 接口子板用于 LED 实验。通过 SPI 接口和 LPC2210 相连接，见图 3.2。

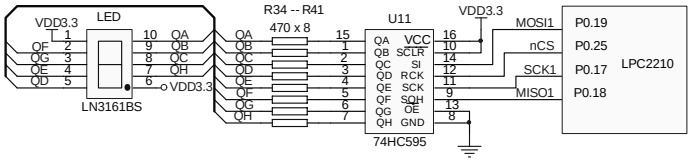


图 3.2 LED 显示 SPI 接口板

范例程序有 2 个任务，任务 0 每 0.5 秒一次更新，循环显示字符 0~F 在 LED 上，任务

1 每秒一次显示字符 8 在 LED 上，如图 3.1 所示。以验证 SPI 中间件可以在 uCOS II 下能正常运行。

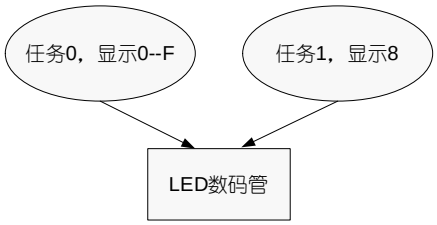


图 3.3 SPI 控制原理

6. 实验步骤

（1）利用光盘提供的模板新建 ARM uCOS II 工程，添加 SPI 总线驱动中间件进入工程。SPI 总线驱动中间件的 spi.c、spi.h 文件见产品光盘。

（2）在工程的 target 组的 irq.s 文件最后，增加 SPI 中断服务程序的汇编语言部分代码“SPI_Handler HANDLER SPI_Exception”。

程序清单 3.1 SPI 中断汇编部分代码

```
;/*SPI 中断*/
SPI_Handler    HANDLER SPI_Exception
```

（3）在工程的 target 组的 target.c 文件中的 VICInit 函数，添加 SPI 向量中断的初始化代码，如所示。

程序清单 3.2 SPI 向量中断初始化

```
extern   void   SPI_Handler(void);
VICVectAddr13 = (uint32)SPI_Handler;
VICVectCntl13 = (0x20 | 11);
VICIntEnable = 1 << 11;
```

（4）在工程的 target 组的 target.c 文件中的 TargetInit()函数，添加初始化 SPI 的代码“SPIInit();”。

（5）在工程的*.h 组的 config.h 文件中，添加配置代码，见程序清单 3.3。

程序清单 3.3 SPI 总线的配置

```
#include    "spi.h"
#define     HC595_CS          1<<25                    // P0.25 口为 74HC595 的片选
#define     SPI_MOD     SPI_CPHA_ONE | SPI_CPOL_HIGH | SPI_LSBF_BIT7   // SPI 模式
extern      uint32 GetOSPrioCur(void);
```

（6）选用 DebugInExram 生成目标，然后编译连接工程。

（7）将 SmartARM2200 教学实验开发平台上的 J5 上 SPI1 信号和 LED 子板的 SPI 接口相连接，并利用教学实验开发平台上的 3.3V 电源给子板供电。

（8）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（9）全速运行程序，观察 LED 显示是否正确，两个任务是否都达到显示的目的。

7. 实验参考程序

SPI 总线实验的参考程序见程序清单 3.4。

程序清单 3.4 LED 控制主程序和任务程序

```
#include "config.h"
#include "stdlib.h"
#define     TaskStkLengh    512          // 定义用户任务堆栈长度
OS_STK    Task0Stk [TaskStkLengh];   // 定义用户任务 0 的堆栈
OS_STK    Task1Stk [TaskStkLengh];   // 定义用户任务 1 的堆栈

uint8 const DISP_TAB[16] = {    //  0     1      2     3     4     5     6     7     8     9
                                  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90,
                                //  A     b     C     d     E     F
                                  0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E};

void  Task0(void *pdata);                 //Task0  任务 0
void  Task1(void *pdata);                 //Task0  任务 1
uint32    GetOSPrioCur(void);

/* 程序主函数 */
int main (void)
{
      OSInit ();
      OSTaskCreate (Task0,(void *)0, &Task0Stk[TaskStkLengh - 1], 2);
      OSTaskCreate (Task1,(void *)0, &Task1Stk[TaskStkLengh - 1], 3);
      OSStart ();
      return 0;
}

/* 任务 0 ，每 0.5 秒一次更新，循环显示字符 0~F 在 LED 上*/
void Task0 (void *pdata)
{    uint8 temp,i;

      pdata = pdata;

     TargetInit();                        // 系统初始化
     IO0DIR = HC595_CS;                   // 初始化片选 I/O

      while (1)
      {
          SPIStart();
          IO0CLR = HC595_CS;
          SPIRW(&temp, DISP_TAB[i&0xf]);
          IO0SET = HC595_CS;
```

```
            SPIEnd();
            OSTimeDly(OS_TICKS_PER_SEC/2);
            i++;
        }
}

/* 任务 1，每秒一次显示字符 8 在 LED 上 */
void Task1 (void *pdata)
{   uint8 temp;

    pdata = pdata;

    while (1)
    {
        SPIStart();
        IO0CLR = HC595_CS;
        SPIRW(&temp, DISP_TAB[8]);
        IO0SET = HC595_CS;
        SPIEnd();
        OSTimeDly(OS_TICKS_PER_SEC);
    }
}
```

8． 思考

（1）SPI 中间件在两个任务中可以正确的运行说明了什么？

3.2 I^2C 总线的 EEPROM 应用

1． 实验目的

掌握 I^2C 总线驱动中间件在 CAT1025 EEPROM 读写的使用方法。

2． 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件
  μC/OS-II 操作系统(V2.52)，I^2C 总线驱动中间件

3． 实验内容

使用 I^2C 总线驱动中间件实现对 CAT1025 的 EEPROM 的操作，把结果打印在 EasyARM 模拟 DOS 窗口。

4． 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》I^2C 接口相关说明。

（2）了解 SmartARM2200 教学实验开发平台的 I2C 硬件结构。

（3）仔细阅读 CAT1025 的数据手册。

（4）了解 I2C 中间件的函数调用方法。

5. 实验原理

通过两个相互独立的任务，分别写入和读出计数器的值。以验证 I2C 中间件可以在 uCOS II 下能正常运行。

6. 实验步骤

（1）利用光盘提供的模板新建 ARM uCOS II 工程，添加 I2C 总线驱动中间件进入工程。I2C 总线驱动中间件的 i2c.c、i2c.h 文件见产品光盘。

（2）在工程的 target 组的 irq.s 文件最后，增加 I^2C 中断服务程序的汇编语言部分代码，见程序清单 3.5。

程序清单 3.5 I^2C 中断汇编部分代码

```
;/* I2C 中断 */
I2c_Handler    HANDLER    I2c_Exception
```

（3）在工程的 target 组的 target.c 文件中的 VICInit 函数，添加 I^2C 向量中断的初始化代码，见程序清单 3.6。

程序清单 3.6 I^2C 向量中断初始化

```
extern void I2c_Handler(void);
VICVectAddr12 = (uint32)I2c_Handler;
VICVectCntl12 = (0x20 | 9);
```

（4）在工程的 target 组的 target.c 文件中的 TargetInit()函数，添加初始化 I^2C 的代码“I2cInit();”。

（5）在工程的*.h 组的 config.h 文件中，添加配置代码，见程序清单 3.7。

程序清单 3.7 I^2C 总线的配置

```
#include    "i2c.h"
#define     CAT1025          0xA0          /* CAT1025 从 I2C 地址 */
```

（6）选用 DebugInExram 生成目标，然后编译连接工程。

（7）将 SmartARM2200 教学实验开发平台上的跳线器 JP6 短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（8）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（9）全速运行程序，观察 EasyARM 模拟 DOS 窗口的提示，见图 3.4。



图 3.4　CAT1025 读写提示

7．　实验参考程序

I2C 总线实验的参考程序见程序清单 3.8。

程序清单 3.8　CAT1025 主程序和任务程序

```
#include "config.h"
#include "stdlib.h"

#define     TaskStkLengh    512
OS_STK    Task0Stk [TaskStkLengh];      // 定义用户任务 0 的堆栈
OS_STK    Task1Stk [TaskStkLengh];      // 定义用户任务 1 的堆栈

uint8 flag = 1;

void  Task0(void *pdata);                 // 任务 0
void  Task1(void *pdata);                 // 任务 1

/* 程序主函数 */
int main(void)
{
    OSInit ();
```

```
    OSTaskCreate (Task0,(void *)0, &Task0Stk[TaskStkLengh - 1], 2);
    OSTaskCreate (Task1,(void *)0, &Task1Stk[TaskStkLengh - 1], 3);
    OSStart ();
    return 0;
}

/* 任务 0，写计数器值进入 EEPROM */
void Task0(void *pdata)
{   char vendor[] = {"ZLG-MCU CO., LTD. Guangzhou China"};
    char tag[]    = {"CAT1025 EEPROM Example Base On Real Time Kernel"};

    uint32 i;
    char    disp_buf[32];
    uint8   dat_buf[32];
    uint8   counter = 0;                        // 定义计数器

    TargetInit();
    PC_DispClrScr (DISP_FGND_YELLOW+DISP_BGND_BLUE);
    PC_DispStr(24, 24, vendor, DISP_FGND_YELLOW+DISP_BGND_BLUE);
    PC_DispStr(20, 00, tag, DISP_FGND_YELLOW+DISP_BGND_BLUE);
    while (1)
    {
        dat_buf[0] = 0;
        dat_buf[1] = counter;
        counter++;
        I2cWrite(CAT1025, dat_buf, 2);          // 写入计数器的值
        for(i=0; i<5000; i++);                  // 延时
        /* 打印已经写入的计数器值 */
        sprintf(disp_buf, " ** Task0 is writing: %3d ** ", counter);
        PC_DispStr(28, 10, disp_buf, DISP_FGND_YELLOW+DISP_BGND_BLUE);
        OSTimeDly(OS_TICKS_PER_SEC);
    }
}

/* 任务 1，读取 EEPROM 的计数器值 */
void Task1(void *pdata)
{   char  disp_buf[32];
    uint8 dat_buf[32];

    pdata = pdata;

    while (1)
    {
        dat_buf[0] = 0;
```

```
            dat_buf[1] = 0;
            I2cRead(CAT1025, dat_buf, dat_buf, 1, 1);     // 读出计数器的值
            /* 打印已经读出的计数器值 */
            sprintf(disp_buf, " ** Task1 is reading: %3d ** ", dat_buf[0]);
            PC_DispStr(28, 11, disp_buf, DISP_FGND_YELLOW+DISP_BGND_BLUE);
            OSTimeDly(OS_TICKS_PER_SEC/2);
        }
}
```

8． 思考

（1）本实验范例两个任务有可能同时对 I^2C 总线操作，I^2C 中间件是如何解决这个问题的？

（2）写入 EEPROM 后需要延时，任务优先级，这 2 点都是实验的关键点，请说明为什么？

3.3 I^2C 总线的 ZLG7290 应用

1． 实验目的

掌握 I^2C 总线驱动中间件在 ZLG7290 的使用方法。

2． 实验设备

- 硬件：PC 机　　一台
  SmartARM2200 教学实验开发平台一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  EasyARM 软件
  μC/OS-II 操作系统(V2.52)，I^2C 总线驱动中间件

3． 实验内容

使用 I^2C 总线驱动中间件实现对 ZLG7290 的 EEPROM 的操作。

4． 实验预习要求

（1）仔细阅读《ARM 嵌入式系统基础教程》I^2C 接口相关说明。
（2）了解 SmartARM2200 教学实验开发平台的 I^2C 硬件结构。
（3）仔细阅读 ZLG7290 的数据手册。
（4）了解 I^2C 中间件的函数调用方法。

5． 实验原理

利用 I^2C 中间件，读取 ZLG7290 的键盘键值，并打印在把结果打印在 EasyARM 模拟 DOS 窗口。

6． 实验步骤

（1）利用光盘提供的模板新建 ARM uCOS II 工程，将 I2C 总线和 ZLG7290 驱动添加入工程。I2C 总线驱动中间件的 i2c.c、i2c.h 和 ZLG7290 驱动 zlg7290.c、zlg7290.h 见产品光盘。

2）在工程的 target 组的 irq.s 文件最后，增加 I²C 中断服务程序的汇编语言部分代码，见程序清单 3.9。

程序清单 3.9 I²C 中断汇编部分代码

```
;/* I2C 中断 */
I2c_Handler    HANDLER    I2c_Exception
```

（3）在工程的 target 组的 target.c 文件中的 VICInit 函数，添加 I²C 向量中断的初始化代码，见程序清单 3.10。

程序清单 3.10 I²C 向量中断初始化

```
extern void I2c_Handler(void);
VICVectAddr12 = (uint32)I2c_Handler;
VICVectCntl12 = (0x20 | 9);
```

（4）在工程的 target 组的 target.c 文件中的 TargetInit()函数，添加初始化 I²C 的代码“I2cInit();”。

（5）在工程的*.h 组的 config.h 文件中，添加配置代码，见程序清单 3.11。

程序清单 3.11 I²C 总线的配置

```
#include    "i2c.h"
#include    "zlg7290.h"
#define     CAT1025          0xA0              /* CAT1025 从 I2C 地址 */
```

（6）选用 DebugInExram 生成目标，然后编译连接工程。

（7）将 SmartARM2200 教学实验开发平台上的跳线器 JP6 短接。JP9 设置为 OUTSIDE，JP10 跳线设置为 Bank0-RAM、Bank1-Flash。

（8）选择【Project】->【Debug】，启动 AXD 进行 JTAG 仿真调试。

（9）全速运行程序，观察 EasyARM 模拟 DOS 窗口的提示，见图 3.5。

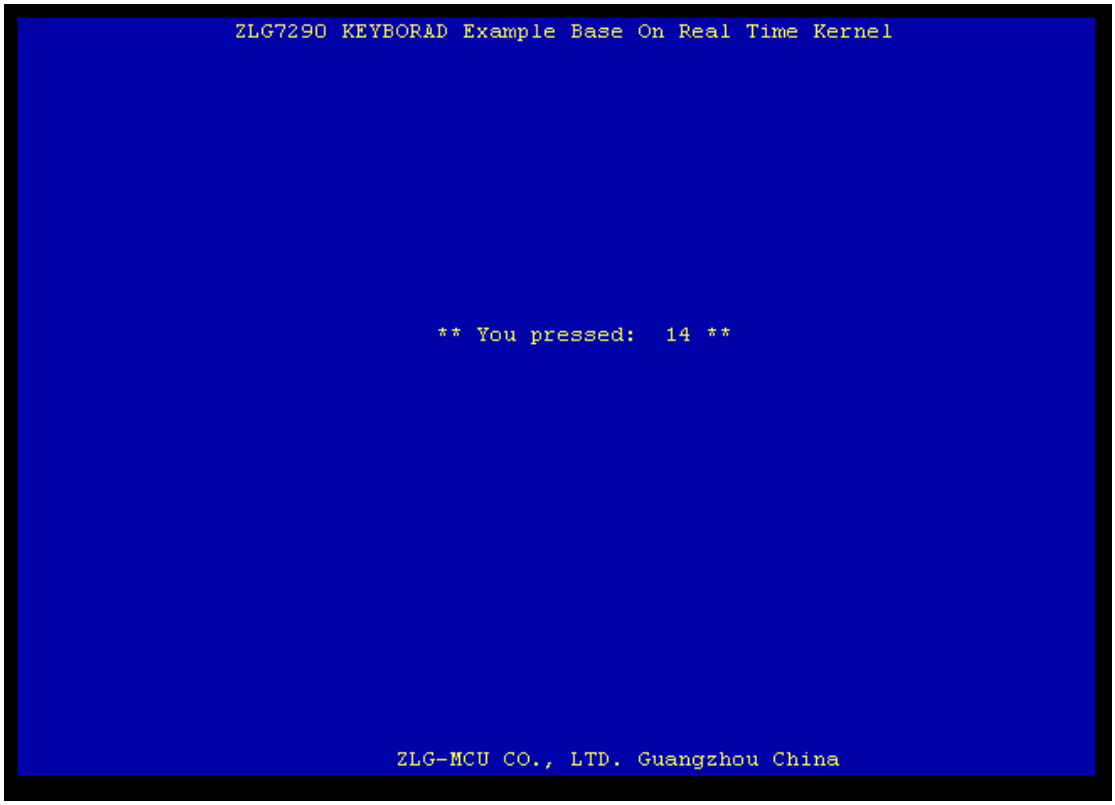


图 3.5　ZLG7290 键值提示

7．　实验参考程序

I^2C 总线实验的参考程序见程序清单 3.8。

程序清单 3.12　ZLG7290 主程序和任务程序

```
#include "config.h"
#include "stdlib.h"

#define     TaskStkLengh    512
OS_STK    Task0Stk [TaskStkLengh];        // 定义用户任务 0 的堆栈
OS_STK    Task1Stk [TaskStkLengh];        // 定义用户任务 1 的堆栈

void  Task0(void *pdata);                 // 任务 0
void  Task1(void *pdata);                 // 任务 1

int main(void)
{
    OSInit ();
    OSTaskCreate (Task0,(void *)0, &Task0Stk[TaskStkLengh - 1], 2);
    OSTaskCreate (Task1,(void *)0, &Task1Stk[TaskStkLengh - 1], 3);
    OSStart ();
    return 0;
```

```
}

void Task0(void *pdata)
{    char vendor[] = {"ZLG-MCU CO., LTD. Guangzhou China"};
     char tag[]      = {"ZLG7290 KEYBORAD Example Base On Real Time Kernel"};
     char   disp_buf[32];

     pdata = pdata;

      TargetInit();
      PC_DispClrScr (DISP_FGND_YELLOW+DISP_BGND_BLUE);
      PC_DispStr(28, 24, vendor, DISP_FGND_YELLOW+DISP_BGND_BLUE);
      PC_DispStr(16, 00, tag, DISP_FGND_YELLOW+DISP_BGND_BLUE);

      while (1)
      {
          sprintf(disp_buf, " ** You pressed: %3d ** ", ZLG7290_Key());
          PC_DispStr(30, 10, disp_buf, DISP_FGND_YELLOW+DISP_BGND_BLUE);
          OSTimeDly(OS_TICKS_PER_SEC/10);
      }
}

void Task1(void *pdata)
{
     pdata = pdata;

      while (1)
      {
          OSTimeDly(OS_TICKS_PER_SEC);
      }
}
```

8． 思考

（1）请在任务 1 这个空任务中添加一个对其他 I^2C 控制的代码，看看任务 0 是否可以正常运行？为什么？

3.4 LPC2000 系列微控制器 MODEM 接口软件包

3.4.1 概述

LPC2000 系列 AMR7 微控制器的 UART1 具有 MODEM 接口功能，通过 RS232 电平转换即可与 MODEM 连接，进行 MODEM 通讯控制操作。

本文介绍使用 LPC2000 系列 AMR7 微控制器的 UART1 驱动 MODEM，驱动程序使用了 AT 命令集的命令进行控制操作，所以驱动的 MODEM 要支持 AT 命令集。

3.4.2 软件包的使用

1. MODEM 状态值

在调用 MODEM 接口软件包的驱动函数时，大部份函数会返回当前的 MODEM 状态，比如 ModemInit()、GetModemState()等函数，用户可以根据这些返回值作出相应的处理。MODEM 状态值定义如下：

NOT_INIT_MODEM ：值为 0x00，表示 MODEM 未初始化；

NOT_FIND_MODEM ：值为 0x01，表示没有发现 MODEM；

MODEM_CLOSE ：值为 0x02，表示 MODEM 已关闭，即 MODEM 没有建立连接；

MODEM_RING ：值为 0x03，表示 MODEM 响铃；

MODEM_CONNECT ：值为 0x05，表示 MODEM 已连接，即接收到数据载波信号。

2. 软件包接口说明

MODEM 接口软件包采用中断方式进行处理，提供了 6 个接口函数，分别见表 3.1～表 3.6。由于 UART1(即 MODEM 接口)向量中断需要根据实际应用来设定(即 VIC 的设置)，**在调用 MODEM 接口软件包接口函数前，用户程序要设置好 UART1 向量中断。**

表 3.1 ModemInit()函数

函数名称	ModemInit	所属文件	modem.c
函数原型	uint8 ModemInit(uint32 bps)		
功能描述	初始化 MODEM。函数会先初始化 UART1。		
函数参数	bps： 串口波特率		
函数返回值	MODEM 状态值		
特殊说明和注意点	只需要在系统初始化时调用一次。它不包含向量中断控制器部分的配置，这部分需要自己编写，且向量中断控制器必须允许 UART1 中断。		

范例	if(ModemInit (115200)!=MODEM_CLOSE)
	{
	/* 出错处理 */
	}		

表 3.2 ModemDialUp()函数

函数名称	ModemDialUp	所属文件	modem.c
函数原型	uint8 ModemDialUp(char Number[])		
功能描述	MODEM 拨号操作(建立连接)		
函数参数	Number： 拨号串，即拨号修饰符及号码字符串		
函数返回值	MODEM 状态值		
特殊说明和注意点	必须先初始化 MODEM 及 UART1 向量中断控制器		

范例	if(ModemDialUp ("#2")!= MODEM_CONNECT)
	{
	/* 出错处理 */
	}

表 3.3 ModemWrite ()函数

函数名称	ModemWrite	所属文件	modem.c
函数原型	uint8 ModemWrite(char *Data, uint16 NByte)		
功能描述	发送多个字节数据		
函数参数	Data 发送数据的存储位置 NByte 发送数据的个数		
函数返回值	MODEM 状态值		
特殊说明和注意点	必须先初始化 MODEM 及 UART1 向量中断控制器，并已拔号建立连接		
范例	ModemWrite("12345\n", 6);		

表 3.4 ModemGetch()函数

函数名称	ModemGetch	所属文件	modem.c
函数原型	uint8 ModemGetch(void)		
功能描述	接收一个字节数据		
函数参数	无		
函数返回值	接收到的数据		
特殊说明和注意点	必须先初始化 MODEM 及 UART1 向量中断控制器，并已拔号建立连接		
范例	Ch = ModemGetch();		

表 3.5 ModemDialDown()函数

函数名称	ModemDialDown	所属文件	modem.c
函数原型	uint8 ModemDialDown(void)		
功能描述	MODEM 断开连接		
函数参数	无		
函数返回值	MODEM 状态值		

特殊说明 和注意点	无
范例	ModemDialDown();

表 3.6 GetModemState ()函数

函数名称	GetModemState	所属文件	modem.c
函数原型	uint8 GetModemState(void)		
功能描述	取得当前 MODEM 状态值		
函数参数	无		
函数返回值	MODEM 状态值		
特殊说明 和注意点	无		

范例	if(GetModemState ()==MODEM_CLOSE)
	{
	/* 建立连接处理 */
	}		

3. 使用实例

使用 LPC2000 系列微控制器 MODEM 接口软件包时，需要将 modem.c、modem.h 文件复制到项目的目录中，然后在工程中添加 modem.c 文件，并在用户程序上使用“#include "modem.h"”包含头文件。

MODEM 接口软件包应用的例子如程序清单 3.13 所示。

注意：若使用 LPC2100/LPC2200 专用工程模板建立工程，由于工程模板默认是关断 IRQ 中断的，所以要在工程中 Startup.s 文件的 InitStack 子程序中，修改设置系统模式堆栈处的代码为“MSR CPSR_c, #0x5f”，即使能 IRQ 中断。

程序清单 3.13 MODEM 接口应用实例

```
/*****************************************************************************
* 文件名：TEST.C
* 功能：MODEM 接口应用实例。使用 UART1 发送 AT 指令控制 MODEM 拨号(#2)，
*       连接成功后发送数据"EasyARM2200---MODEM"。
* 说明：
*****************************************************************************/
#include "config.h"

extern void   DelayNS(uint32   dly);
extern void   UART1_Exception(void);
/*****************************************************************************
* 名称：main()
* 功能：主函数。控制 MODEM 拨号，并发送数据。
```

```
* 说明：在 STARTUP.S 文件中使能 IRQ 中断(清零 CPSR 中的 I 位)。
*********************************************************************************/
int   main(void)
{    uint8   i;

     VICIntSelect = 0x00000000;              // 设置所有通道为 IRQ 中断
     VICVectCntl0 = 0x27;                    // UART1 中断通道分配到 IRQ slot 0，即优先级最高
     VICVectAddr0 = (uint32)UART1_Exception;  // 设置 UART1 向量地址
     VICIntEnable = 0x00000080;              // 使能 UART1 中断

     ModemInit(115200);
     ModemDialUp("#2");
     for(i=0; i<10; i++)
     {
          ModemWrite("EasyARM2200---MODEM\r\n", 21);
     }

     DelayNS(5);
     ModemDialDown();

     while(1);
     return(0);
}
```

3.4.3 设计原理

1. MODEM 状态变量

为了正确的处理 MODEM 各种事件，程序定义一个静态全局变量 ModemState(在 modem.c 文件中定义)，用来表示当前的 MODEM 所处状态。当初始化 MODEM 或有 MODEM 事件中断时，将会设置 ModemState 状态值，以便于其它模块跟据状态进行相应的处理。

```
static volatile uint8   ModemState=NOT_INIT_MODEM;      // 定义 MODEM 状态变量
```

2. 初始化 MODEM

在使用 MODEM 前先要进行初始化操作，实始化函数 ModemInit()的代码见程序清单 3.14，关于 AT 指令的详细介绍请参考 AT 指令集的相关资料。

UART1Init()的代码见程序清单 3.15。

程序清单 3.14 初始化 MODEM

```
/*********************************************************************************
* 名称：ModemInit()
* 功能：初始化 MODEM。
* 入口参数：bps             串口波特率
* 出口参数：返回当前 MODEM 状态值
*********************************************************************************/
uint8   ModemInit(uint32 bps)
```

```
{
    ModemState = MODEM_CLOSE;              // 设置 MODEM 已关闭状态
    UART1Init(bps);                        // 初始化 UART1

    if((U1MSR & 0x30) != 0x30)             // 判断 DSR、CTS 是否有效
    {
        DelayNS(1);
        if((U1MSR & 0x30) != 0x30)
        {
            ModemState = NOT_FIND_MODEM;   // 没有发现 MODEM
        }
    }
    if(ModemState == MODEM_CLOSE)          // 进行 MODEM 初始化
    {
        ModemCommand("ATE0");              // 关闭命令回显
        ModemCommand("ATV0");              // 以数字形式返回结果码
        ModemCommand("AT&C1");             // 数据载波检测(DCD)选择有效
        ModemCommand("AT&D2");             // 数据终端准备就绪(DTR)选择，当 DTR 由 ON-OFF 时，
                                           // MODEM 将挂机，并返回命令状态
        ModemCommand("AT&R0");             // 请求发送(RTS)/清除发送(CTS)选择，当 MODEM 在线
                                           // 时，CTS 跟随 RTS 的变化
        ModemCommand("AT&S0");             // 数据设备就绪(DSR)选择，DSR 一直有效
        ModemCommand("ATS0=2");            // 自动摘机应答设置，响铃 2 次后 MODEM 自动摘机
    }

    return(ModemState);
}
```

程序清单 3.15 UART1 初始化—MODEM 接口

```
/*****************************************************************************
* 名称：UART1Init()
* 功能：UART1 初始化(MODEM 功能)。
* 入口参数：bps              波特率
* 出口参数：无
*****************************************************************************/
void   UART1Init(uint32 bps)
{
    uint16 Fdiv;

    PINSEL0 = (PINSEL0 & 0x0000FFFF) | 0x55550000;   // 选择管脚为 UART0

    U1LCR = 0x80;                          // 允许访问分频因子寄存器
    Fdiv = (Fpclk / 16) / bps;             // 设置波特率
```

```
    U1DLM = Fdiv / 256;
    U1DLL = Fdiv % 256;
    U1LCR = 0x03;                           // 禁止访问分频因子寄存器
                                            // 且设置为 8,1,n
    U1IER = 0x0D;                           // 允许接收和 MODEM 中断
    U1FCR = 0x87;                           // 初始化 FIFO
    U1MCR = 0x03;
}
```

3. 发送 MODEM 命令

发送 MODEM 命令使用 ModemCommand()函数，其代码如程序清单 3.16 所示。发送 AT 指令前，首先要判断 MODEM 是否处理命令状态，即 ModemState 状态值是否为 MODEM_CLOSE，若不是则直接退出。在发送 MODEM 命令之前先读取 UART1 所有已接收到 FIFO 的数据，以便准确接收 AT 指令的结果码，然后再发送 AT 指令，并等待 MODEM 回应。

ModemWrite()的代码如程序清单 3.19 所示。

程序清单 3.16 ModemCommand()代码

```
/*****************************************************************************
* 名称：ModemCommand()
* 功能：发送 MODEM 命令。
* 入口参数：Command          AT 指令(字符串)
* 出口参数：返回当前 MODEM 状态值
*****************************************************************************/
uint8   ModemCommand(char   *Command)
{
    char   *cp;
    uint8   i, err;

    if(ModemState == MODEM_CLOSE)
    {
        while((U1LSR&0x01) != 0)            // 读取完接收到的数据
        {
            err = U1RBR;
        }

        cp = Command;
        i = 0;
        while(*cp++ != 0)                   // 取得 AT 指令字符个数
        {
            i++;
        }
        ModemWrite(Command, i);             // 发送 AT 指令
        ModemWrite("\r", 1);                // 发送命令结束符
```

```
        i = ModemGetch();
        if((i=='A') || (i=='a'))                    // 若命令回显，接字符形式等待回应
        {
            while(1)
            {
                err = i;
                i = ModemGetch();
                if((err=='O') || (err=='o'))
                if((i=='K') || (i=='k'))
                {
                    i = ModemGetch();
                    i = ModemGetch();
                    break;
                }
            }
        }
        else                                        // 按数字形式等待回应
        {
            while(1)
            {
                if(i == '0')
                {
                    i = ModemGetch();
                    break;
                }
                i = ModemGetch();
            }
        }
    }

    return(ModemState);
}
```

4. MODEM 拔号建立连接

程序首先判断 MODEM 是否处于未建立连接状态，若已建立连接，则不再拔号操作，否则进行拔号操作，建立连接。在拔号之前先读取 UART1 所有已接收到 FIFO 的数据，以便准确接收 AT 指令的结果码，然后再使用 AT 指令控制拔号，并等待 MODEM 回应。代码如程序清单 3.17 所示。

程序清单 3.17　ModemDialUp()代码

```
/*****************************************************************************
* 名称：ModemDialUp()
* 功能：Modem 拔号操作。
```

```
* 入口参数：Number        拔号字符串
* 出口参数：返回当前 MODEM 状态值
*******************************************************************************/
uint8   ModemDialUp(char Number[])
{
    char    *cp;
    uint8   i, err;

    if(ModemState == MODEM_CLOSE)
    {
        while((U1LSR & 0x00000001) != 0)       // 读取完接收到的数据
        {
            err = U1RBR;
        }
        ModemWrite("ATD", 3);                  // 发送拔号命令
        i = 0;
        cp = Number;
        while(*cp++ != 0)                      // 取得 AT 指令字符个数
        {
            i++;
        }
        ModemWrite(Number, i);                 // 发送拔号号码
        ModemWrite("\r", 1);                   // 发送命令结束符

        U1IER = U1IER | 0x01;                  // 允许接收中断
        for (i = 0; i < 200; i++)
        {
            DelayNS(5);
            if (ModemState == MODEM_CONNECT)
            {
                break;
            }
        }
    }

    return(ModemState);
}
```

5. UART1 的 MODEM 接口中断服务程序

使用 MODEM 控制时，需要 UART1 中断中处理各种状态变化，比如数据载波 DCD、响铃信号 RI 等等，程序会设置 ModemState 状态值。UART1 的 MODEM 接口中断服务程序见程序清单 3.18。

程序清单 3.18 MODEM 接口中断服务程序

```
/*****************************************************************************
* 名称：UART1_Exception()
* 功能：UART1 中断服务程序。
* 入口参数：无
* 出口参数：无
* 说明：
*****************************************************************************/
void __irq   UART1_Exception(void)
{
    uint8 IIR, temp;

    while(((IIR=U1IIR) & 0x01) == 0)
    {                                       // 有中断未处理完
        switch (IIR & 0x0e)
        {
            case 0x00:                      // Modem 状态变化中断
                if ((U1MSR & 0x80) != 0)
                {
                    ModemState = MODEM_CONNECT;
                }
                else
                {
                    ModemState = MODEM_CLOSE;
                }
                if ((U1MSR & 0x40) != 0)
                {
                    ModemState = MODEM_RING;
                }
                if ((U1MSR & 0x30) != 0x30)
                {
                    ModemState = NOT_FIND_MODEM;
                }
                break;

            case 0x04:                      // 接收数据可用
                U1IER &= (~0x01);           // 禁止接收及字符超时中断
                break;

            case 0x06:                      // 接收线状态
                temp = U1LSR;
                break;

            case 0x0c:                      // 字符超时指示
```

```
                U1IER &= (~0x01);           // 禁止接收及字符超时中断
                break;

            default:
                break;
        }
    }

    VICVectAddr = 0;            // 通知中断控制器中断结束
}
```

6. 发送/接收数据

当建立连接后，使用 UART1 发送的数据即可通过 MODEM 发送到远程中端出，发送数据函数很简单，如程序清单 3.19 所示。发送数据时，由于使用了发送 FIFO，所以一次最多可以发送 16 个字节数据。

程序清单 3.19　ModemWrite()代码

```
/*****************************************************************************
* 名称：ModemWrite()
* 功能：向 UART1(MODEM)发送多字节数据。
* 入口参数：Data          待发送数据指针
*           NByte         发送数据字节个数
* 出口参数：返回当前 MODEM 状态值
*****************************************************************************/
uint8   ModemWrite(char *Data, uint16 NByte)
{
    uint8 i;

    while (NByte > 0)                       // 发送 NByte 字节数据
    {
        for (i = 0; i < 8; i++)             // 一次发送 8 个字节(已使能 FIFO)
        {
            U1THR = *Data++;
            NByte--;
            if (NByte == 0)
            {
                break;
            }
        }
        while((U1LSR&0x20) == 0);           // 等待数据发送完毕
    }

    return(ModemState);
}
```

接收数据的 ModemGetch()函数代码见程序清单 3.20，若接收 FIFO 中已有数据，则直接读取接收到的数据(读取 U1RBR 寄存器)；倘若接收 FIFO 没有数据，则等待串口接收到一字节数据。

程序清单 3.20 ModemGetch()代码

```
/*****************************************************************************
* 名称：ModemGetch()
* 功能：接收 UART1(MODEM)字节数据。
* 入口参数：无
* 出口参数：返回取得的数据
*****************************************************************************/
uint8   ModemGetch(void)
{
    uint8 rt;

    while((U1LSR&0x01) == 0)
    {                                       // 没有收到数据
        U1IER = U1IER | 0x01;               // 允许接收中断
        for(rt=0; rt<200; rt++);
    }
    rt = U1RBR;                             // 读取收到的数据

    return(rt);
}
```

7. 断开 MODEM 连接

断开 MODE 连接状态只要将 UART1 的 DTR 置为 0，MODEM 检测到这个信号后立即断开连接，具体代码见程序清单 3.21。

程序清单 3.21 ModemDialDown()代码

```
/*****************************************************************************
* 名称：ModemDialDown()
* 功能：Modem 断开。
* 入口参数：Number      拨号字符串
* 出口参数：返回当前 MODEM 状态值
*****************************************************************************/
uint8   ModemDialDown(void)
{
    U1MCR = 0x02;               // 设置 DTR 为 0，使 MODEM 挂机并返回命令状态
    DelayNS(1);                 // MODEM 中断会设置 ModemState 为 MODEM_CLOSE
    U1MCR = 0x03;               // 恢复 DTR 为 1
    DelayNS(1);
```

```
    return(ModemState);
}
```

第4章 ZLG/SD 软件包使用手册

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

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

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

在本章中，除了特别说明以外，“卡”都是指 SD 卡或 MMC 卡。

4.1 SD/MMC 卡的外部物理接口

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

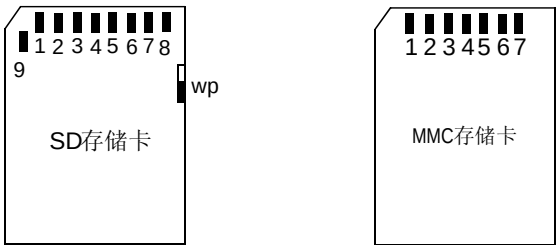


图 1 SD 卡和 MMC 卡的形状和接口（上视图）

表 4.1 为 SD/MMC 卡各触点的名称及作用，其中 MMC 卡只使用了 1～7 触点。

表 4.1 SD/MMC 卡的焊盘分配

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

注：1. S：电源；I：输入；O：推挽输出；PP：推挽 I/O。

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

这样定义是为了与 MMC 卡保持兼容。

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

由表 4.1 可见，SD 卡和 MMC 卡在不同的通信模式下，各引脚的功能也不相同。这里的通信模式是指微控制器（主机）访问卡时使用的通信协议，分为两种：SD 模式及 SPI 模式。

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

4.1.1 SD 模式

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

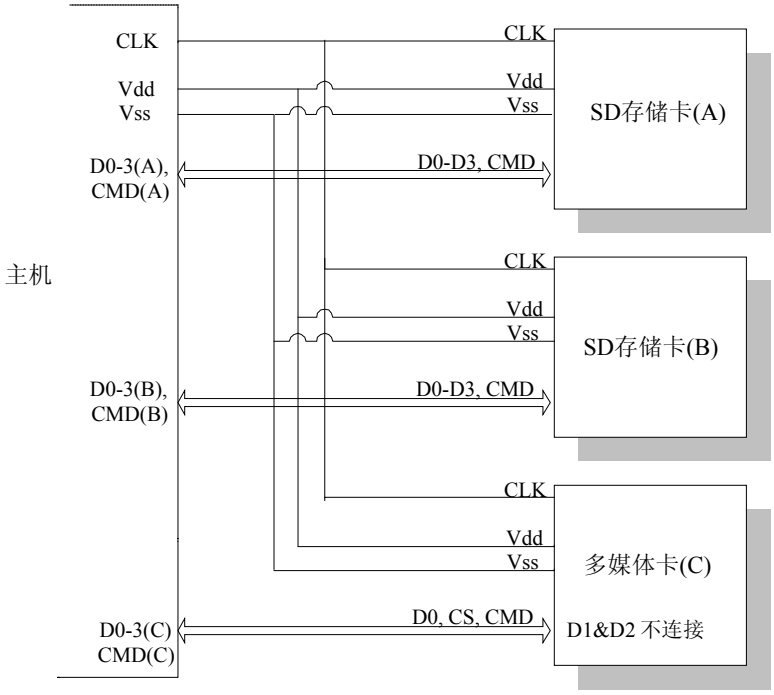


图 2 SD 存储卡系统（SD 模式）的总线拓扑结构

SD 总线上的信号线的详细功能描述如表 4.2 所示。

表 4.2 SD 总线信号线功能描述

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

续上表

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

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

4.1.2 SPI 模式

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

卡的 SPI 接口与大多数微控制器的 SPI 接口兼容。卡的 SPI 总线的信号线如表 4.3 所示。

表 4.3 SD 卡与 MMC 卡的 SPI 接口描述

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

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

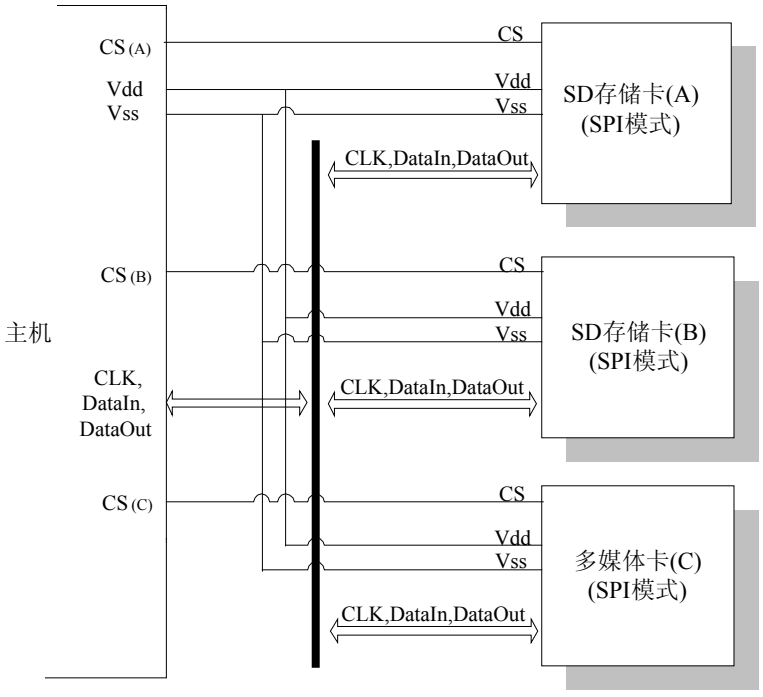


图 3 SD 存储卡系统（SPI 模式）的总线拓扑结构

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

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

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

4.2 访问 SD/MMC 卡的 SPI 模式硬件电路设计

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

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

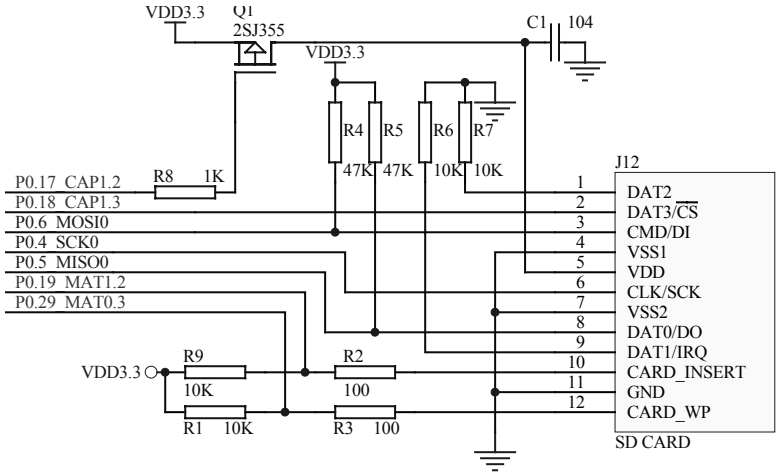


图 4 SD 卡卡座与 LPC2210 接口电路（SPI 模式）

图中，LPC2210 与 SD/MMC 卡卡座的连接引脚如表 4.4 所示。

表 4.4 LPC2210 与 SD/MMC 卡卡座的连接引脚

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

注意：对于不同的卡座，卡完全插入检测电平和写保护检测电平可能有所不同。

接口电路包括以下几个部分。

4.2.1 SPI 总线

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

4.2.2 卡供电控制

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

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

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

4.2.3 卡检测电路

卡检测电路包括两部分：卡是否完全插入到卡座中和卡是否写保护。

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

卡是否写保护的检测与卡是否完全插入到卡座中的检测原理是一样的。

4.3 ZLG/SD 软件包的文件结构及整体构架

本小节介绍 ZLG/SD 软件包的组成文件以及它们之间的联系。

4.3.1 ZLG/SD 软件包的文件组成

ZLG/SD 软件包包括的文件如表 4.5 所示。

表 4.5 ZLG/SD 软件包包含的文件

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

以上这些文件构成了本软件包，下面说明由这些文件构成的整体框架。

4.3.2 ZLG/SD 软件包整体框架

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

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

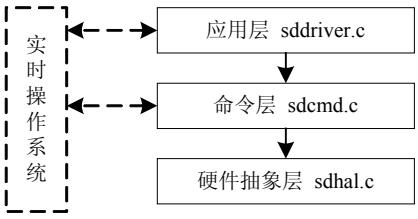


图 5 ZLG/SD 软件包结构图

各层的特点如下：

（1）硬件抽象层：读写 SD/MMC 卡的硬件条件配置，与硬件相关的函数；

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

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

4.4 ZLG/SD 软件包的使用说明

使用本软件包之前，必须配置好本软件包使用的硬件条件，如果你的硬件条件与 4.2 小节中的硬件条件一样，那么无须配置就可以立即使用。下面说明怎样配置本软件包的硬件条件，才能将其用于 LPC2210 系列微控制器。

4.4.1 ZLG/SD 软件包的硬件配置

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

1. 软件包参数配置

软件包的参数配置如程序清单 4.1 所示，配置选项如下：

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

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

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

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

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

程序清单 4.1 软件包参数配置

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

2． 功能配置

软件包中有一些功能不是所有用户都可能用到的，所以可以裁剪去不需要的函数，以减小其代码量。程序清单 4.2 的宏定义用于使能编译软件包中的某些比较少用的函数，当取值为 1 时，使能编译对应的函数；为 0 时，禁止编译对应的函数。这些宏定义起到裁剪软件包代码大小的目的。

程序清单 4.2 软件包函数使能

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

3． 硬件条件配置

这部分以宏的形式对卡的 SPI 口和 IO 口相关操作进行定义，尽可能地将硬件相关的部分放于这一部分。例如，配置 LPC2210 的 4 个 IO 口为 SPI 接口的宏定义如程序清单 4.3（1）所示。对于 SD 卡卡座供电引脚的控制如程序清单 4.3（2）所示。用户阅读 LPC2210 的数据手册就可以了解这些配置的含义。该文件还包括对其它 IO 口的配置，程序清单 4.3 都有详细的注释。

程序清单 4.3 LPC2210 的 IO 口配置宏定义

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

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

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

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

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

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

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

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

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

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

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

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

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

4. 设置 SPI 接口的时钟频率小于 400kHz

该函数主要是在 SD/MMC 卡初始化阶段，用于设置 SPI 接口的时钟频率小于 400kHz，因为 MMC 卡在初始化期间 SPI 总线的时钟频率不能高于 400kHz，这样本软件包才能达到兼容 MMC 卡的目的。该函数如程序清单 4.4 所示（见 sdhal.c 文件）。该函数只是修改 LPC2210 SPI 接口的 SPI 时钟计数寄存器 SPI_SPCCR 的分频值来达到目的。

程序清单 4.4 设置 SPI 接口的时钟频率小于 400kHz

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

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

得到的 SPI 接口 SCK 的频率为：

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

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

44.2368 / 8 = 5.5296MHz。

这个值正是 sdconfig.h 文件中宏定义 SPI_CLOCK 的值：

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

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

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

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

4.4.2 ZLG/SD 软件包提供的 API 函数

用户可以利用本软件包提供的 API 函数对 SD/MMC 卡进行访问，见表 4.6 至表 4.11。

表 4.6 SD_Initialize()

函数名称	SD_Initialize
函数原型	INT8U SD_Initialize(void)
功能描述	初始化 SD/MMC 卡、设置块大小为 512 字节，获取卡的相关信息
函数参数	无
函数返回值	SD_NO_ERR：初始化成功； > 0: 初始化失败（错误码，见表 4.12）
特殊说明和注意点	该函数设置了卡的读/写块长度为 512 字节

表 4.7 SD_ReadBlock ()

函数名称	SD_ReadBlock
函数原型	INT8U SD_ReadBlock(INT32U blockaddr, INT8U *recbuf)
功能描述	读 SD/MMC 卡的一个块
函数参数	blockaddr：以块为单位的块地址。例如，卡开始的 0 ~ 511 字节为块地址 0，512 ~ 1023 字节的块地址为 1 recbuf： 接收缓冲区，长度固定为 512 字节
函数返回值	SD_NO_ERR：读成功； > 0: 读失败（错误码，见表 4.12）
特殊说明和注意点	recbuf 的长度必须是 512 字节

表 4.8 SD_WriteBlock()

函数名称	SD_WriteBlock
函数原型	INT8U SD_WriteBlock(INT32U blockaddr, INT8U *sendbuf)
功能描述	写 SD/MMC 卡的一个块
函数参数	blockaddr：以块为单位的块地址。例如，卡开始的 0 ~ 511 字节为块地址 0，512 ~ 1023 字节的块地址为 1 sendbuf： 发送缓冲区，长度固定为 512 字节
函数返回值	SD_NO_ERR：写成功； > 0: 写失败（错误码，见表 4.12）
特殊说明和注意点	sendbuf 的长度必须是 512 字节

表 4.9 SD_ReadMultiBlock()

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

功能描述	读 SD/MMC 卡的多个块
函数参数	blockaddr： 以块为单位的块地址 blocknum： 块数 recbuf： 接收缓冲区，长度为 512 * blocknum 字节
函数返回值	SD_NO_ERR：读成功； ＞0: 读失败（错误码，见表 4.12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_ReadMultiBlock_EN 置为 1

表 4.10 SD_WriteMultiBlock ()

函数名称	SD_WriteMultiBlock
函数原型	INT8U SD_WriteMultiBlock(INT32U blockaddr, INT32U blocknum, INT8U *sendbuf)
功能描述	读 SD/MMC 卡的多个块
函数参数	blockaddr：以块为单位的块地址 blocknum：块数 sendbuf： 发送缓冲区，长度为 512 * blocknum 字节
函数返回值	SD_NO_ERR：写成功； ＞0: 写失败（错误码，见表 4.12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_WriteMultiBlock_EN 置为 1

表 4.11 SD_EraseBlock()

函数名称	SD_EraseBlock
函数原型	INT8U SD_EraseBlock(INT32U startaddr, INT32U blocknum)
功能描述	擦除 SD/MMC 卡的多个块
函数参数	startaddr： 以块为单位的块擦除起始地址 blocknum：块数（取值范围 1～sds.block_num）
函数返回值	SD_NO_ERR：擦除成功； ＞0: 擦除失败（错误码，见表 4.12）
特殊说明和注意点	使用时必须将 sdconfig.h 中的宏定义值 SD_EraseBlock_EN 置为 1 。Startaddr 和 blocknum 建议为 sds.erase_unit 的整数倍, 因为有的卡只能以 sds.erase_unit 为单位进行擦除

其它函数不常用，这里就不一一列出了。需要用到其它函数的读者可以阅读源码中的函数说明。表 4.6 至表 4.11 函数返回值所代表的含义如表 4.12 所示。

表 4.12 错误代码列表

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

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

第5章 ISP1161A1 和 ISP1181B 简介及硬件电路设计

USB 总线主要用于 USB 设备与主机之间的数据通信，特别为 USB 设备与主机之间大量数据的传输提供了高速、可靠的传输协议。例如：在嵌入式系统中，可以利用 USB 设备与微控制器构成 USB 设备。USB 设备与 PC 机 USB 主控器相连就可以实现嵌入式系统与 PC 机之间的通信了，也就可以实现诸如 U 盘、移动硬盘、USB 接口打印机等功能。

ISP1181B 是 Philips 公司生产的符合通用串行总线 2.0 规范（全速）的 USB 设备芯片，该芯片简称为 D13。

ISP1161A1 是一个带有两个 USB 主机接口和一个 USB 设备接口的 USB 芯片，芯片内部的 USB 设备控制器与 ISP1181B 结构完全一样（除了芯片 ID 号不同）。ISP1181B 又简称为 D13，因此，本文中提到的 D13 既可以指 ISP1181B，也可以指 ISP1161A 的 USB 设备控制器。

ZLG/USBD13 软件包就是用于控制 D13 的软件包，实现 D13 端点数据的收发，该软件包适用于以下芯片：

（1）ISP1161A1 芯片中的 USB 设备控制器；

（2）ISP1181A，ISP1181B。

下面先简单介绍 USB 芯片 ISP1161A1 和 ISP1181B。

5.1 ISP1161A1 简介及硬件电路设计

ISP1161A1 是一个单片通用串行总线（USB）主机控制器（HC）和设备控制器（DC）。ISP1161A1 的主机控制器部分符合通用串行总线 2.0 规范，支持全速（12Mbit/s）和低速（1.5Mbit/s）的数据传输。ISP1161A1 的设备控制器部分也符合通用串行总线 2.0 规范，支持全速（12Mbit/s）的数据传输。这两个 USB 控制器共用一个微处理器总线接口，有相同的数据总线，但 I/O 地址不同。它们也有各自的中断请求输出管脚和独立的 DMA 通道，DMA 通道含有各自的 DMA 请求输出管脚和 DMA 应答输入管脚。这就使微处理在应用中可以同时对 USB HC 和 USB DC 进行控制。

ISP1161A1 引脚分配图，如图 5.1 所示。

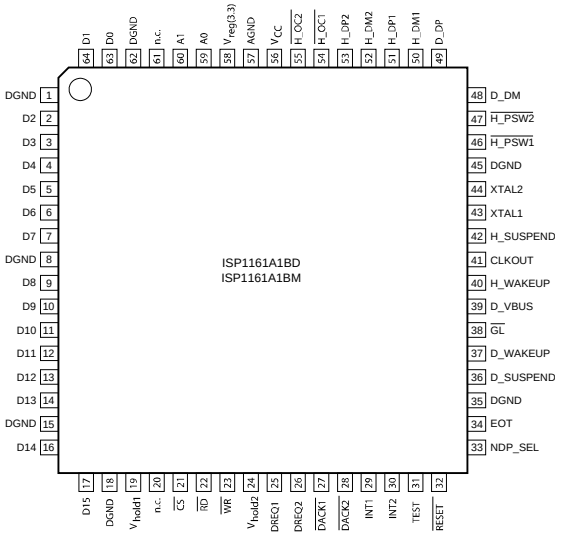


图 5.1 引脚分配图

5.1.1 ISP1161A1 管脚描述

ISP1161A1 管脚描述如表 5.1 所示。

表 5.1 ISP1161A1 芯片 LQFP64 管脚描述

符号[1]	管脚	类型	描 述
DGND	1	-	数字地
D2	2	I/O	双向数据的位 2；转换速率控制；TTL 电平输入；三态输出
D3	3	I/O	双向数据的位 3；转换速率控制；TTL 电平输入；三态输出
D4	4	I/O	双向数据的位 4；转换速率控制；TTL 电平输入；三态输出
D5	5	I/O	双向数据的位 5；转换速率控制；TTL 电平输入；三态输出
D6	6	I/O	双向数据的位 6；转换速率控制；TTL 电平输入；三态输出
D7	7	I/O	双向数据的位 7；转换速率控制；TTL 电平输入；三态输出
DGND	8	-	数字地
D8	9	I/O	双向数据的位 8；转换速率控制；TTL 电平输入；三态输出
D9	10	I/O	双向数据的位 9；转换速率控制；TTL 电平输入；三态输出
D10	11	I/O	双向数据的位 10；转换速率控制；TTL 电平输入；三态输出
D11	12	I/O	双向数据的位 11；转换速率控制；TTL 电平输入；三态输出
D12	13	I/O	双向数据的位 12；转换速率控制；TTL 电平输入；三态输出
D13	14	I/O	双向数据的位 13；转换速率控制；TTL 电平输入；三态输出
DGND	15	-	数字地
D14	16	I/O	双向数据的位 14；转换速率控制；TTL 电平输入；三态输出
D15	17	I/O	双向数据的位 15；转换速率控制；TTL 电平输入；三态输出
DGND	18	-	数字地
V_{hold1}	19	-	电压保持管脚 1；内部连接到 $V_{reg(3.3)}$和 V_{hold2} 管脚。当 V_{CC} 连接到 5V 时，该管脚将输出 3.3V，因此需避免与 5V 电压连接。当 Vcc 连接到 3.3V 时，该管脚既可与 3.3V 相连，也可悬空。在所有情况下，需将该管脚去耦接地。
n.c.	20	-	不连接
$\overline{CS}$	21	I	片选输入
$\overline{RD}$	22	I	读选通输入
$\overline{WR}$	23	I	写选通输入
V_{hold2}	24	-	电压保持管脚 2；内部连接到 $V_{reg(3.3)}$和 V_{hold1} 管脚。当 V_{CC} 连接到 5V 时，该管脚将输出 3.3V，因此需避免与 5V 电压连接。当 Vcc 连接到 3.3V 时，该管脚既可与 3.3V 相连，也可悬空。在所有情况下，需将该管脚去耦接地。
DREQ1	25	O	HC DMA 请求输出（极性可编程）；当 DMA 控制器上获得信号时，ISP1161A1 启动 DMA 传输；见 10.4.1 部分
DREQ2	26	O	DC DMA 请求输出（极性可编程）；当 DMA 控制器上获得信号时，ISP1161A1 启动 DMA 传输；见 13.1.4 部分
$\overline{DACK1}$	27	I	HC DMA 应答输入。当不使用时，该管脚必须外接一个 10kΩ 电阻至 Vcc
$\overline{DACK2}$	28	I	DC DMA 应答输入。当不使用时，该管脚必须外接一个 10kΩ 电阻至 Vcc

续上表

INT1	29	O	HC 中断输出；电平、触发沿和极性可编程；见 10.4.1 部分
INT2	30	O	DC 中断输出；电平、触发沿和极性可编程；见 13.1.4 部分
TEST	31	O	测试输出；仅用于测试；在正常操作的过程中该管脚不连接
$\overline{\text{RESET}}$	32	I	复位输入（施密特触发）；一个低电平产生一次异步复位(内部上拉电阻)
NDP_SEL	33	I	指示 HC 的当前下行端口号： 0－选择 1 号下行端口 1－选择 2 号下行端口 仅改变 HcRhDescriptorA 寄存器的 NDP 阶段值；2 个端口将一直使能；见 10.3.1 部分（内部上拉电阻）
EOT	34	I	DMA 主机设备通知 ISP1161A1 DMA 传输结束；有效电平可编程；见 10.4.1 部分
DGND	35	-	数字地
D_SUSPEND	36	O	DC 的‘挂起’状态指示器输出；高电平有效
D_WAKEUP	37	I	DC 的唤醒输入；从‘挂起’状态产生一个远程唤醒信号（高电平有效）；当不使用时，该管脚必须通过一个外部 10KΩ的电阻连接到 DGND（内部下拉电阻）
$\overline{\text{GL}}$	38	O	GoodLink LED 指示器输出（开漏，8mA）；LED 默认为 ON，在 USB 通信时闪烁关闭；用一个 470Ω（Vcc＝5.0V）或 330Ω（Vcc＝3.3V）的串联电阻连接一个 LED
D_VBUS	39	I	DC 的 USB 上行端口 V_{BUS} 检测输入；当不使用时，该管脚必须通过一个 1MΩ的电阻连接到 DGND
H_WAKEUP	40	I	HC 的唤醒输入；从‘挂起’状态产生一个远程唤醒信号（高电平有效）；当不使用时，该管脚必须通过一个外部 10KΩ的电阻连接到 DGND（内部下拉电阻）
CLKOUT	41	O	可编程的时钟输出（3 到 48MHz）；默认为 12MHz
H_SUSPEND	42	O	HC 的‘挂起’状态指示器输出；高电平有效
XTAL1	43	I	晶振输入；直接与一个 6MHz 的晶振相连；当 XTAL1 被连接到一个外部时钟源时，管脚 XTAL2 必须悬空
XTAL2	44	O	晶振输出；直接与一个 6MHz 的晶振相连；当管脚 XTAL1 被连接到一个外部时钟源时，XTAL2 必须悬空
DGND	45	-	数字地
$\overline{\text{H_PSW1}}$	46	O	下行端口 1 的电源转换控制输出；开漏输出
$\overline{\text{H_PSW2}}$	47	O	下行端口 2 的电源转换控制输出；开漏输出
D_DM	48	AI/O	DC 上行端口的 USB D－数据线；当不使用时，该管脚必须悬空
D_DP	49	AI/O	DC 上行端口的 USB D+数据线；当不使用时，该管脚必须悬空
H_DM1	50	AI/O	HC 下行端口 1 的 USB D－数据线
H_DP1	51	AI/O	HC 下行端口 1 的 USB D+数据线
H_DM2	52	AI/O	HC 下行端口 2 的 USB D－数据线；当不使用时，该管脚必须悬空
H_DP2	53	AI/O	HC 下行端口 2 的 USB D+数据线；当不使用时，该管脚必须悬空
$\overline{\text{H_OC1}}$	54	I	HC 下行端口 1 的过流检测输入
$\overline{\text{H_OC2}}$	55	I	HC 下行端口 2 的过流检测输入

续上表

Vcc	56	-	电源电压输入（3.0 到 3.6V 或 4.75 到 5.25V）。该管脚与内部 3.3V 调整器的输入端相连。当它连接到 5V 时，内部调整器将输出 3.3V 到管脚 Vreg(3.3)，V_{hold1} 和 V_{hold2}。当它与 3.3V 相连时，不再连接内部调整器
AGND	57	-	模拟地
Vreg(3.3)	58	-	内部 3.3V 调整器输出；当管脚 Vcc 与 5V 相连时，该管脚输出 3.3V。当管脚 Vcc 与 3.3V 相连时，该管脚连接到 3.3V
A0	59	I	地址输入；用于选择命令（A0=1）或数据（A0=0）
A1	60	I	地址输入；选择将 AutoMux 切换到 DC（A1=1）或 AutoMux 切换到 HC（A1=0）；见表 3
n.c.	61	-	不连接
DGND	62	-	数字地
D0	63	I/O	双向数据的位 0；转换速率控制；TTL 电平输入；三态输出
D1	64	I/O	双向数据的位 1；转换速率控制；TTL 电平输入；三态输出

[1] 管脚符号上的横线（如：$\overline{\text{NAME}}$ ）表示此信号低电平有效。

ISP1161A1 定义了一个通用 PIO 接口，以增加速度及易用性。并可与大多数微控器直接接口。对于一个微控制器，ISP1161A1 类似于一个具有 16 位数据总线的存储器设备，只使用 A1 和 A0 两条地址线访问内部控制寄存器及 FIFO 缓冲区 RAM。因此，ISP1161A1 仅占用四个 I/O 端口，或一个微处理器中的四个存储单元。在可编程的 I/O（PIO）操作模式下，外部微处理器可读/写 ISP1161A1 的内部控制寄存器及 FIFO 缓冲区 RAM。

5.1.2 ISP1161A1 与 LPC2200 的硬件电路设计

在 SmartARM2200 开发板上，ISP1161A1 与 LPC2200 硬件连接原理图如图 5.2 所示。

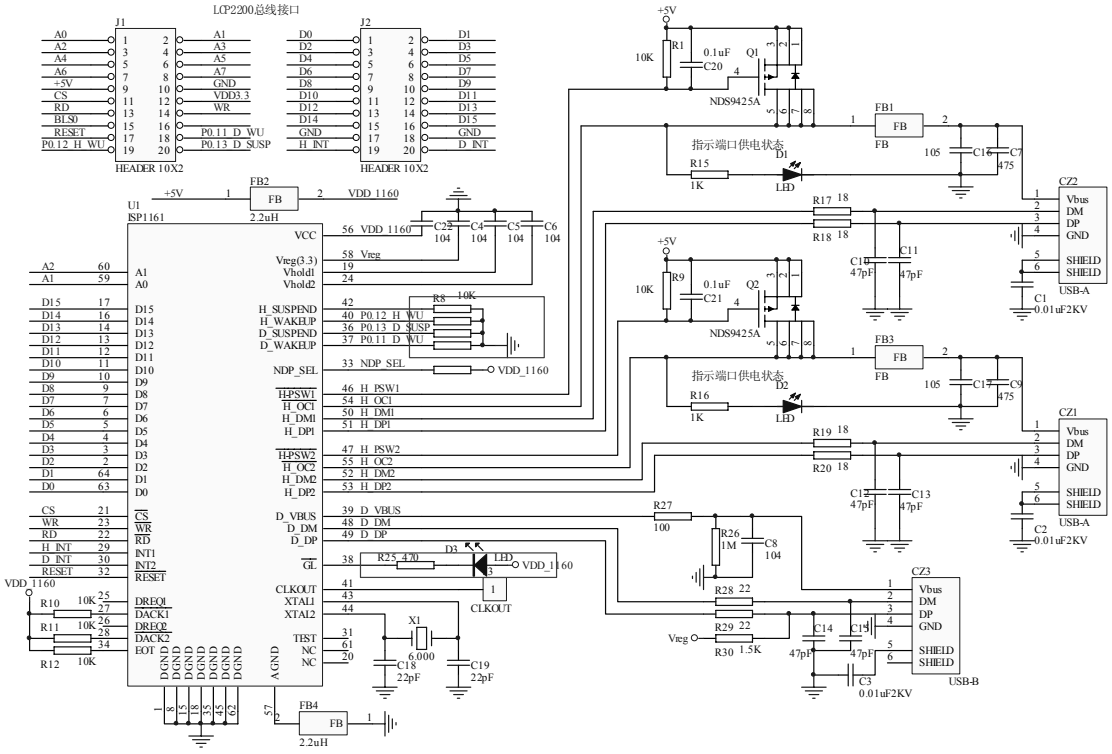


图 5.2 ISP1161A1 硬件连接图

图 5.2 中，J1、J2 为 LPC2200 微处理器的数据总线、地址总线、中断和 GPIO 引脚（J1 和 J2 这两排引脚为 ISP1161BM PACK 的插入座）。ISP1161A1 与 LPC2200 微处理器引脚连接描述，如表 5.2 所示。

表 5.2 ISP1161A1 与 LPC2200 连接描述

ISP1161A1 引脚	功能描述	LPC2200 引脚
D0~D15	16 位数据总线	D0~D15
A0~A1	ISP1161A1 只使用两个地址线	A1~A2
INT1	HC 中断	P0.16
INT2	DC 中断	P0.15
/CS	ISP1161A1 片选	nCS2
/WR	写使能	nWE
/RD	读使能	nOE
/RESET	控制 ISP1161A1 硬复位	P0.10
H_WAKEUP	控制 ISP1161A1 HC 唤醒	P0.12
H_SUSPEND	获取 ISP1161A1 HC 挂起状态	不连接
D_WAKEUP	控制 ISP1161A1 DC 唤醒	P0.11
D_SUSPEND	获取 ISP1161A1 DC 挂起状态	P0.13

ISP1161A1 使用 LPC2200 外部存储控制的 Bank2 部分，所以 Bank2 的存储空间范围为 0x82000000~0x82FFFFFF。由于 ISP1161A1 只有 2 根地址线 A0 ~ A1 分别使用了 LPC2200 微处理器 A1~A2（Bank2 配置为 16 位模式，A0 无效），所以 ISP1161A1 的可操作地址为：

HC 命令地址：0x82000002；

HC 数据地址：0x82000000；

DC （D13）命令地址：0x82000006；

DC （D13）数据地址：0x82000004。

5.2 ISP1181B 简介及硬件电路设计

ISP1181B 是一款性价比很高的 USB1.1 器件，并完全符合 USB2.0 规范。具有以下特性：

- 符合通用串行总线 2.0 规范（全速）和大多数器件的分类规范。
- 高性能 USB 接口器件，集成了串行接口引擎（SIE）、FIFO 存储器、收发器和 3.3V 的电压调节器。
- 高速（11.1MB/秒或 90ns 的读/写周期）并行接口。
- 完全自治和多配置的 DMA 操作。
- 多达 14 个可编程 USB 端点，包含 2 个固定的控制 IN/OUT 端点。
- 集成 2462 字节的可配置 FIFO 存储器。
- 端点的双缓冲配置增加了数据吞吐量并轻松实现实时数据传输。
- 与大多数微控制器/微处理器实现无缝连接。
- 总线供电可降低功耗和‘挂起’电流。
- 集成 PLL 的 6MHz 晶体振荡器，降低了 EMI。
- 在‘挂起’时可控制 LazyClock（100±50％）输出。
- 可通过软件控制与 USB 总线的连接（SoftConnect^{TM}）。
- 良好的 USB 连接指示器，通信时闪烁（GoodLink^{TM}）。

- 可编程频率的时钟输出（高达 48MHz）。
- 符合 ACPI™、OnNow™ 和 USB 电源管理的要求。
- 内置有上电和低电压复位电路，也可通过软件复位。
- 可工作在扩展的 USB 总线电压范围（4.0～5.5V）内，I/O 口可承受 5V 的电压。
- 工作温度范围：－40 ～ + 85℃。
- 具有高错误恢复率的全扫描设计。
- 封装：TSSOP48 和 HVQFN48。

5.2.1 ISP1181B 管脚描述

ISP1181B 管脚排列如图 5.3 所示。

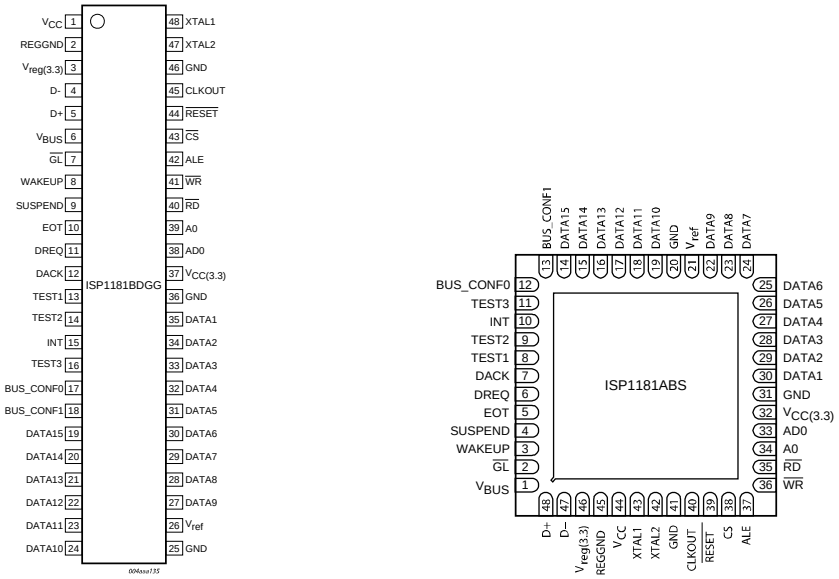


图 5.3 ISP1181B 管脚排列图

表 5.3 为 ISP1181B 的管脚描述。

表 5.3 ISP1181B 管脚描述

符号[1]	管脚		类型	描述
	TSSOP48	HVQFN48		
Vcc	1	44	-	供电电源 (3.3 或 5.0V)。
REGGND	2	45	-	电压调节器地供电。
Vreg(3.3)	3	46	-	内部调节器调节后的电压(3.3V±10％)；用来连接 D+线上的去耦电容和上拉电阻； 注：不能用作外部器件的供电电源。
D-	4	47	AI/O	USB D-连接（模拟）。
D+	5	48	AI/O	USB D+连接（模拟）。
V_{BUS}	6	1	I	V_{BUS} 检测输入。
$\overline{GL}$	7	2	O	GoodLink LED 指示器输出（开漏，8mA）；默认输出高电平，USB 通信时闪烁关断；通过串联一个 470Ω（Vcc＝5.0V）或 330Ω（Vcc＝3.3V）的电阻与 LED 相连。

WAKEUP	8	3	I	唤醒输入（边沿触发，低到高）；在‘挂起’状态下产生一个远程唤醒信号。
SUSPEND	9	4	O	‘挂起’状态指示器输出（4mA）；用作掉电应用时的电源开关控制输出（低有效）。
EOT	10	5	I	传输终止输入(极性可编程，见表 22)；DMA 控制器使用该信号来强制终止 ISP1181B 的 DMA 传输。
DREQ	11	6	O	DMA 请求输出(4mA;极性可编程，见表 22)；告知 DMA 控制器 ISP1181B 请求启动一次 DMA 传输。
DACK	12	7	I	DMA 应答输入（极性可编程，见表 22)；DMA 控制器使用该信号来回应 ISP1181B 启动 DMA 传输的请求(即通知 ISP1181B 可以开始 DMA 传输了)。
TEST1	13	8	I	测试输入；该管脚必须通过一个外部 10kΩ的电阻连接到 Vcc。
TEST2	14	9	I	测试输入；该管脚必须通过一个外部 10kΩ的电阻连接到 Vcc。
INT	15	10	O	中断输出；极性（高或低有效）和信号方式（电平或脉冲）可编程。
TEST3	16	11	O	测试输出；该管脚仅用于测试。
BUS_CONF 0	17	12	I	总线配置选择器。
BUS_CONF 1	18	13	I	总线配置选择器。
DATA15	19	14	I/O	D[15：0]的位 15；双向数据线（输出的斜率可控制，4mA）。
DATA14	20	15	I/O	D[15：0]的位 14；双向数据线（输出的斜率可控制，4mA）。
DATA13	21	16	I/O	D[15：0]的位 13；双向数据线（输出的斜率可控制，4mA）。
DATA12	22	17	I/O	D[15：0]的位 12；双向数据线（输出的斜率可控制，4mA）。
DATA11	23	18	I/O	D[15：0]的位 11；双向数据线（输出的斜率可控制，4mA）。
DATA10	24	19	I/O	D[15：0]的位 10；双向数据线（输出的斜率可控制，4mA）。
GND	25	20	-	电源地
V_{ref}	26	21	-	I/O 管脚参考电压（3.3V)；Vcc＝5.0V 时该管脚不连。
DATA9	27	22	I/O	D[15：0]的位 9；双向数据线（输出的斜率可控制，4mA）。
DATA8	28	23	I/O	D[15：0]的位 8；双向数据线（输出的斜率可控制，4mA）。
DATA7	29	24	I/O	D[15：0]的位 7；双向数据线（输出的斜率可控制，4mA）。

DATA6	30	25	I/O	D[15：0]的位 6；双向数据线（输出的斜率可控制，4mA）。
DATA5	31	26	I/O	D[15：0]的位 5；双向数据线（输出的斜率可控制，4mA）。
DATA4	32	27	I/O	D[15：0]的位 4；双向数据线（输出的斜率可控制，4mA）。
DATA3	33	28	I/O	D[15：0]的位 3；双向数据线（输出的斜率可控制，4mA）
DATA2	34	29	I/O	D[15：0]的位 2；双向数据线（输出的斜率可控制，4mA）。
DATA1	35	30	I/O	D[15：0]的位 1；双向数据线（输出的斜率可控制，4mA）。
GND	36	31	-	电源地
$V_{cc(3.3)}$	37	32	-	供电电源（3.0～3.6V）；Vcc＝5.0V 时该管脚不连。
AD0	38	33	I/O	复用双向地址和数据线；和输入 ALE 一起决定该管脚为地址 A0 或 D[15：0]的位 0；输入时低电平有效，输出的斜率可控制（4mA）。 **地址阶段**：输入 ALE 上高到低的跳变将该管脚的电平锁存，用作地址 A0（1＝命令，0＝数据）。 **数据阶段**：读操作时该管脚输出位 D[0]的值；写操作时该管脚锁存位 D[0]的电平。
A0	39	34	I	地址输入；选择命令（A0=1）或数据（A0=0）；当总线配置成地址/数据复用时，该管脚不用且必须固定为低（与 GND 相连）。
$\overline{RD}$	40	35	I	读选通输入。
$\overline{WR}$	41	36	I	写选通输入。
ALE	42	37	I	地址锁存使能输入；当总线配置成地址/数据复用时，该管脚高到低的跳变将 AD0 的电平锁存作为地址信息；当总线配置成地址/数据分开使用时该管脚必须固定为低（与 GND 相连）。
$\overline{CS}$	43	38	I	片选输入。
$\overline{RESET}$	44	39	I	复位输入（施密特触发器）；低电平产生异步复位；上电复位时连接到 Vcc（内部 POR 电路）。
CLKOUT	45	40	O	可编程的时钟输出（2mA）。
GND	46	41	-	电源地
XTAL2	47	42	O	晶振输出（6MHz）；连接一个基本的并联谐振晶体；当使用 XTAL1 脚上的外部时钟源时该管脚悬空。
XTAL1	48	43	I	晶振输入（6MHz）；连接一个基本的并联谐振晶体或外部时钟源（XTAL2 悬空）。

注： [1] 表中符号上的横线（如 $\overline{NAME}$ ）表示所代表的信号低电平有效。

ISP1181B 的详细介绍请到 http://www.zlgmcu.com 网站下载 ISP1181B 使用手册。

5.2.2 ISP1181B 与 LPC2200 的硬件电路设计

根据 ISP1181B 的引脚功能，就可以设计一个 USB 设备了，首先必须明白硬件设计的目标：

- 在 LPC2200 上实现 ISP1181B 的硬件连接，实现用 LPC2200 控制 ISP1181B；
- LPC2200 与 ISP1181B 数据总线的连接为 8 位总线或 16 位总线；
- LPC2220 的 GPIO 来唤醒进入挂起状态的 ISP1181B；
- 用 LED 的亮与灭来表示 ISP1181B 引脚 SUSPEND 进入挂起状态时电平的变化。

根据以上设计目标指导设计，得到 ISP1181B 连接到 LPC2200 的硬件原理图如图 5.4 所示，由该图可见 ISP1181B 各引脚与 LPC2200 的连接关系，如表 5.4 所示。

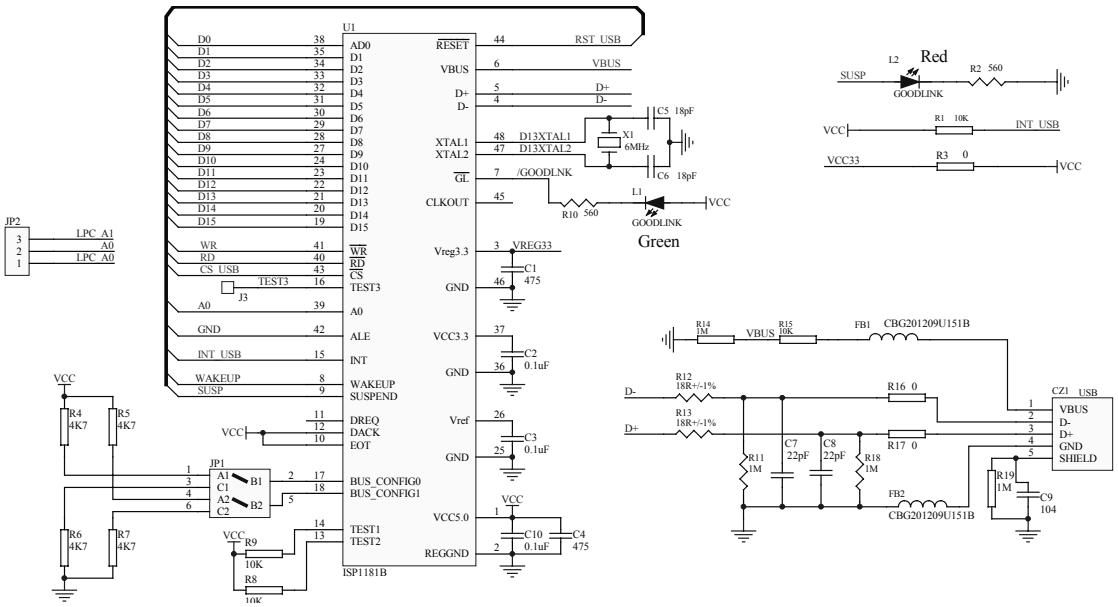


图 5.4 ISP1181B 与 LPC2200 硬件连接原理图

表 5.4 ISP1181B 与 LPC2200 连接关系

ISP1181B	功 能	LPC2200
D0～D15	ISP1181B 数据总线	D0～D15
A0	ISP1181B 地址总线	A0 或 A1（由 JP2 跳线选择）
$\overline{CS}$	ISP1181B 片选线	nCS2
$\overline{RD}$	ISP1181B 读使能（低电平有效）	nOE
$\overline{WR}$	ISP1181B 写使能（低电平有效）	nEW
INT	ISP1181B 中断输出信号	P0.16_EINT0
$\overline{RESET}$	ISP1181B 复位输入信号（低电平复位）	P0.10_RTS1
SUSPEND	ISP1181B 挂起输出信号	P0.13_DTR1
WAKEUP	ISP1181B 唤醒信号	P0.11

如图 5.4 所示，L1 为 GoodLink 指示灯。L2 为挂起指示灯，由于 ISP1181B 挂起时，SUSPEND 输出为高电平，此时 L2 亮（红色）；正常情况下，SUSPEND 为低平，此时 L2 灭。

D0 ～ D15 与 LPC2200 的 32 位总线中的 16 根线相连，构成了 16 位总线连接。由于 LPC2200 的数据总线与地址总线是分开的，又由于访问 ISP1181B 的数据总线可以为 8 位总

线或 16 位总线，因此 LPC2200 的数据线 A0 或 A1 由跳线 JP2 决定哪一引脚连接到 ISP1181B 的 A0 引脚。

当 ISP1181B 的 A0 引脚与 LPC2200 的 A0 相连时，LPC2200 用 8 位数据总线访问 ISP1181B，当 ISP1181B 的 A0 引脚与 LPC2200 的 A1 相连时，LPC2200 用 16 位数据总线访问 ISP1181B。

JP1 的跳线选择决定了 ISP1181B 接受访问的数据总线的长度，必须与 JP2 配合使用。JP1 和 JP2 的跳线含义如表 5.5 所示。

表 5.5 JP1 与 JP2 跳线

JP1 跳线	JP2	含义
B1 与 C1 短接，B2 与 C2 短接	3 与 2 短接	使能 ISP1181B 为 16 位数据总线
B1 与 C1 短接，A2 与 B2 短接	1 与 1 短接	使能 ISP1181B 为 8 位数据总线
其它方式		不允许

ISP1181B 的复位信号、唤醒信号、挂起信号线都连接到 LPC2200 的 GPIO。

USB 总线的 V_{BUS} 连接到 ISP1181B 的 V_{BUS} 引脚是为了让 ISP1181B 检测到 USB 总线的电源是否存在。

D+与 D-线串了 18 欧电阻是为了使 ISP1181B 的输出阻抗达到全速传输的要求。

LPC2200 使用 nCS2 片选 ISP1181B，对应于 LPC2200 外部存储控制的 Bank2 部分，其地址如下：

数据地址 0x82000000（8 位数据总线模式）
命令地址 0x82000001（8 位数据总线模式）
数据地址 0x82000000（16 位数据总线模式）
命令地址 0x82000002（16 位数据总线模式）

5.2.3 ISP1181B PACK 简介

ISP1181B PACK 是广州周立功单片机发展有限公司生产的 ISP1181B 开发接口板。该开发板直接插入到 EasyARM2200 开发板上，就可以进行 ISP1181B 的固件开发。该开发接口板原理图见图 5.4。

ISP1181B PACK 实物图如图 5.5 所示。

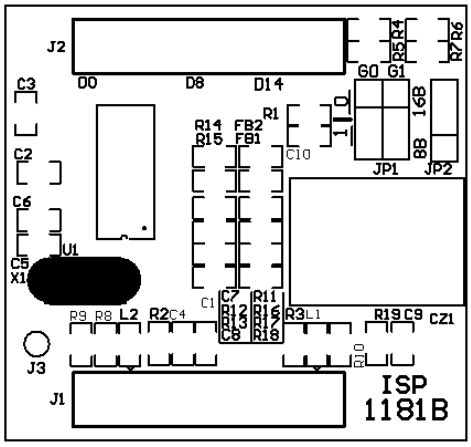


图 5.5 ISP1181B PACK 开发接口板

其中 JP1 与 JP2 跳线用于选择 LPC2200 访问 ISP1181B 的数据总线宽度。跳线含义见表 5.6。

表 5.6 ISP1181B PACK 跳线

JP1 跳线	JP2	含义
G0 跳至 0，G1 跳至 0	跳至 16B	使能 ISP1181B 为 16 位数据总线
G0 跳至 0，G1 跳至 1	跳至 8B	使能 ISP1181B 为 8 位数据总线
其它方式		不允许

第6章 ZLG/USBD13 软件包

ZLG/USBD13 软件包是周立功公司嵌入式系列软件包中的重要成员，该软件包用于控制 D13，它提供的 API 函数可以使用户很方便很容易地实现 D13 端点数据的收发，该软件包运行于嵌入式实时操作系统 μC/OS-II 上。下面介绍本软件包的结构以及它提供的 API 函数。

6.1 ZLG/USBD13 软件包结构

为了使软件可移植性强、易维护，该软件包采用分层的方法编写。综合考虑 USB 协议、D13 硬件接线、μC/OS-II 的结构来进行分层，如表 6.1 所示为 ZLG/USBD13 软件包分层结构表。大部分文件名以‘D13’作为开头。

表 6.1 ZLG/USBD13 软件包分层结构表

文件名	简要说明	相关性
D13HAL.c	D13 与 MCU 之间的硬件接口	与硬件相关
D13CI.c	ISP1181B 命令接口	与硬件无关
Chap_9.c	USB 协议层	与 μC/OS-II 相关
Descriptor.c	USB 协议层(ISP1181B 描述符)	与 μC/OS-II 相关
D13Driver.c	USB 应用层	与 μC/OS-II 相关

以上各层关系可用图 6.1 来表示。

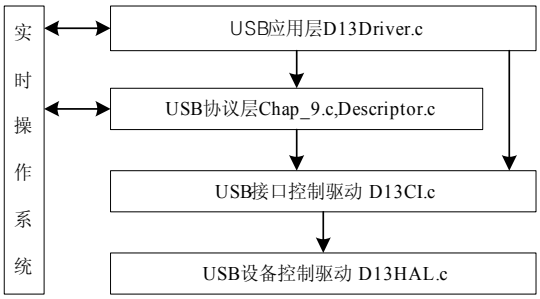


图 6.1 ZLG/USBD13 软件包分层结构图

图 6.1 中，双向线表示实时操作系统 μC/OS-II 与 ZLG/USBD13 软件包之间的数据交换。单向线表示上层软件对下层软件的调用。这样软件包的结构分明。

6.2 ZLG/USBD13 软件包的使用方法

ZLG/USBD13 软件包包括的文件如表 6.2 所示，目前本软件包的版本号为 1.0。

表 6.2 ZLG/D13 软件包软件包主要文件列表

文件名	描述及作用
D13Config.h	ZLG/D13 软件包的配置，包括所有与硬件相关的配置
D13HAL.c	USB 设备控制驱动，与具体使用的 MCU 及硬件相关
D13HAL.h	USB 设备控制驱动 D13HAL.c 的头文件

续上表

D13CI.c	USB 接口控制驱动，与具体使用的 USB 芯片相关
D13CI.h	USB 接口控制驱动 D13CI.c 的头文件
Descriptor.c	USB 协议层，ISP1181B 所有描述符取值
Descriptor.h	USB 描述符结构体的定义，以及 ISP1181B 用到的一些宏定义的取值
Chap_9.c	USB 协议层，完成所有与控制传输相关的工作
Chap_9.h	USB 协议层 Chap_9.c 的头文件，包含控制传输用到的结构体的定义
D13Driver.c	USB 应用层，包含提供给用户使用的 API 函数
D13Driver.h	USB 应用层 D13Driver.c 的头文件

使用该软件包时，建议使用工程模板：ARM Executable Image for UCOSII(for lpc2200)。

6.2.1 ZLG/USBD13 软件包的硬件配置方法

ZLG/USBD13 软件包的硬件配置的定义在 D13Config.h 文件中，下面进行详细介绍。

为了方便用户修改 D13 与 LPC2200 之间的配置，使用了一个用于配置硬件的头文件 D13Config.h。用户可以根据自己硬件电路设计的不同，修改该头文件自定义 D13 与 LPC2200 之间的硬件配置。

下面以图 5.2 和图 5.4 为例子，介绍如何配置这个头文件。

1. 定义访问 D13 的数据总线的长度

由于 ISP1181B 可用 8 位数据总线访问，又可以用 16 位总线访问，为了方便移植，在程序清单 6.1 中定义了 16 位总线是否使能宏定义 D13_16BIT_EN。当该宏定义值为 1 时，该软件包使用 16 位总线访问 ISP1181B；当值为 0 时，使用 8 位总线进行访问。

程序清单 6.1 ISP1181B 的数据总线长度

```
#define D13_16BIT_EN        1                          //是否使用 16 位数据总线访问
```

由于 ISP1161A1 的数据总线固定为 16 位，所以使用 ISP1161A 的 USB 设备控制器时，该宏定义的值应为 1。

2. 定义 D13 的数据地址与命令地址

由于 ISP1181B 的引脚 $\overline{CS}$ 、A0 分别与 LPC2200 的 nCS2、A0 或 A1 相互连接，故 ISP1181B 地址范围处于 0x82000000~ 0x82FFFFFF 之间。ISP1181B 占用其中的两个地址，请见程序清单 6.2。这时，宏定义 D13_16BIT_EN 就起作用了。

程序清单 6.2 定义访问 ISP1181B 的命令与数据地址

```
#if D13_16BIT_EN
#define D13_COM         (*((volatile INT16U *) 0x82000002))    //16 位总线方式下的命令地址
#define D13_DAT         (*((volatile INT16U *) 0x82000000))    //16 位总线方式下的数据地址
#else
#define D13_COM         (*((volatile INT8U *) 0x82000001))     //8 位总线方式下的命令地址
#define D13_DAT         (*((volatile INT8U *) 0x82000000))     //8 位总线方式下的数据地址
```

```
#endif
```

而 ISP1161A1 的 USB 设备控制器的命令地址和数据地址分别为 0x82000006 和 0x82000004。因此，ISP1161A1 的 USB 设备控制器的命令地址和数据地址的配置如程序清单 6.3 所示。

程序清单 6.3 定义访问 ISP1161A1 的命令与数据地址

```
#if D13_16BIT_EN
#define D13_COM          (*((volatile INT16U *) 0x82000006))    //16 位总线方式下的命令地址
#define D13_DAT          (*((volatile INT16U *) 0x82000004))    //16 位总线方式下的数据地址
#else
#define D13_COM          (*((volatile INT8U *) 0x82000001))     //8 位总线方式下的命令地址
#define D13_DAT          (*((volatile INT8U *) 0x82000000))     //8 位总线方式下的数据地址
#endif
```

3. 定义 ISP1181B 和 ISP1161A1 控制引脚 $\overline{\text{RESET}}$ 、SUSPEND 和 WAKEUP

定义 $\overline{\text{RESET}}$ 、SUSPEND 和 WAKEUP 这 3 个功能端口连接到 LPC2200 P0 口的第几个引脚，见程序清单 6.4。

程序清单 6.4 定义 D13 的功能端口连接到 LPC2200 P0 口的第几个引脚

```
#define USBRST_NUM    10          //RESET     引脚连接到 LPC2200 P0 口引脚的引脚数
#define SUSP_NUM      13          //SUSPEND 引脚连接到 LPC2200 P0 口引脚的引脚数
#define WKUP_NUM      11          //WAKEUP   引脚连接到 LPC2200 P0 口引脚的引脚数
```

4. 定义 D13 中断引脚

将 D13 连接到 LPC2200 的 INT 引脚定义为中断输入口，并定义中断处理结束后，清除中断标志以及通知 LPC2200 中断结束的宏定义。请见程序清单 6.5。

程序清单 6.5 配置 D13 的中断引脚

```
#define INIT_INTD13()   PINSEL1 &= 0xFFFFFFFC;PINSEL1 |= 0x01     //初始化 INT 为中断输入口
#define CLR_INTD13()    EXTINT = 0x01                              //清除 EINT0 中断标志
#define CLR_INT()       VICVectAddr = 0x00                         //通知中断结束
```

与软件包硬件配置相关的内容就介绍到此，但与软件包运行相关，但不属于本软件包的相关配置还有两点需要注意。

5. D13 数据总线配置

由于 D13 的数据总线使用 LPC2200 外部存储控制的 Bank2 部分，所以需对 Bank2 的存储器配置寄存器 BCFG2 进行配置。如表 6.3 所示为 BCFG2 各个配置位的含义以及与 ISP1181B 相关的配置值。

表 6.3 BCFG2 配置值

BCFG2	名称	配置值	配置值含义
3:0	IDCY	0000	空闲 CCLK 周期数
4	保留	0	
9:5	WST1	00000	读访问长度
10	RBLE	1	
15:11	WST2	00000	写访问长度
16:23	保留	0	
24	BUSERR	0	
25	WPERR	0	
26	WP	0	非写保护
27	BM	0	非 Burst ROM
29:28	MW	00	使用 8 位总线
		01	使用 16 位总线
31:30	AT	00	通常写入 0

当前只有 D13 占用了 Bank2，所以可以用以下语句来配置寄存器 BCFG2，如程序清单 6.6 所示。首先将 BCFG2 寄存器清 0，然后根据表 6.3 置位相关位。由于考虑到用户可能还需要在 Bank2 挂接其它设备，需要修改该寄存器的值。所以不将该配置值列入 USB 设备控制层中，该寄存器的配置应放在 ARM 微控制器 LPC2200 的启动代码初始化部分。

当软件包使用不同的数据总线（8 位或 16 位）访问 ISP1181B 时，BCFG2 使用不同的配置，而 ISP1161A1 只能使用 16 位数据总线配置。

程序清单 6.6 配置 Bank2 的存储器配置寄存器 BCFG2

```
    BCFG2 = 0;
#if D13_16BIT_EN
    BCFG2 |= (0x01 << 10)|(0x01 << 28);          //配置 BCFG2 控制寄存器
#else
     BCFG2 |= (0x01 << 10);                      //配置 BCFG2 控制寄存器
#endif
```

同样地，与 D13 数据总线相关的还有 LPC2200 管脚功能选择寄存器 PINSEL2，该寄存器与 D13 的相关性如表 6.4 所示。

表 6.4 配置管脚功能选择寄存器 2 PINSEL2

PINSEL2	配置值	描 述
1：0	00	保留, 与 D13 的配置无关
2	X	与 D13 的配置无关
3	X	与 D13 的配置无关
5：4	10	使能管脚为数据总线
6	X	与 D13 的配置无关
7	X	与 D13 的配置无关
8	1	使能 WE

续上表

10：9	00	保留
11	X	与 D13 的配置无关
12	0	保留
13	X	与 D13 的配置无关
15：14	01	使能 CS2, 初始化代码已进行了配置
17：16	XX	与 D13 的配置无关
19：18	00	保留
20	X	与 D13 的配置无关
21	X	与 D13 的配置无关
22	X	与 D13 的配置无关
23	1	使能 A0 地址线
24	X	与 D13 的配置无关
27：25	XXX	与 D13 的配置无关
31：28	0000	保留

由表 6.4 可见，涉及到 D13 的位有[5:4]，8，[15:14]，23。由于该寄存器的配置可能影响到挂接到数据总线上的其它设备，所以不能单独对该寄存器进行配置。应该在 ARM 微控制器 LPC2200 的启动代码中，综合考虑各种因素对该寄存器进行配置。

基于这样的原因，所以也不将该寄存器的配置列入 USB 设备控制层中，请用户在启动代码中自行添加。

在 LPC2200 的 ADS1.2 工程模板的启动代码中，已对 PINSEL2 已进行了配置，所以不需要再对该寄存器进行配置。

6． 初始化 D13 的中断,设置 D13 的中断服务程序地址

这些工作也不属于本软件包的一部分，因为 D13 中断的优先级需要用户根据实际情况来决定，由用户实现。程序清单 6.7 给出范例代码，代码对 D13 中断的配置只适应于 LPC2200。

程序清单 6.7 初始化 D13 的中断,设置 D13 的中断服务程序地址

```
VICVectCntl1 = (0x20 | 0x0E);          //EINT0 通道分配到 IRQ slot 1
VICVectAddr1 = (INT32U)Usb_Handler;    //设置 EINT0 向量地址
VICIntEnable = 1 << 14;                //允许 EINT0
```

与硬件配置相关的程序就介绍完毕了。

6.2.2 软件包的其它配置

除了以上的硬件配置以外，还有几点需要注意：

（1）每个 C 文件的开始都包括了头文件（#include "config.h"），该头文件是工程模板：ARM Executable Image for UCOSII(for lpc2200)的配置文件，软件包各个 C 文件都默认已加上。

（2）在工程模板的 IRQ.s 文件的 END 语句之前加上下面的语句：

```
    Usb_Handler     HANDLER   Usb_Exception
```

（3）由于 D13 收发大量数据时中断比较频繁，因此必须修改工程模板中允许中断嵌套

的层数，因此 Startup.s 文件的第二行应改为（这里的中断嵌套层数只考虑了 D13 的中断以及 μC/OS-II 的时钟中断）：

```
IRQ_STACK_LEGTH    EQU    9*25       ;每层嵌套需要 9 个字堆栈，允许 25 层嵌套
```

（4）该软件包的控制传输处理使用了 1 个信号量，所以在配置 μC/OS-II 使用的最大事件数时，要考虑到该事件。配置 μC/OS-II 使用的最大事件数用 os_cfg.h 文件中的宏定义 OS_MAX_EVENTS 来定义。

（5）工程模板配置外部 RAM 和外部 FLASH ROM 总线的访问速度是适应于大部分型号的 RAM 与 FLASH ROM 的，而且 RAM 和 FLASH ROM 这两部分分别占用了外部存储器的 Bank0 和 Bank1。必须重新配置访问这些器件的数据总线与地址总线的有关参数，这些参数由 BCFG0 和 BCFG1 寄存器控制，这样程序的运行速度才能跟上 D13 的运行速度。这些值的配置的具体方法见 LPC2200 的数据手册以及你所使用的 RAM 与 FLASH ROM 的数据手册。程序的运行速度主要与 CPU 主频与外部 RAM、FLASH ROM 速度相关。这方面在模板的 target.c 的 TargetResetInit()函数修改。

如程序清单 6.8 所示的配置适合于 EasyARM2200 开发板。程序清单 6.9 的配置则适合于 SmartARM2200 开发板。

程序清单 6.8 配置 TargetResetInit ()函数

```
#ifdef __DEBUG                          //Bank0-RAM, Bank1-Flash, 外部存储器启动系统
    MEMMAP = 0x3;                       //remap
    BCFG0  = 0x10000400;                //外部 RAM
    BCFG1  = 0x10000460;                //外部 Flash
#endif

#ifdef __OUT_CHIP                       //Bank0-Flash, Bank1-RAM, 外部存储器启动系统
    MEMMAP = 0x3;                       //remap
    BCFG0  = 0x10000460;                //外部 FLASH
    BCFG1  = 0x10000400;                //外部 RAM
#endif

#ifdef __IN_CHIP                        //Bank0-RAM,Bank1-Flash , 适用于内部 Flash 存储器启动系统
    MEMMAP = 0x1;                       //remap
    BCFG0  = 0x10000400;                //外部 RAM
    BCFG1  = 0x10000460;                //外部 Flash
#endif
```

程序清单 6.9 配置 TargetResetInit ()函数

```
#ifdef __DEBUG                          //Bank0-RAM,Bank1-Flash, 外部存储器启动系统
    MEMMAP = 0x3;                       //remap
    BCFG0 =  0x10001460;                //外部 RAM
    BCFG1  = 0x10000460;                //外部 Flash
#endif

#ifdef __OUT_CHIP                       //Bank0-Flash,Bank1-RAM, 外部存储器启动系统
```

```
    MEMMAP = 0x3;                   //remap
    BCFG0  = 0x10000460;            //外部 FLASH
    BCFG1  = 0x10001460;            //外部 RAM
#endif

#ifdef __IN_CHIP                    // Bank0-RAM,Bank1-Flash , 适用于内部 Flash 存储器启动系统
    MEMMAP = 0x1;                   //remap
    BCFG0  = 0x10001460;            //外部 RAM
    BCFG1  = 0x10000460;            //外部 Flash
#endif
```

6.3 ZLG/USBD13 软件包 API 函数

在本软件包中，由于端点 1 和端点 2 构成了发送与接收的关系，因此，称这一对端点为端口 1，其发送数据函数为：WritePort1()，接收数据函数为 ReadPort1()。类似，端点 3 和端点 4 合称为端口 2；端点 5 和端点 6 合称为端口 3。

ZLG/D13 软件包软件包提供给用户读写端点的函数为：

```
写端口 1（写端点 1）：INT8U WritePort1(INT32U len,INT8U *sendbuff,INT16U timeout)
读端口 1（读端点 2）：INT8U ReadPort1(INT32U len,INT8U *recbuff,INT16U timeout)
写端口 2（写端点 3）：INT8U WritePort2(INT32U len,INT8U *sendbuff,INT16U timeout)
读端口 2（读端点 4）：INT8U ReadPort2(INT32U len,INT8U *recbuff,INT16U timeout)
写端口 3（写端点 5）：INT8U WritePort3(INT32U len,INT8U *sendbuff,INT16U timeout)
读端口 3（读端点 6）：INT8U ReadPort3(INT32U len,INT8U *recbuff,INT16U timeout)
```

调用读写端点函数：WritePort1()、WritePort2()、WritePort3()、ReadPort1()、ReadPort2()、ReadPort3()一定要遵守以下规则：

读写端点的任务必须保证有机会能调用到以上函数，才能成功接收/发送数据。

这是因为以上函数给出了接收数据的缓冲区或发送数据的缓冲区。只有给出了缓冲区，从主机发来的数据才有“地方（接收缓冲区）”让它存放。否则本软件包将收到的数据都“扔掉”。同理，只有这些函数给出了需要发送的数据在什么“地方（发送缓冲区）”，软件包才能有目的地把数据发送出去。

以上提到的函数都在 D13Driver.c 文件中。与软件包相关的 API 函数介绍如表 2.6.5 到表 2.6.12 所示。

表 2.6.5 Init_D13()函数

函数名称	Init_D13
函数原型	INT8U Init_D13(void)
功能描述	设置 D13 与硬件的连接, 初始化 D13, 复位 D13，初始化相关变量
函数参数	无
函数返回值	0: 初始化成功； 1: 初始化失败
特殊说明和注意点	该函数执行时间比较长，且执行时关闭了中断，所以只允许在 μC/OS-II 开始运行时调用。

表 2.6.6 TaskSetup()函数

函数名称	TaskSetup
函数原型	void TaskSetup(void *pdata)
功能描述	控制传输处理任务
函数参数	void *pdata：任务入口参数
函数返回值	无
特殊说明和注意点	在 μC/OS-II 开始运行时要创建该任务，该任务的堆栈为：OS_STK TaskSetupStk[128]; 该任务的优先级必须高于用户任务，以便 USB 能够及时完成枚举。建议设置该任务优先级设为 0。

表 2.6.7 ReadPort1()函数

函数名称	ReadPort1
函数原型	INT8U ReadPort1(INT32U len,INT8U *recbuff,INT16U timeout)
功能描述	从端点 2 读出 len 个字节
函数参数	INT32U len： 要读出的字节数（取值范围为 0x00000001～0xFFFFFFFF） INT8U *recbuff： 接收数据指针（长度为 len） INT16U timeout： 超时时间, 必须大于等于 0
函数返回值	0: 读出成功； > 0: 读出失败（错误码，见**表 2.6.13**）
特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（读出成功）为止。

表 2.6.8 ReadPort2()函数

函数名称	ReadPort2
函数原型	INT8U ReadPort2(INT32U len,INT8U *recbuff,INT16U timeout)
功能描述	从端点 4 读出 len 个字节
函数参数	INT32U len： 要读出的字节数（取值范围为 0x00000001～0xFFFFFFFF） INT8U *recbuff： 接收数据指针（长度为 len） INT16U timeout： 超时时间, 必须大于等于 0
函数返回值	0: 读出成功； > 0: 读出失败（错误码，见**表 2.6.13**）
特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（读出成功）为止。

表 2.6.9 ReadPort3()函数

函数名称	ReadPort3
函数原型	INT8U ReadPort3(INT32U len,INT8U *recbuff,INT16U timeout)
功能描述	从端点 6 读出 len 个字节
函数参数	INT32U len： 要读出的字节数（取值范围为 0x00000001～0xFFFFFFFF） INT8U *recbuff： 接收数据指针（长度为 len） INT16U timeout： 超时时间, 必须大于等于 0
函数返回值	0: 读出成功； > 0: 读出失败（错误码，见**表 2.6.13**）

特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（读出成功）为止。

表 2.6.10 WritePort1()函数

函数名称	WritePort1
函数原型	INT8U WritePort1(INT32U len,INT8U *sendbuff,INT16U timeout)
功能描述	用端点 1 发送 len 个字节
函数参数	INT32U len: 发送的字节数（取值范围为 0x00000001 ~ 0xFFFFFFFF） INT8U sendbuff: 发送缓冲区指针 INT16U timeout: 超时等待时间, 必须大于等于 0
函数返回值	0: 发送成功； > 0: 发送失败（错误码，见**表 2.6.13**）
特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（发送成功）为止。

表 2.6.11 WritePort2()函数

函数名称	WritePort2
函数原型	INT8U WritePort2(INT32U len,INT8U *sendbuff,INT16U timeout)
功能描述	用端点 3 发送 len 个字节
函数参数	INT32U len: 发送的字节数（取值范围为 0x00000001 ~ 0xFFFFFFFF） INT8U sendbuff: 发送缓冲区指针 INT16U timeout: 超时等待时间, 必须大于等于 0
函数返回值	0: 发送成功； > 0: 发送失败（错误码，见**表 2.6.13**）
特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（发送成功）为止。

表 2.6.12 WritePort3()函数

函数名称	WritePort3
函数原型	INT8U WritePort3(INT32U len,INT8U *sendbuff,INT16U timeout)
功能描述	用端点 5 发送 len 个字节
函数参数	INT32U len: 发送的字节数（取值范围为 0x00000001 ~ 0xFFFFFFFF） INT8U sendbuff: 发送缓冲区指针 INT16U timeout: 超时等待时间, 必须大于等于 0
函数返回值	0: 发送成功； > 0: 发送失败（错误码，见**表 2.6.13**）
特殊说明和注意点	如果用户需要无限等待，那么请在该函数返回超时错误时再次调用该函数，直至函数返回 0（发送成功）为止。

表 2.6.7 至表 2.6.12 中提到的函数返回值所代表的错误码的定义如表 2.6.13 所示，这些错误码的定义在 D13Driver.h 文件中。

表 2.6.13 读写端点 API 函数返回值

错误码宏定义	宏定义值	含义
USB_NO_ERR	0x00	函数执行成功
USB_ERR_NO_CONFIG	0x01	USB 总线未配置完成错误
USB_ERR_BUFF_INVALID	0x02	接收或发送缓冲区错误
USB_ERR_WR_TIMEOUT	0x03	接收或发送超时错误

6.4 EasyUSBD13.dll 动态库

为了方便用户使用 D13，周立功单片机发展有限公司提供了 PC 机端的 D13 的驱动程序，驱动程序在“D13 PC 驱动”目录下，该驱动程序适用于 Windows98/2000/XP。

对于 D13 各端点的读写，可使用周立功单片机发展有限公司提供的 EasyUSBD13.dll 动态库，该动态库在“EasyUSBD13 动态库”目录下，动态库的使用方法请见“EasyUSBD13 动态库”目录下的“EasyUSBD13 说明书.pdf”文件。

ZLG/USBD13 提供的 API 函数与 PC 机端 EasyUSBD13.dll 动态库提供的函数的对应关系如表 6.14 所示。表中每一行的两个函数是发送与接收的关系。

表 6.14 设备端与主机端函数对应表

D13 端点号	ZLG/USBD13 软件包函数	PC 机主机端 EasyUSBD13.dll 接口函数
2	INT8U ReadPort1(INT32U len,INT8U *recbuff, INT16U timeout)	发送数据函数 Multi_WriteDataD13 ()，其中 pipenum 取值为 1
4	INT8U ReadPort2(INT32U len,INT8U *recbuff, INT16U timeout)	发送数据函数 Multi_WriteDataD13 ()，其中 pipenum 取值为 3
6	INT8U ReadPort3(INT32U len,INT8U *recbuff, INT16U timeout)	发送数据函数 Multi_WriteDataD13 ()，其中 pipenum 取值为 5
1	INT8U WritePort1(INT32U len,INT8U *sendbuff, INT16U timeout)	接收数据函数 Multi_ReadDataD13 ()，其中 pipenum 取值为 0
3	INT8U WritePort2(INT32U len,INT8U *sendbuff, INT16U timeout)	接收数据函数 Multi_ReadDataD13 ()，其中 pipenum 取值为 2
5	INT8U WritePort3(INT32U len,INT8U *sendbuff, INT16U timeout)	接收数据函数 Multi_ReadDataD13 ()，其中 pipenum 取值为 4

注意：EasyUSBD13.dll各函数的参数len不得超过1024(目的是防止LPC2200 中断嵌套过多，使系统堆栈溢出)，ZLG/USBD13各函数的参数len无此限制。

第7章 ZLG 驱动中间件使用说明

1.1 ZLG/CF 驱动中间件

CF 存储卡可以工作在 PC Card ATA I/O 模式（I/O 口模式）、PC Card ATA Memory 模式(MEMORY 模式)、True IDE 模式。考虑到 CF 卡在 True IDE 模式下工作时兼容 IDE 硬盘，并且使用的 I/O 引脚较少，因此推荐使用 True IDE 模式，并提供 CF 卡在 True IDE 模式驱动中间件 ZLG/CF。

1.1.1 概述

1、ZLG/CF 简介

ZLG/CF 是广州周立功单片机发展有限公司开发的面向嵌入式系统开发中间件，是 ZLG 系列中间件的重要成员之一。该驱动可以实现对 CF 存储卡及 IDE 硬盘等大容量 ATA 设备读写等基本操作。它使用μC/OS-II 实时操作系统的信号机制来实现一个多任务并行，完全使用 ANSI C 进行编写，可以象μC/OS-II 那样支持多种 CPU。ZLG/CF 已在广州周立功单片机发展有限公司的 EasyARM2200 开发学习套件上调试通过。

2、ZLG/IP 的特点

- 该驱动通过简单的配置可以用于单任务平台或基于μC/OS-II 实时多任务操作系统平台；全部代码由 ANSI C 编写，并且与目标处理器的存储器结构无关（即与无论存储器是大端结构还是小端结构均不影响程序的执行），方便用户移植到自己的目标系统中；ZLG/CF 为嵌入系统设计，如果您有固化手段，它可以嵌入到您的产品中成为产品的一部分。

1.1.2 ZLG/CF 驱动的结构视图

CF 存储卡驱动的结构如图 7.1 所示。

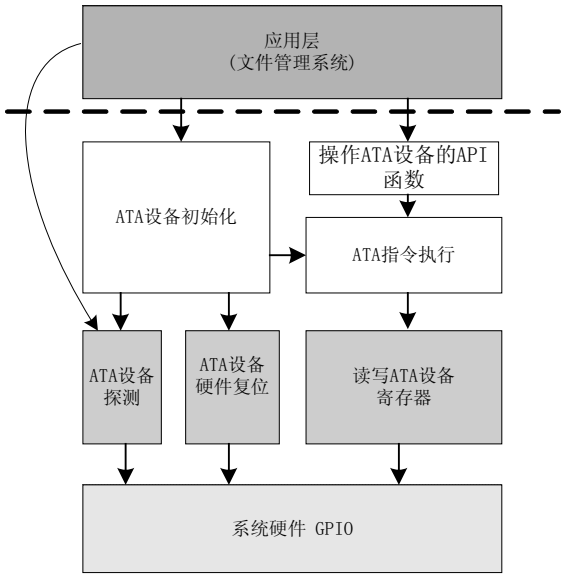


图 7.1 CF 卡驱动结构图

1、系统硬件 GPIO

使用硬件系统的 I/O 引脚模拟 CF 存储卡接口及 IDE 硬盘接口。为读写 ATA 设备寄存器、ATA 设备硬件复位及 ATA 设备控测提供硬件接口。

2、读写 ATA 设备寄存器

控制硬件系统的 I/O 引脚，产生访问 CF 存储卡等 ATA 设备寄存器的读写时序，实现对 ATA 设备寄存器的访问，为 ATA 指令执行提供接口函数。

3、ATA 设备硬件复位

使用系统硬件层提供的硬件接口，控制 CF 存储卡等 ATA 设备进行硬件复位，为 ATA 设备初始化提供接口函数。

4、ATA 设备探测

使用系统硬件层提供的硬件接口，探测 CF 存储卡等 ATA 设备是否存在、插入和移出。使系统支持 CF 存储卡等 ATA 设备的热插拔。

5、ATA 设备初始化

系统起动时或探测到有 CF 存储卡等 ATA 设备插入时，需要使 ATA 设备复位，并初始化设备信息。初始化设备信息是通过执行获取设备识别指令获得。

6、ATA 指令执行

调用 ATA 设备寄存器的读写函数，操作 CF 存储卡的 ATA 指令集实现对 CF 存储卡等 ATA 设备进行操作。为操作 ATA 设备的 API 函数提供函数接口。

7、操作 ATA 设备的 API 函数

调用执行 ATA 指令函数，为应用层提供操作 CF 存储卡等 ATA 设备提供接口函数。使应用层操作 ATA 设备不需要了解 ATA 的协议规范。

8、应用层（文件管理系统）

调用 ZLG/CF 驱动提供的 API 函数，结合 ZLG/FS 等文件管理系统对 CF 存储卡等大容量 ATA 设备进行数据管理。

1.1.3 中间件配置选项

1、基于 uC/OS-II 操作系统

本驱动可使用在单任务系统（不基于操作系统）和基于 uC/OS-II 多任务操作系统，通过对一个宏定义“UCOSII”参数来选择。如果基于 uC/OS-II 需定义配置文件 config.h 中定义宏“UCOSII”，如程序清单 7.1 中的(2)所示。

程序清单 7.1 驱动配置

```
/* CF 卡 IDE 的配置 */
#include    "ide.h"                                    (1)
#define     UCOSII                                     (2)
#define     ATA_BUS_AT_8bit                            (3)
```

2、头文件

使用本驱动前，需在配置文件中包含头文件“IDE.H”如程序清单 7.1 中的（1）所示。

3、使用一个信号量

本驱动需要使用一个信号量事件，在配置 uC/OS-II 中的 OS_MAX_EVENTS 参数时需注意。

4、总线设置

CF 存储卡默认支持 16 位总线，但也可以设置为 8 位。如果使用的是 8 位总线，需要对宏“ATA_BUS_AT_8bit”进行定义，如程序清单 7.1 中的（3）。

1.1.4 API 函数手册

1．硬件接口函数

硬件接口函数是与系统硬件相关的函数集，这部分函数为用户系统硬件与本驱动的接口函数，用户需针对自己的硬件系统来编写这部分函数。

表 7.1 和表 7.2 描述了硬件系统须要提供给本驱动使用的接口函数。

- 硬件复位函数

硬件系统使用 1 个 I / O 口控制 ATA 设备的-RESET 引脚，使 ATA 进行硬件复位。系统须提供 ATA_IdeHardReset()函数给本驱动使用。该函数描述如表 7.1 所示。

表 7.1 SYS_IdeHardReset 函数

函数名称	SYS_IdeHardReset	所属文件	SysATA.C
函数原型	uint8 SYS_IdeHardReset (void)		
功能描述	ATA 设备硬件复位		
编译开关	无		
函数参数	无		
函数返回值	无		
调用模块	Sys_WaitInUS() 延时微秒级函数		
特殊说明和注意要点	使复位引脚 IDE_RST 有效(低电平)保持大于 25 微秒，IDE_RST 无效(高电平)后至少延迟 2 毫秒才可以对 ATA 设备读写寄存器操作。		

范　例	void SYS_IdeHardReset(void) /*基于 ARM LPC2210，GPIO 引脚 IDE_RST*/
	{
	IOCLR = IDE_RST; /* 复位引脚置低 */
	SYS_WaitInUS(30); /* 延时大于 25 微秒 */
	IOSET = IDE_RST; /* 复位引脚置高 */
	SYS_WaitInUS(5000); /* 延时大于 2 毫秒 */
	}		

- ATA 设备寄存器操作函数

对 ATA 设备寄存器的读写是对 ATA 设备最基本操作，而对 ATA 设备各寄存器的读写与系统硬件的连接有直接的关系。使用本驱动，系统须提供与硬件相关的，对设备寄存器操作的底层函数。所需要的接口函数为表 7.2 中的函数。

表 7.2 ATA 设备寄存器操作函数列表

函数	输入数据类型	返回数据类型	描述
GetData()	无	unsigned char / unsigned short	读数据寄存器
SetData(x)	unsigned char / unsigned short	无	写数据寄存器
GetERR()	无	unsigned char	读错误寄存器
SetFeature(x)	unsigned char	无	写特征寄存器
GetSECCNT()	无	unsigned char	读扇区计数寄存器
SetSECCNT(x)	unsigned char	无	写扇区计数寄存器
GetSector()	无	unsigned char	读扇区寄存器
SetSector(x)	unsigned char	无	写扇区寄存器
GetCylinderLow()	无	unsigned char	读柱面低 8 位寄存器
SetCylinderLow(x)	unsigned char	无	写柱面低 8 位寄存器
GetCylinderHigh()	无	unsigned char	读柱面高 8 位寄存器
SetCylinderHigh(x)	unsigned char	无	写柱面高 8 位寄存器
GetDeviceHead()	无	unsigned char	读设备磁头寄存器
SetDeviceHead(x)	unsigned char	无	写设备磁头寄存器
GetStatus()	无	unsigned char	读状态寄存器
SetCommand(x)	unsigned char	无	写命令寄存器
GetAStatus()	无	unsigned char	读辅助状态寄存器
SetControl(x)	unsigned char	无	写控制寄存器

- **ATA 设备探测函数**

如果系统主机在使用 ATA 设备的过程中，CF 卡或 IDE 硬盘被移走，系统主机读得 ATA 设备的数据是不可知的，这样会使系统主机产生误操作。如果要使系统主机支持 ATA 设备热插拔，就很有必要对 ATA 设备的插入及移出的检测。该函数描述如表 7.3 所示。

表 7.3 ATA 设备探测函数

函数名称	SYS_HaveCard	所属文件	SysATA.C
函数原型	uint8 SYS_HaveCard(void)		
功能描述	探测 CF 存储卡的插入及移出		
编译开关	无		
函数参数	无		
函数返回值	TRUE：ATA 设备存在 FALSE：ATA 设备已移出或不存在		
调用模块	无		
特殊说明和注意要点	该函数是必须的，即使硬件系统中没有设计对 ATA 设备探测，也应使该函数一直返回 TRUE。		

（续上表）

范　　例	uint8 SYS_HaveCard(void)
	{
	#define DEVICE_IS_ATTACHED 0 /*硬件系统不探测设备是否存在*/
	/*假设使用 LPC2210 的 P1.23（有 10K 的电上电阻）引脚的 GPIO 输入方式检测设备是否存在，可以如下操作*/
	//#define DEVICE_IS_ATTACHED (IO1PIN & (1<<23))
	
	/*如果 DEVICE_IS_ATTACHED 为 0 却有设备存在，为 1 即没有*/
	if(DEVICE_IS_ATTACHED)
	return FALSE; /*没有设备存在，返回 FALSE*/
	else
	return TRUE; /*有设备存在，返回 TRUE*/
	}

2．用户使用函数

使用本驱动程序，系统可以通过以下函数对 CF 存储卡进行基本操作。

- ATA 设备驱动初始化函数

ATA 设备驱动初始化函数描述见表 7.4。

表 7.4 ATA_INI 函数

函数名称	ATA_INI	所属文件	IDE.C
函数原型	void ATA_INI(void)		
功能描述	ATA 设备驱动初始化，应用于 uCOSII 操作系统，在该函数中创建信号量 CFIDESem。		
编译开关	无		
函数参数	无		
函数返回值	无		
特殊说明和注意要点	在 uCOSII 操作系统中，对 ATA 设备进行操作前，必须先执行该函数。		
范　　例	main() { …. ATA_INI(void); ATA_FindIDEDevice(); …. }		

- 查找 ATA 设备函数

ATA 设备驱动初始化后，应该调用该函数查找可用的 ATA 设备；当系统发现 ATA 设备插入，应调用该函数查找可用的 ATA 设备。查找 ATA 设备函数如表 7.5 所描述。

表 7.5 ATA_FindIDEDevice 函数

函数名称	ATA_FindIDEDevice	所属文件	IDE.C
函数原型	uint8 ATA_FindIDEDevice(void)		
功能描述	对 ATA 接口设备进行进硬件复位，查找 ATA 接口上的设备。		
编译开关	无		
函数参数	无		
函数返回值	TRUE：ATA 设备存在 FALSE：ATA 设备已移出或不存在		

特殊说明和注意要点	发现在新的设接入 ATA 接口时，调用该函数查找设备。
范　　例	见表 7.4 范例。

- **读扇区函数**

读扇区函数作用是从 ATA 设备中读取一或多个扇区，存放在指定的内存中，该函数描述如表 7.6 所示。

表 7.6 ATA_ReadSector 函数

函数名称	ATA_ReadSector	所属文件	IDE.C
函数原型	uint8 ATA_ReadSector(uint8 Device,uint16 *Buffer,uint32 Sector,uint8 count)		
功能描述	从硬盘读取一或多个扇区，放入到形参指定的内存中		
编译开关	无		
函数参数	Buffer，读出数据存放缓冲区地址 Sector，读扇区号 Count，连续读出的扇区数		
函数返回值	TRUE：成功 FALSE：失败		
特殊说明和注意要点	每读出一个扇区值为 512 节字，读出的扇区数为 1~256 个，扇区数为 0 即为 256 个扇区。		

范　　例	// 在从设备 0 号扇区开始连续读出 2 个扇区的数据。
	if(ATA_ReadSector(0x01,Buffer,0x00,0x02)==0)
	{　　/* 错误处理程序 */
	}		

- **写扇区函数**

写扇区函数函数的作用是向 CF 卡及硬盘写入一或多个扇区数据，该函数描述如表 7.7 所示。

表 7.7 ATA_WriteSector 函数

函数名称	ATA_WriteSector	所属文件	IDE.C
函数原型	uint8 ATA_WriteSector(uint8 Device,uint16 * Buffer, uint32 Sector ,uint8 count)		
功能描述	向 CF 卡或硬盘写入一或多个扇区数据		
编译开关	无		
函数参数	Device,选择操作的设备，0 为主设备，1 为从设备 Buffer,需写数的缓冲区地址 Sector,写入的扇区号 count, 写入的扇区数		
函数返回值	TRUE：成功 FALSE：失败		
特殊说明和注意要点	每写入一个扇区值为 512 节字，写入的扇区数为 1~256 个，扇区数为 0 即为 256 个扇区。		

范　　例	// 从主设备 0 号扇区开始连续写入 2 个扇区的数据。
	if(ATA_WriteSector(0,Buffer,0x00,0x02)==0)
	{　　/* 错误处理程序 */
	}		

- **获取设备信息**

每一个 ATA 设备都有自己的描述信息，设备的描述信息包含了设备容量、厂商信息、序列号等，并以一定的数结构保存在 ATA 设备内。获取设备信息函数可以获取 ATA 设备相关的描述信息。获取设备信息函数描述如表 7.8 所示，设备信息结构体成员如表 7.9 所示。

表 7.8 ATA_IdentifyDrive()函数

函数名称	ATA_IdentifyDrive	所属文件	IDE.C
函数原型	uint8 ATA_IdentifyDrive(uint8 Device,void *PTR)		
功能描述	获取 ATA 设备描述信息。		
编译开关	无		
函数参数	Device,选择操作的设备，0 为主设备，1 为从设备 PTR,保存设备信息的缓冲区首址。		
函数返回值	TRUE：成功 FALSE：失败		
特殊说明 和注意要点	PTR 指向的缓冲区大于 512 节字。		

范　　例	if(ATA_IdentifyDrive(SelectDevice,buffer8))
	{
	#define DRIVEINFO ((PCF_IDENTIFY_DATA)buffer8) /*宏定义结构指针*/
	Send("型　　号：");
	for(i=0;i<20;i++)
	{
	printf ("柱　面　数：");
	SendHEXtoASC16(DRIVEINFO->NumberOfCylinders);
	newline();// 换行
	printf ("每柱面磁头数：");
	SendHEXtoASC16(DRIVEINFO->NumberOfHeads);
	newline();// 换行
	printf ("每磁道扇区数：");
	SendHEXtoASC16(DRIVEINFO->SectorsPerTrack);
	newline();// 换行
	}
	else { /*出错处理*/ }		

表 7.9 保存 ATA 设备信息结构成员

成员类型	成员名称	描述信息
unsigned short	GeneralConfiguration	普通设置-CF 卡标志
unsigned short	NumberOfCylinders	默认柱面数
unsigned short	Reserved1	保留
unsigned short	NumberOfHeads	默认磁头数
unsigned short	UnformattedBytesPerTrack	不再使用
unsigned short	UnformattedBytesPerSector	不再使用
unsigned short	SectorsPerTrack	每磁道默认扇区数
unsigned int	SectorsPerCard	每卡扇区数（字 7=MSW,字 8=LSW）
unsigned short	VendorUnique	不再使用
unsigned short	SerialNumber[10]	ASCII 序列号（靠右填写）

续上表

成员类型	成员名称	描述信息
unsigned short	BufferType	不再使用
unsigned short	BufferSectorSize	不再使用
unsigned short	NumberOfEccBytes	读/写长指令时的 ECC 字节数
unsigned short	FirmwareRevision[4]	ASCII 固件版本。大端格式
unsigned short	ModelNumber[20]	ASCII 模式号（靠左填写），大端格式
unsigned short	MaximumBlockTransfer	多重 读/写指令最大扇区号。
unsigned short	Reserved2	保留
unsigned short	Capabilities	功能支持
unsigned short	Reserved3	保留
unsigned short	PioCycleTimingMode	PIO 数据传输周期时序模式
unsigned short	DmaCycleTimingMode	不再使用
unsigned short	TranslationFieldsValid	有效域
unsigned short	NumberOfCurrentCylinders	当前柱面数
unsigned short	NumberOfCurrentHeads	当前磁头数
unsigned short	CurrentSectorsPerTrack	当前每磁道扇区数
unsigned int	CurrentSectorCapacity	当前扇区容量（LBAs）(字 57=LSW，字 58=MSW)
unsigned short	CurrentMultiSectorSetting	多重扇区设置
unsigned int	UserAddressableSectors	LBA 模式下可寻址的扇区总数
unsigned short	Reserved4[2]	保留
unsigned short	AdvancedPIOModes	高级 PIO 模式
unsigned short	Reserved6[2]	保留
unsigned short	MinimumPIOCycleTime	非流控制下最小 PIO 传输周期时间
unsigned short	MinimumPIOCycleTimeIORDY	IORDY 浮动控制小最小 PIO 传输周期时间
unsigned short	Reserved7[10]	保留
unsigned int	AtaVersion	保留－CF 卡不返回 ATA 版本号
unsigned short	FeaturesCommandSupported[3]	特征 / 命令集
unsigned short	FeaturesCommandEnabled[3]	特征 / 命令集
unsigned short	Reserved8	保留
unsigned short	MajorRevision	安全擦除单位完成的时间要求
unsigned short	MinorRevision	增强型安全擦除单位完成的时间要求
unsigned short	PowerManagementValue	当前高级电源管理值
unsigned short	Reserved9[36]	保留
unsigned short	SecurityStatus	安全状态
unsigned short	VendorUniqueBytes[32]	唯一向量字
unsigned short	PowerRequirementDescription	电源要求描述
unsigned short	Reserved10	CFA 的保留分配
unsigned short	KeyManagementSupported	键管理表
unsigned short	Reserved11[83]	CFA 的保留分配

- **设备立即待机**

立即待机函数可以使 ATA 设备立即从空闲状态进入特机（CF 存储卡的睡眠模式）状态

（使 IDE 硬盘停止转动）。该函数描述如表 7.10 所示。

表 7.10 ATA_StandbyImmediate 函数

函数名称	ATA_StandbyImmediate	所属文件	IDE.C
函数原型	uint8 ATA_StandbyImmediate(uint8 Device)		
功能描述	使 ATA 设备立即进入特机状态。		
编译开关	无		
函数参数	Device,选择操作的设备，0 为主设备，1 为从设备		
函数返回值	TRUE：成功 FALSE：失败		
特殊说明 和注意要点	如果该设备不支持电源管理特性设置，执行该指令将发生指令中止出错。		

范　　例	if(ATA_StandbyImmediate(SelectDevice))
	　printf("进入待机，硬盘停止转动。\n");
	else
	　printf ("设备不支持电源管理特性，或设备不可用。\n");		

- **设备立即空闲**

立即空闲函数可以使 ATA 设备立即从待机状态返回空闲状态（使 IDE 硬盘电机转动）。该函数描述见表 7.11。

表 7.11 ATA_IdleImmediate()函数

函数名称	ATA_ IdleImmediate	所属文件	IDE.C
函数原型	uint8 ATA_ IdleImmediate (uint8 Device)		
功能描述	使 ATA 设备立即返回空闲状态。		
编译开关	无		
函数参数	Device,选择操作的设备，0 为主设备，1 为从设备		
函数返回值	TRUE：成功 FALSE：失败		
特殊说明 和注意要点	如果该设备不支持电源管理特性设置，执行该指令将发生指令中止出错。		

范　　例	if(ATA_ IdleImmediate (SelectDevice))
	　Send("设备返回空闲，硬盘开始转动。\n");
	else
	　Send("设备不支持电源管理特性，或设备不可用。\n");		

第8章 读写U盘模块驱动的使用

8.1 读写U盘模块简介

读写U盘模块由硬件及软件两部分组成。硬件使用的是ISP1161BM PACK，软件使用HostMassLib.a驱动库。

ISP1161BM PACK是使用PHILIPS公司生产的USB主控芯片ISP1161BM，该芯片是一款兼容USB 2.0版规范的USB HC（Host Controller：主控器）。ISP1161A1有2个下行口，每个下行口都拥有独立的过流检测输入引脚及电源开关控制输出引脚。ISP1161A1同时也提供唤醒输入引脚及挂机状态输出引脚，使电源供电管理更加灵活。主控器的下行口可以与具有上行口的USB设备及USB集线器连接。

ISP1161A1适合使用于具有USB主机功能的嵌入式系统及便携式设备，它的出现使系统具有更高的灵活性。一个嵌入ISP1161A1的系统可以直接与具有USB上行口的设备连接，如：USB打印机，USB相机，USB键盘，及USB鼠标等。

HostMassLib是基于ZLG USB HOST STACK及ZLG Mass Storage Class驱动封装的ISP1161A1读写U盘驱动库。该库使用RAM资源少于4K，使用CODE资源少于12.4K，可同时支持两个U盘。

硬件功能特点：

- 符合通用串总线2.0规范
- 支持全速（12Mbit/s）及低速（1.5Mbit/s）两种数据模式
- 两个下行端口，可同时操作两个USB设备
- 工业级工作温度-40度到+85度

驱动功能特点：

- 支持μC/OS-II多任务操作系统
- 同时支持两个大容量设备（如：U盘、移动软驱）
- 仅需RAM：4K和CODE：12.4K
- 支持通用文件管理系统（如：ZLG/FS）

8.2 固件驱动说明

8.2.1 初始化配置

HostMassLib库最多使用12个μC/OS-II事件，所以需在OS_CFG.H文件中配置μC/OS-II事件OS_MAX_EVENTS值不能小于12。

创建一个USB主机服务任务，如程序清单 8.1所示。该任务使用的优先级应比其它使用USB主机传输任务的优先级高；主机服务任务推荐使用堆栈长度为256。

程序清单 8.1 USB 主机任务

```
/*********************************************************************************************
**                                    USB 主机任务
*********************************************************************************************/
void Task_USB_Host_Serve(void *pdata)
{
    pdata = pdata;
```

```
    while (1)
    {
        USB_Host_Serve();
    }
}
```

8.2.2 用户编写的接口函数

1. 写命令寄存器函数

写命令寄存器函数如表 8.1 所示。

表 8.1 写命令寄存器函数

函数名称	***outcommand***	所属文件	
函数原型	void outcommand (unsigned short data)		
功能描述	写 ISP116X 命令寄存器函数。		
编译开关	无		
函数参数	data，需写入的数据。		
函数返回值	无		
特殊说明 和注意要点	无		
范 例	void outcommand(unsigned short data) { * (volatile unsigned short *) HC_COMMAND_ADDRESS = data; }		

2. 写数据寄存器函数

写数据寄存器函数如表 8.2 所示。

表 8.2 写数据寄存器函数

函数名称	***outdata***	所属文件	
函数原型	void outdata(unsigned short data)		
功能描述	写 ISP116X 数据寄存器函数。		
编译开关	无		
函数参数	data，需写入的数据。		
函数返回值	无		
特殊说明 和注意要点	无		
范 例	void outdata(unsigned short data) { * (volatile unsigned short *) HC_DATA_ADDRESS = data; }		

3. 读数据寄存器函数

读数据寄存器函数，如表 8.3 所示。

表 8.3 读数据寄存器函数

函数名称	***indata***	所属文件	
函数原型	unsigned short indata(void)		
功能描述	读 ISP116X 数据寄存器函数。		
编译开关	无		
函数参数	无		
函数返回值	从数据寄存器读写的 16 位数位		
特殊说明 和注意要点	无		
范　　例	unsigned short indata(void) {　　return *((volatile unsigned short*) HC_DATA_ADDRESS); }		

4. 禁止中断函数

禁止中断函数，如表 8.4 所示。

表 8.4 禁止系统中断

函数名称	***disable***	所属文件	
函数原型	void disable(void)		
功能描述	禁止系统中断		
编译开关	无		
函数参数	无		
函数返回值	无		
特殊说明 和注意要点	无		
范　　例	void disable(void) {　　OS_ENTER_CRITICAL();　　// μC/OS-II 关系统中断处理函数 }		

5. 使能中断函数

使能中断函数，如表 8.5 所示。

表 8.5 使能中断函数

函数名称	***enable***	所属文件	
函数原型	void enable (void)		
功能描述	使能中断函数		
编译开关	无		
函数参数	无		

续上表

函数返回值	无
特殊说明 和注意要点	无
范　　例	void enable (void) {　　OS_EXIT_CRITICAL ();　　// μC/OS-II 关系统中断处理函数 }

6. 调用中断服务函数

用户需编写一个中断函数，并在中断函数中调用 USB 主机中断服务函数 isr_USB_Hc，如表 8.6 所示。

表 8.6 主机中断服务函数

函数名称	***isr_USB_Hc***	所属文件	
函数原型	void isr_USB_Hc(void);		
功能描述	主机中断服务函数		
编译开关	无		
函数参数	无		
函数返回值	无		
特殊说明 和注意要点	无		

范　　例	void ISP1160_Exception(void)
	{
	　　isr_USB_Hc();　　// 与 ISP116X 驱动接口的中断处理函数
	　　EXTINT = 0x04;　　// EINT2
	　　VICVectAddr = 0;
	}		

7. 硬件复位函数

硬件复位函数，如表 8.7 所示。

表 8.7 硬件复位函数

函数名称	***Sys_Hc_RESET***	所属文件	
函数原型	void Sys_Hc_RESET (void)		
功能描述	使能中断函数		
编译开关	无		
函数参数	无		
函数返回值	无		
特殊说明 和注意要点	该函数需在 ISP116X 初始化之前被调用一次。如果使用 ISP116X 上电复位则可以不使用该函数。		

续上表

范　　例	void Sys_Hc_RESET(void)
	{
	IO1CLR = HcRESET; // 置低与 ISP116X 复位引脚连接的 IO 口
	Sys_WaitinMS(2);
	IO1SET = HcRESET; // 置高与 ISP116X 复位引脚连接的 IO 口
	Sys_WaitinMS(2);
	}

8.2.3 API 函数

表 8.8、表 8.9、表 8.10、表 8.11、表 8.12、表 8.13、表 8.14 和表 8.15 分别描述了大容量类驱动提供的 API 函数。使用以这些 API 函数前需包含“HostStack.h”头文件。

表 8.8 USB 主机初始化函数

函数名称	***USB_Host_Stack_Initialize***	所属文件	
函数原型	void USB_Host_Stack_Initialize(void);		
功能描述	USB 主机驱动初始化。		
编译开关	无		
函数参数	无		
函数返回值	无		
特殊说明 和注意要点			
范　　例			

表 8.9 查找大容量设备

函数名称	***Creat_Medium***	所属文件	
函数原型	device_instance *find_mass_device(unsigned char Index)		
功能描述	查找大容量类设备，。		
编译开关	无		
函数参数	Index，逻辑单元索引，0 为第 1 个逻辑单元。		
函数返回值	dvi_ptr，大容量设备的设备描述信息结构指针； 设备不存在，返回一个空指针。		
特殊说明 和注意要点			
范　　例			

表 8.10 Creat_Medium()函数

函数名称	***Creat_Medium***	所属文件	
函数原型	hMedLUN * Creat_Medium(device_instance *dvi_ptr ,unsigned char LUNIndex);		
功能描述	创建（获取）大容量设备逻辑单元描述符指针。		

续上表

编译开关	无
函数参数	dvi_ptr，大容量设备的设备描述信息结构指针； LUNIndex，逻辑单元索引，0 为第 1 个逻辑单元。
函数返回值	逻辑单元存在，返回逻辑单元描述符指针； 逻辑单元不存在，返回一个容指针。
特殊说明 和注意要点	
范　　例	

表 8.11 关闭大容量设备逻辑单元

函数名称	***Close_Medium***	所属文件	
函数原型	void Close_Medium(hMedLUN *hMedLUNPtr,unsigned char LUNIndex)		
功能描述	关闭大容量设备逻辑单元		
编译开关	无		
函数参数	hMedLUNPtr，大容量设备的逻辑单元描述符指针； LUNIndex，逻辑单元索引，0 为第 1 个逻辑单元。		
函数返回值	无		
特殊说明 和注意要点	如果再次打开一个大容量逻辑单元前，必须先使用该函数关闭该逻辑单元。否则将创建失败。		
范　　例			

表 8.12 获取设备总扇区数

函数名称	***GetLastSectorAddr***	所属文件	
函数原型	unsigned int GetLastSectorAddr(hMedLUN *hMedLUNPtr)		
功能描述	获取逻辑单元最后一个扇区地址。		
编译开关	无		
函数参数	hMedLUNPtr，大容量设备的逻辑单元描述符指针		
函数返回值	最后一个扇区地址		
特殊说明 和注意要点	使用该函数前需成功创建一个逻辑单元，关闭逻辑单元其返回值为 0。		
范　　例			

表 8.13 获取扇区大小

函数名称	***GetSectorSize(hMedLUN *hMedLUNPtr)***	所属文件	
函数原型	unsigned short GetSectorSize***(***hMedLUN *hMedLUNPtr***)***		
功能描述	获取大容量存储类设备逻辑单元扇区大小，单位为字节		
编译开关	无		
函数参数	hMedLUNPtr，大容量设备的逻辑单元描述符指针		
函数返回值	扇区大小字节数。		

续上表

特殊说明 和注意要点	使用该函数前需成功创建一个逻辑单元，关闭逻辑单元其返回值为 0。
范　　例	

表 8.14 ReadBlockData()函数

函数名称	***ReadBlockData***	所属文件	
函数原型	unsigned short ReadBlockData(hMedLUN *MediumPtr, unsigned char *BufferPtr, unsigned int LBA, unsigned short TrBLength);		
功能描述	从逻辑单元设备指定的逻辑块地址读出数据。		
编译开关	无		
函数参数	MediumPtr，逻辑单元描述符指针； BufferPtr，数据缓冲区指针； LBA，逻辑块地址； TrBLength，读取逻辑块个数，值范围为：1~127。		
函数返回值	实际读入的字节数，设备不存在或未准备好返回 NULL。		
特殊说明 和注意要点	TrBLength 不能大于 127。		
范　　例			

表 8.15 WriteBlockData()函数

函数名称	***WriteBlockData***	所属文件	
函数原型	unsigned short WriteBlockData(hMedLUN *MediumPtr, unsigned char *BufferPtr, unsigned int LBA, unsigned short TrBLength);		
功能描述	往逻辑单元设备指定的逻辑块地址写入数据。		
编译开关	无		
函数参数	MediumPtr，逻辑单元描述符指针； BufferPtr，数据缓冲区指针； LBA，逻辑块地址； TrBLength，写逻辑块个数，值范围为：1~127。		
函数返回值	实际读入的字节数，设备不存在或未准备好返回 NULL。		
特殊说明 和注意要点	TrBLength 不能大于 127。		
范　　例			

第9章 MiniGUI 图形界面实验

9.1 MiniGUI for uC/OS-II 移植实验

1. 实验目的

(1) 了解 MiniGUI 的基本特点和 MiniGUI 源文件目录结构。
(2) 掌握 MiniGUI for uC/OS-II 移植的基本方法和应用配置。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

学习移植 MiniGUI-STR(for uC/OS-II)软件到 SmartARM2200 开发板的基本方法，然后编写一个简单的 MiniGUI 应用程序，使用 MessageBox 函数创建一个消息框，消息框中显示字符串“Hello MiniGUI!”。

4. 实验预习要求

(1) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。
(2) 阅读 MiniGUI 的文档<<MiniGUI 用户手册>>和<<MiniGUI 编程指南>>，了解 MiniGUI 的基本特点和应用配置选项，以及应用程序的编写。

5. 实验原理

(1) MiniGUI 简介

MiniGUI 项目的最初目标是为基于 Linux 的实时嵌入式操作系统提供一个轻量级的图形用户界面支持系统，发展到今天，MiniGUI 已进入成熟和稳定阶段，并且所支持的操作系统已不仅仅限于 Linux，它还可以在 uClinux、uC/OS-II、eCos、VxWorks 等系统上运行。

MiniGUI 为应用程序定义了一组轻量级的窗口和图形设备接口，利用这些接口，每个应用程序可以建立多个主窗口，然后在这些主窗口中创建按钮、编辑框等控制。MiniGUI 还为用户提供了丰富的图形功能，以显示各种格式的位图并在窗口中绘制复杂图形。

MiniGUI 和嵌入式操作系统的关系如图 9.1 所示，**基于 MiniGUI 的应用程序一般通过 ANSI C 库以及 MiniGUI 自身提供的 API 来实现自己的功能**。

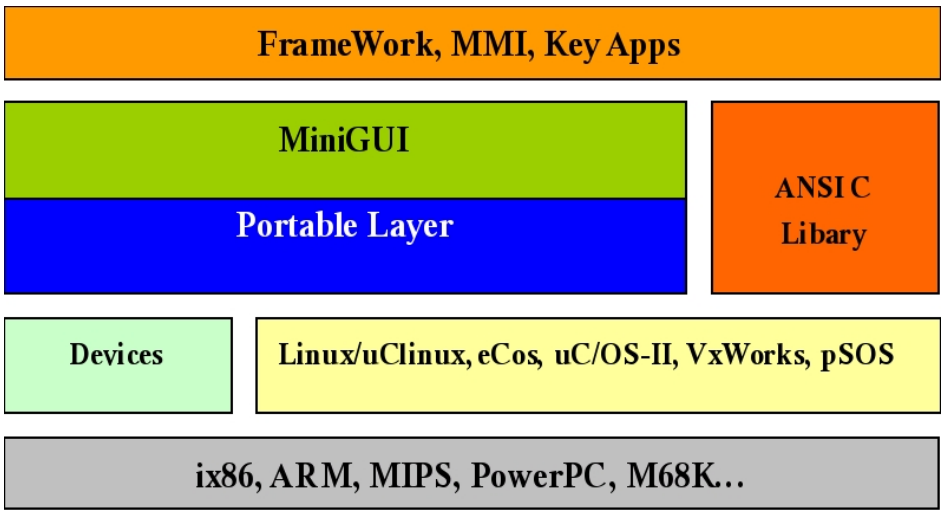


图 9.1 MiniGUI 和嵌入式操作系统的关系

MiniGUI 是由北京飞漫软件技术有限公司拥有版权并主持和维护的自由软件，相关信息可访问 www.minigui.com 网站。

(2) MiniGUI-STR 功能特性

MiniGUI-STR 版本具有以下功能特性：

- 支持 MiniGUI-Threads、MiniGUI-Lite 和 MiniGUI-Standalone 三种运行模式；
- 完整的窗口/消息 API 及基本的绘图 API；
- 两种 GAL 图形引擎：FrameBuffer 和 qvfb；
- 字符集的支持：ISO8859-1～ISO8859-16、GB2312 和 BIG5；
- RBF 及 VBF 字体的支持，在资源包中包含设备字体：
  rbf-fixed-rrncnn-8-16-ISO8859-1、rbf-fixed-rrncnn-16-16-GB2312.1980-0、
  rbf-fixed-rrncnn-16-16-BIG5、vbf-Courier-rrncnn-10-15-ISO8859-1、
  vbf-Helvetica-rrncnn-15-16-ISO8859-1、vbf-Times-rrncnn-13-15-ISO8859-1
- 支持 Windows BMP、GIF、JPEG 和 PNG 图像文件格式；
- 窗口支持 PC 3D、Flat 和 Phone 三种风格；
- 对键盘布局的支持；
- 对 GB2312 全拼输入法的支持；
- 基本控制包括：
  Static、Button、Simple edit box、Single-line edit box、Multiple-line edit box、
  ListBox、ComboBox、ProgressBar、NewToolbar、MenuButton、TrackBar、
  PropertySheet、ScrollView

说明：由于 uC/OS-II 操作系统本身特点和限制，MiniGUI-STR for uC/OS-II 版本有些功能是不能使用的，比如只支持 MiniGUI-Threads 运行模式、只能使用内建资源、不支持 JPEG、PNG 图像文件、不能使用 FrameBuffer 图形引擎等等。

(3) MiniGUI-STR 软件包

用于 SmartARM2200 教学实验开发平台的 MiniGUI-STR for uC/OS-II 共有 2 个目录，说明如下：

- MiniGUI_Lib，MiniGUI-STR 函数库源代码，即 libminigui。

- Libc，针对于 uC/OS-II 操作系统的 POSIX 线程库。

(4) MiniGUI 的使用

要在目标板上运行 MiniGUI for uC/OS-II 应用程序，首先要使用缩主机(即 PC 机)对 MiniGUI 的源代码进行编译(交叉编译)，生成 MiniGUI 静态链接库文件(如 MiniGUI_Lib.a)。然后，MiniGUI 应用程序也在缩主机上进行交叉编译，并且与 MiniGUI 静态链接库、POSIX 线程库链接生成*.axf 映象文件或*.bin 二进制文件。最后，将 MiniGUI 应用程序(*.axf 映象文件或*.bin 二进制文件)下载到目标板，并运行。

当然，由于不同目标机的图形显示设备和输入设备是不一致的，所以用户需要为 MiniGUI 编写相应的图形引擎和输入引擎(即驱动程序)，再与 MiniGUI 的源代码一起交叉编译生成 MiniGUI 静态链接库文件。

(5) MiniGUI 的 GAL 的移植

从 MiniGUI v0.3.xx 版本开始，MiniGUI 引入了图形抽象层和输入抽象层(Graphics and Input Abstract Layer，即 GAL 和 IAL)的概念，它定义了一组不依赖于任何特殊硬件的抽象接口，所有顶层的的图形操作和输入处理都建立在抽象接口之上。而用实现这一抽象接口的底层代码称为“图形引擎”和“输入引擎”，类似于操作系统中的驱动程序。

MiniGUI-STR 不支持 newgal 功能，所以需要按照老的 GAL 接口来编写适合于 SmartARM2200 开发板的图形引擎。首先，在 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\gal\native 目录下的 commlcd.c 文件中编写老的 GAL 接口的几个接口函数，代码参考程序清单 9.1。

程序清单 9.1 移植的图形引擎—GAL 接口函数

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "native.h"

#ifdef  _NATIVE_GAL_COMMLCD
#include "lcddrive.h"
#include "fb.h"

static PSD fb_open(PSD psd)
{
    PSUBDRIVER subdriver;

    TftInit();

#if SWAP_XY_EN == 1
    psd->xres = GUI_LCM_YMAX;
    psd->yres = GUI_LCM_XMAX;
#else
    psd->xres = GUI_LCM_XMAX;
    psd->yres = GUI_LCM_YMAX;
```

```
#endif
        psd->planes = 1;
        psd->bpp = 16;
        psd->linelen = psd->xres * 2;
        psd->ncolors = 1 << 16;

        if (!(psd->addr = malloc(GUI_LCM_XMAX * GUI_LCM_YMAX * 2))) {
                fprintf(stderr, "ERROR: Couldn't allocate buffer!(COMMLCD_VideoInit) \n");
                  return -1;
        }

        psd->gr_mode = MODE_SET;
        psd->size = 0;
        psd->flags = PSF_MEMORY;
        psd->flags |= PSF_MSBRIGHT;
        psd->pixtype = PF_TRUECOLOR565;

        subdriver = select_fb_subdriver(psd);
        if (!subdriver) {
                fprintf(stderr,"GAL Common LCD engine: No driver for bpp %d\n", psd->bpp);
                goto fail;
        }

        if (!set_subdriver(psd, subdriver, TRUE)) {
                fprintf(stderr,"GAL Common LCD engine: Driver initialize failed for bpp %d\n", psd->bpp);
                goto fail;
        }

        return psd;

fail:
        return NULL;
}

/* close framebuffer*/
static void fb_close (PSD psd)
{
        if (psd->addr)
                free(psd->addr);
        psd->addr = NULL;
}

static void fb_setpalette (PSD psd, int first, int count, GAL_Color *palette)
{
```

```
}

static void fb_getpalette (PSD psd, int first, int count, GAL_Color *palette)
{
}

static void update_rect(PSD psd, int l, int t, int r, int b)
{
    int    bak;

     if(l > r)
     {
          bak = l;
          l = r;
          r = bak;
     }
     if(t > b)
     {
          bak = t;
          t = b;
          b = bak;
     }

     TFT_UpdateRects(l, t, r +1 - l, b +1 - t, psd->addr);
}

SCREENDEVICE commlcd = {
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0,
    NULL, NULL,
    fb_open,
    fb_close,
    fb_setpalette,
    fb_getpalette,
    native_gen_allocatememgc,
    fb_mapmemgc,
    native_gen_freememgc,
    native_gen_clippoint,
    native_gen_fillrect,
    NULL,              /* DrawPixel subdriver */
    NULL,              /* ReadPixel subdriver */
    NULL,              /* DrawHLine subdriver */
```

```
    NULL,               /* DrawVLine subdriver */
    NULL,               /* Blit subdriver */
    NULL,               /* PutBox subdriver */
    NULL,               /* GetBox subdriver */
    NULL,               /* PutBoxMask subdriver */
    NULL,               /* CopyBox subdriver */
    update_rect
};

#endif /* _NATIVE_GAL_COMMLCD */
```

在程序清单 9.1 中，宏 GUI_LCM_XMAX、GUI_LCM_YMAX 和 SWAP_XY_EN 均在 lcddrive.h 头文件中定义，分别表示液晶 x 轴和 y 轴方向的点像素，x 轴和 y 轴是否交换。比如 240x320 液晶屏，则定义 GUI_LCM_XMAX 为 240，GUI_LCM_YMAX 为 320。如果定义 SWAP_XY_EN 为 1，则液晶驱动程序(lcddrive.c 文件)将会对显示数据按 x、y 轴方向进行交换，使液晶屏变成了 320x240 显示模式。初始化液晶模块函数 TftInit 和更新液晶显示函数 TFT_UpdateRects 均在液晶驱动程序(lcddrive.c 文件)中实现。

然后，在 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\gal\native 目录建立液晶驱动程序文件 lcddrive.c 和 lcddrive.h，液晶驱动程序是根据液晶模块和硬件连接电路来设计的，具体代码请参考产品光盘上的源文件。由于在 lcddrive.c 文件中使用了 LPC2294 的头文件 lpc2294.h，所以要将此文件复制到此目录下。

(6) MiniGUI 的 IAL 的移植

MiniGUI 通过 INPUT 数据结构来表示输入引擎，此数据结构的定义是在 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\include\ial.h 文件中，具体代码请参考产品光盘上的源文件。INPUT 数据结构定义了很多函数指针，这些函数指针就是用来指向设定的 IAL 的对应函数，从而实现硬件的输入。

在 INPUT 数据结构中，MiniGUI 是使用 init_input、term_input 两个函数指针作为与 IAL 可见的接口，所以用户 IAL 程序通用接口只需要提供这两个函数(因为调用 init_input 程序正确初始化后，MiniGUI 就可以间接访问用户 IAL 的其它功能函数)。

首先，在 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\ial 目录下 comminput.c 文件中编写 IAL 接口的几个接口函数，代码参考程序清单 9.2。

程序清单 9.2 移植的输入引擎—IAL 接口函数

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

#ifdef  _COMM_IAL

#include "minigui.h"
#include "misc.h"
#include "ial.h"
#include "comminput.h"
```

```
//////////////////////////////////////////////////////////////////////////
// COMM_IAL 相关函数声明 (在 mgdrv-ucosii.c 中实现)

extern int comm_ts_getdata (short *x, short *y, short *button);
extern int comm_kb_getdata (short *key, short *status);
extern int comm_wait_for_input(void);

//////////////////////////////////////////////////////////////////////////
// 引擎所所需要的变量
static int MOUSEX = 0, MOUSEY = 0, MOUSEBUTTON = 0;        // 鼠标
static short KEYCODE = 0, KEYSTATUS = 0;                   // 按键

// 通知底层引擎更新鼠标信息 (返回 1)
static int mouse_update (void)
{
    return 1;
}

// 获得最新的鼠标(x,y)坐标值
static void mouse_getxy (int *x, int* y)
{
    *x = MOUSEX;
    *y = MOUSEY;
}

// 获取鼠标按钮状态
// 该函数的返回值可以是 IAL_MOUSE_LEFTBUTTON(表示左键按下)、
// IAL_MOUSE_MIDDLEBUTTON(表示中键按下)、IAL_MOUSE_RIGHTBUTTON
// (表示右键按下)等值"或"的结果。
static int mouse_getbutton (void)
{
    return MOUSEBUTTON;
}

//////////////////////////////////////////////////////////////////////////
static unsigned char kbd_state[NR_KEYS];    // 键盘状态缓冲区

// 通知底层引擎更新键盘信息 (更改 kbd_state 数组，表明有按键)
static int keyboard_update (void)
{
    kbd_state[KEYCODE] = KEYSTATUS;
    return KEYCODE + 1;
}
```

```
// 获取键盘状态，返回一个字符数组(包括多个按键的状态)，其中包含以扫描码索引的键盘按键状态。
static const char* keyboard_getstate (void)
{
    return kbd_state;
}

////////////////////////////////////////////////////////////////////////////
static int wait_event( int which, fd_set *in, fd_set *out, fd_set *except,
                       struct timeval *timeout)
{
    int retvalue;

    retvalue = comm_wait_for_input ();      // 查询键盘或鼠标事件(需要挂起任务)

    if (retvalue & IAL_MOUSEEVENT)     // 鼠标事件
    {
        // 如果是触摸屏，则有 button 按键时才会更新 MOUSEX、MOUSEY
        comm_ts_getdata (&MOUSEX, &MOUSEY, &MOUSEBUTTON);
        retvalue = IAL_MOUSEEVENT;
    }
    else
    {
        if (retvalue & IAL_KEYEVENT)        // 键盘事件
        {
            comm_kb_getdata (&KEYCODE, &KEYSTATUS);
            if (kbd_state[KEYCODE] == KEYSTATUS) return -1;
            retvalue = IAL_KEYEVENT;
        }
        else
        {
            retvalue = -1;
        }
    }

    return(retvalue);
}

////////////////////////////////////////////////////////////////////////////
// 输入引擎的初始化函数。该函数负责对 INPUT 结构的其它成员赋值。
BOOL InitCOMMInput (INPUT* input, const char* mdev, const char* mtype)
{
    // 在此函数调用之前必须初始化输入设备硬件。
    // (可以在此调用初始化函数)
```

```
    input->update_mouse = mouse_update;
    input->get_mouse_xy = mouse_getxy;
    input->set_mouse_xy = NULL;                 // 上层调用该函数可以设置鼠标位置到新的坐标值。
    input->get_mouse_button = mouse_getbutton;
    input->set_mouse_range = NULL;              // 设置鼠标的活动范围。
    input->suspend_mouse= NULL;
    input->resume_mouse = NULL;

    input->update_keyboard = keyboard_update;
    input->get_keyboard_state = keyboard_getstate;
    input->suspend_keyboard = NULL;             // 暂停键盘设备的读取，用于虚拟控制台切换。
    input->resume_keyboard = NULL;              // 继续键盘设备读取，用于虚拟控制台切换。
    input->set_leds = NULL;                     // 设置键盘的锁定 LED。
    input->wait_event = wait_event;             // 上层调用该函数等待底层引擎上发生输入事件
    return(TRUE);
}

// 输入引擎的终止清除函数
void TermCOMMInput (void)
{
}

#endif /* _COMM_IAL */
```

如程序清单 9.2 所列，在等待底层引擎发生输入事件函数 wait_event 中，先调用 comm_wait_for_input 函数等待输入事件，然后根据事件类型调用 comm_ts_getdata 函数或 comm_kb_getdata 函数来读取鼠标或键盘信息。comm_wait_for_input、comm_ts_getdata 和 comm_kb_getdata 函数均在底层键盘驱动程序(comm_drive.c 文件)中实现。

然后，在 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\ial 目录建立键盘驱动程序文件 comm_drive.c，实现键盘的读取和键码转换，具体代码请参考产品光盘上的源文件。键盘驱动程序文件 comm_drive.c 把 SmartARM2200 开发板上的 16 个按键定义为 0～9 和模拟鼠标(移动键←、↓、→、↑，左键 LB，右键 RB)等功能，如图图 9.2 所示。

LB (S13 键)	**↑** **(S14 键)**	**RB** (S15 键)	**0** (S16 键)
← (S9 键)	**↓** **(S10 键)**	**→** **(S11 键)**	**9** (S12 键)
5 (S5 键)	**6** (S6 键)	**7** (S7 键)	**8** (S8 键)
1 (S1 键)	**2** (S2 键)	**3** (S3 键)	**4** (S4 键)

图 9.2 输入引擎定义的按键功能

(7) 交叉编译 MiniGUI

运行时配置文件

MiniGUI 具有众多的资源，比如各种字体、光标、图形引擎和输入引擎等等，支持用户运行时(相对于编译时而言)进行相应的系统参数配置。MiniGUI for uC/OS-II 是使用内建式资源，其运行时配置文件为 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\src\sysres\mgetc.c 文件，此文件需要根据实际硬件平台进行修改。为了让用户最少的改动 mgetc.c，可以把与硬件平台相关的配置项抽取出来，独立保存到一个以目标硬件平台命名的文件中(比如 mgetc-ucosii-xxx.c)，然后在 mgetc.c 包含(include)此文件即可。对于 SmartARM2200 教学实验开发平台，在 mgetc.c 文件中包含目标板的特定配置文件 mgetc-ucosii-SmartARM2200.c，文件内容如程序清单 9.3 所列。

程序清单 9.3 针对目标板的配置文件

```
#if defined(__UCOSII__)

static char *SYSTEM_VALUES[] = {"commlcd", "comm", "/dev/mouse", "IMPS2"};

static char *FBCON_VALUES[] = {"240x320-16bpp"};

static char *SYSTEMFONT_KEYS[] =
{"font_number", "font0", "font1", "font2", "default", "wchar_def", "fixed", "caption", "menu", "control"};

static char *SYSTEMFONT_VALUES[] =
{
    "3","rbf-fixed-rrncnn-8-16-ISO8859-1", "*-fixed-rrncnn-*-16-GB2312", "*-Helvetica-rrncnn-*-16-GB2312",
    "0", "1", "1", "2", "2", "2"
};

#endif
```

如程序清单 9.3 所列，SYSTEM_VALUES 数组的第 1 项用来指定图形引擎，第 2 项用来指定输入引擎，第 3、4 项是鼠标设备和协议类型，在 MiniGUI for uC/OS-II 中这两项没有意义。FBCON_VALUES 数组用于指定液晶屏的分辨率和颜色深度。SYSTEMFONT_KEYS 数组和 SYSTEMFONT_VALUES 数组用于定义 MiniGUI 使用的系统字体，各项的具体含义请参考《MiniGUI 用户手册》的运行时配置文件说明部分。

编译配置

MiniGUI 是可配置的(比如是否支持某种字体，是否支持光标显示，使用哪个 IAL 输入引擎等等)，MiniGUI for uC/OS-II 的配置文件为 MiniGUI_Lib\libminigui-str-1.6.2-ucosii\config.h 文件，通过配置此文件，然后编译 MiniGUI(for uC/OS-II)函数库源代码，即可生成具有相应功能的库文件。以下介绍部分重要配置宏的功能。

_INCORE_RES：内建资源支持，uC/OS-II 不包含文件系统，此宏必须定义为 1；

_DUMMY_IAL：若需要使用 dummy 输入引擎，则定义此宏为 1；

_COMM_IAL：MiniGUI for uC/OS-II 的通用输入引擎(IAL 移植就是使用这个引擎)，使用时定义此宏为 1；

_NATIVE_GAL_COMMLCD：MiniGUI for uC/OS-II 的通用图形引擎(GAL 移植就是使用这个引擎)，使用时定义此宏为 1；

_PC3D_WINDOW_STYLE：MiniGUI for uC/OS-II 的窗口风格设置，若此宏定义为 1，表示使用手持终端的风格(pc3d)；

_PHONE_WINDOW_STYLE：MiniGUI for uC/OS-II 的窗口风格设置，若此宏定义为 1，表示使用类似于 PC 的三维风格(phone)；

_GB_SUPPORT：GB2312 中文字符集支持。如果需要使用中文，定义此宏为 1；

_RBF_SUPPORT：RBF 字体支持，必须设置为 1。

_INCORERBF_GB16：需要中文 16 点阵字体的支持，则定义此宏为 1；

_INCORERBF_LATIN1_16：需要英文 16 点阵字体支持(英文 12 点阵字体的支持是默认的，不需要配置)，则定义此宏为 1；

__NOLINUX__：非 linux 操作系统必须定义此宏为 1；

__UCOSII__：MiniGUI for uC/OS-II 必须定义此宏为 1。

编译操作

使用 ADS 1.2 集成开发环境打开 MiniGUI_Lib 目录下的 MiniGUI_Lib 工程，然后使用鼠标单击 MiniGUI_Lib 工程窗口的 Make 图标按钮，如图 9.3 所示，开始编译连接 MiniGUI 的源文件。

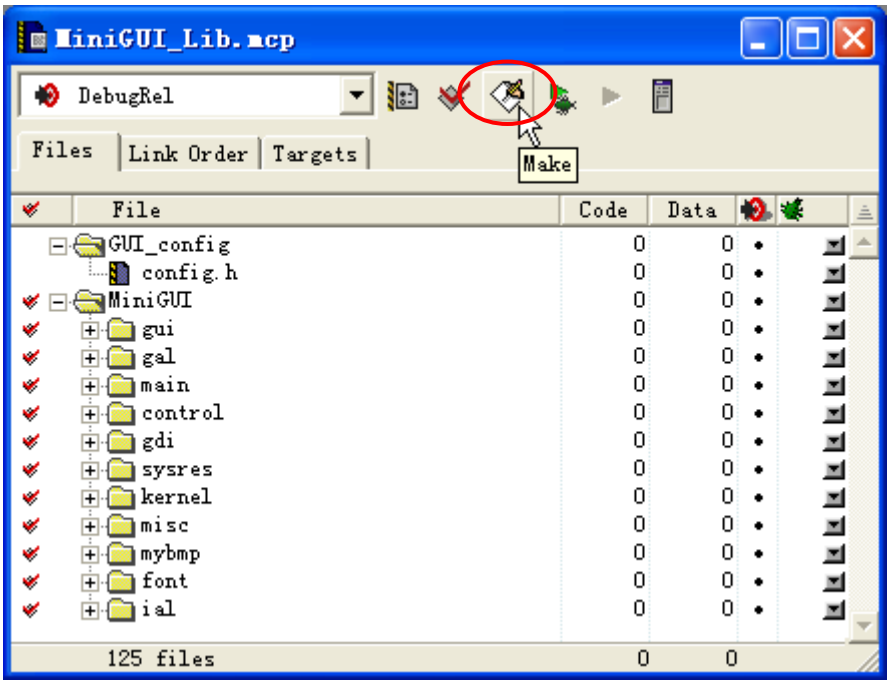


图 9.3　编译 MiniGUI_Lib 工程

编译通过后，即可生成 MiniGUI_Lib.a 库文件(在 MiniGUI_Lib\MiniGUI_Lib_Data\Debug Rel 目录下)。

注意：在 MiniGUI_Lib\ucos_include\os_cfg.h 文件中的一些宏定义要与应用工程(比如后面介绍的 GUI_Sample 工程)中的定义一致，因为 MiniGUI for uC/OS-II 会使用到其中的部分宏，如 OS_TICKS_PER_SEC。

libextc.a 库文件说明

在 Libc 目录下的 libextc.a 库文件，是针对于 uC/OS-II 操作系统的 POSIX 线程库，同时该库文件还提供了 malloc/free 等堆内存管理接口和 printf/sprintf 等 I/O 接口。

libextc.a 库文件是按照 SmartARM2200 教学实验开发平台进行移植的，它所使用的资源如下所示：

- 动态内存的大小为 3M 字节；
- 1 个 uC/OS-II 的互斥型信号量，优先级定义为 42；
- 每增加一个 MiniGUI 线程，需要的信号量至少增加 3 个；
- 最大 POSIX 线程(即 MiniGUI 线程)数为 16 个，任务(即 MiniGUI 线程)优先级范围是 43～58；
- 每 1 个 MiniGUI 线程堆栈大小为 8K 字节。

6. 实验步骤

(1) 按照实验原理说明，在 PC 机上编译 MiniGUI_Lib 工程(即 MiniGUI-STR 所有源程序)，生成 MiniGUI_Lib.a 库文件。

(2) 在本地硬盘上建立一个目录 uCOS-II，将 LPC2200 的 uC/OS-II 移植代码(即 arm 和 Arm_Pc 两个目录)复制到 uCOS-II 目录下。

其中，LPC2200 的 uC/OS-II 移植代码(即 arm 和 Arm_Pc 两个目录)可以在产品光盘上找到。

(3) 在 uCOS-II 目录下再建立一个 SOURCE 目录，将 uC/OS-II v2.52 的 11 个内核源文件复制到 SOURCE 目录下。

(4) 在 uCOS-II 下建立一个目录 MiniGUI，将 MiniGUI_Lib.a 和 libextc.a(针对于 uC/OS-II 操作系统的 POSIX 线程库)复制到 MiniGUI 目录下，并且将 MiniGUI_Lib\libminig ui-str-1.6.2-ucosii\include 目录整个目录复制到 MiniGUI 目录下。将 MiniGUI_Lib\lib minigui-str-1.6.2-ucosii\config.h 文件复制到 MiniGUI 目录下，并改名为 MiniGUI_conf ig.h。至此，MiniGUI 目录的内容如图 9.4 所示。

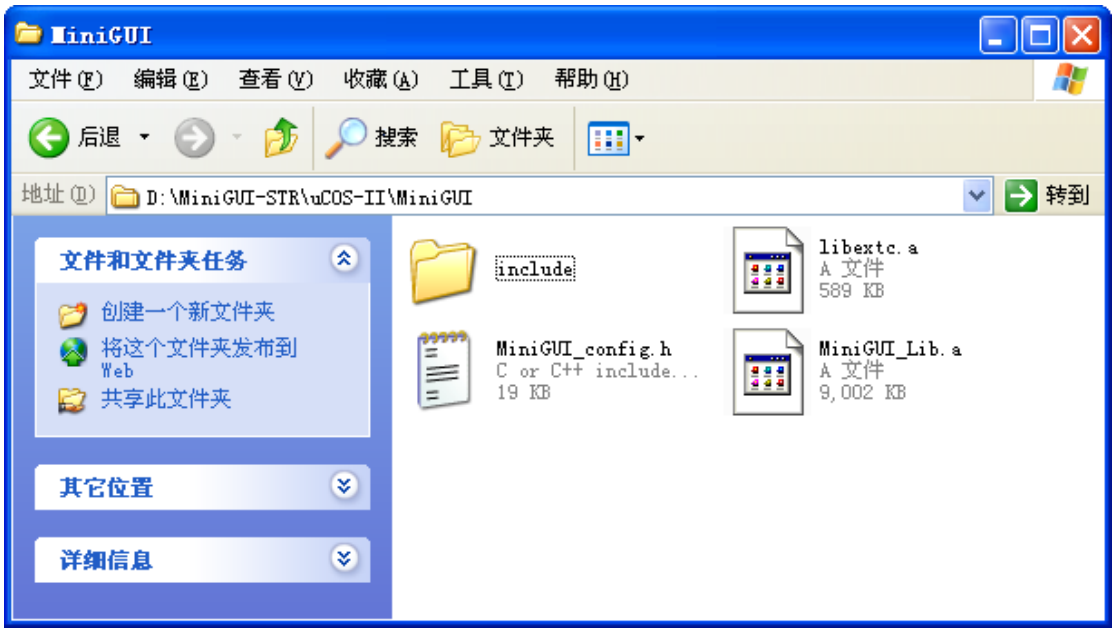


图 9.4 应用程序的 MiniGUI 目录内容

(5) 打开 ADS 1.2 集成开发环境，使用 LPC2200 专用工程模板“ARM Executable Image for UCOSII(for lpc2200)”来建立一个工程，工程名为 GUI_Sample，工程建立在 uCOS-II 目录下。

(6) 在工程管理窗口中建立一个 MiniGUI 组，把 uCOS-II\MiniGUI 目录下的两个库文件添加到 MiniGUI 组中。在工程目录的 src 目录下建立一个 messagebox 子目录，并在 messagebox 目录下新建 MiniGUI 应用程序 messageb.c 并编写实验代码，再把此文件加入到 GUI_Sample 工程的 usr 组中。

(7) 在 usr 组中的 main.c 文件添加用于启动 uC/OS-II 操作系统的代码，如程序清单 9.4 所列。首先要调用 minigui_app_entry 函数初始化动态内存管理模块和线程模块，并创建一个 MiniGUI 的主线程(由 minigui_app_entry 函数实现)。然后建立 uC/OS-II 的一个任务 Task0，任务的主要作用是调用 TargetInit 函数启动系统时钟，并负责读取键盘输入然后给 MiniGUI 发送按键消息。

程序清单 9.4 main.c 文件源程序

```
#include   "config.h"
#include   "stdlib.h"
extern void minigui_app_entry (void);

/* 按键消息邮箱 */
OS_EVENT   *KeyMbox = NULL;            // 按键消息邮箱

#define     TaskStkLengh    64              // Define the Task0 stack length 定义用户任务 0 的堆栈长度
OS_STK    TaskStk [TaskStkLengh];          // Define the Task0 stack 定义用户任务 0 的堆栈
void  Task0(void *pdata);                  // Task0 任务 0

int   main(void)
{
      OSInit ();
      OSTaskCreate (Task0,(void *)0, &TaskStk[TaskStkLengh - 1], 10);
      minigui_app_entry();
      KeyMbox = OSMboxCreate(NULL);    // 建立一个邮箱，用于传递按键消息
      OSStart();
      return(0);
}

/*********************************************************************************************
**                                    Task0 任务 0
** 键盘驱动任务。读取 ZLG7290 扫描的按键值，然后把键值发送到 KeyMbox 邮箱。
** 由 MiniGUI 的"comm" IAL 输入引擎接收 KeyMbox 邮箱的消息，然后进行相应的处理。
*********************************************************************************************/
void Task0 (void *pdata)
{    static uint8   s_key_sta = 0;
     static uint8   s_key_in = 0;
     uint16   key;
```

```
    pdata = pdata;
    TargetInit();

    while (1)
    {   key = ZLG7290GetKey();
        if(key&0x00FF)              // 如果有按键，则判断是否要发送按下消息
        {   if(s_key_sta==0)        // 按键单击
            {   /* 设置消息中的按键值。d7 位表示按键状态，按下为 1，
                   放开为 0；d3--d0 为相应的按键值。 */
                s_key_in = (key-1)&0x000F;   // 原按键值为 1-16，所以要减 1 变为 0-15
                s_key_in |= 0x80;
                OSMboxPost(KeyMbox, (void *)&s_key_in);

                s_key_sta = 1;
            }
            else                    // 按键连击
            {   /* 倘若是(摸拟)鼠标移动按键连击，则需要发送按键按下消息 */
                switch(key&0xFF)
                {   case 9:
                    case 10:
                    case 11:
                    case 14:
                        s_key_in = (key-1)&0x000F;
                        s_key_in |= 0x80;
                        OSMboxPost(KeyMbox, (void *)&s_key_in);
                        break;
                    default:
                        break;
                }
            }
        }
        else  // 如果是没有按键，则判断是否要发送放开消息
        {   if(s_key_sta != 0)      // 按键放开，则发送按键放开消息
            {   s_key_in &= 0x0F;
                OSMboxPost(KeyMbox, (void *)&s_key_in);
            }
            s_key_sta = 0;
        }
         OSTimeDly(5);
    }
}
```

(8) 将 I^2C 总线驱动中间件的 I2C.c 和 I2C.h 文件和 ZLG7290 软件包的 ZLG7290.c

和 ZLG7290.h 文件复制到 GUI_Sample\src 目录下。然后在工程中建立一个 I2C 组，并且 I2C.c、ZLG7290.c 把添加到 I2C 组，添加文件后的 GUI_Sample 工程的结构如图 9.5 所示。

其中，I^2C 总线驱动中间件(I2C.c 和 I2C.h 文件)以及 ZLG7290 软件包(ZLG7290.c 和 ZLG7290.h 文件)可以在产品光盘上找到。

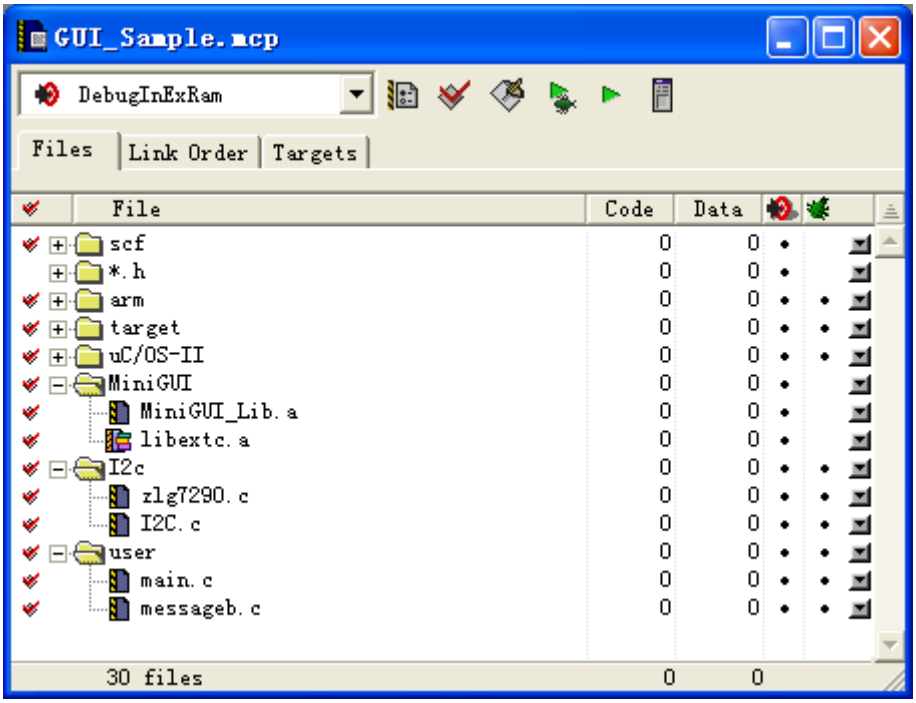


图 9.5　GUI_Sample 工程结构

(9) 在项目中的 config.h 文件中加入“#include　"I2C.h""和“#include "ZLG7290.h""语句，在工程的 target 组的 irq.s 文件最后，增加 I^2C 中断服务程序的汇编语言部分代码“I2c_Handler　HANDLER　I2c_Exception”。

(10) 在工程的 target 组的 target.c 文件中的 VICInit 函数，添加 I^2C 向量中断的初始化代码，如程序清单 9.5 所示。由于在 I2C.c 文件的 I2cInit()函数中会设置 VIC 使能 I^2C 中断，所以程序清单 9.5 中的中断使能设置的代码可以删除。

程序清单 9.5　I^2C 向量中断初始化

```
extern   void   I2c_Handler(void);
VICVectAddr12 = (uint32)I2c_Handler;
VICVectCntl12 = (0x20 | 9);
//VICIntEnable = 1 << 9;
```

(11) 在工程的 target 组的 target.c 文件中的 TargetInit 函数，添加初始化 I^2C(主模式)的代码“I2cInit(30000);”以及设置 P0.22 为 I/O 输出模式，代码如程序清单 9.6 所列。

说明：由于 MiniGUI(for uC/OS-II)库里面的“commlcd”图形引擎、“comm”输入引擎的底层驱动程序并没有初始化自己所用到的 I/O，所以在应用程序中要正确初始化硬件。

程序清单 9.6　GUI_Sample 工程的硬件初始化

```
……
I2cInit(30000);
/* 设置 P0.22 为输出 (HD66781_RST 连接到 P0.22) */
PINSEL1 = PINSEL1 & 0xFFFFCFFF;
IO0DIR = (1<<22);
OS_EXIT_CRITICAL();
```

(12) 修改工程中的 Startup.s 文件的 ResetInit 程序段，初始化与液晶模块驱动相关的总线配置(Bank3 存储块的设置)，如程序清单 9.7 所列。

程序清单 9.7 GUI_Sample 工程的总线配置

```
……
; 定义总线速度控制字
BCFG_DEF      EQU       0x10000400
IDCY          EQU       (0x00<<0)
WST1          EQU       (0x01<<5)
WST2          EQU       (0x01<<11)
BCFG3_SET     EQU       (BCFG_DEF | IDCY | WST1 | WST2)
IDCYF         EQU       (0x01<<0)
WST1F         EQU       (0x03<<5)
WST2F         EQU       (0x03<<11)
BCFG_PSRAM    EQU       (BCFG_DEF | IDCYF | WST1F | WST2F)

        LDR       R0, =BCFG0
        LDR       R1, =0x1000ffef
        STR       R1, [R0]

        LDR       R0, =BCFG1
        LDR       R1, =BCFG_PSRAM
        STR       R1, [R0]

        LDR       R0, =BCFG2
        LDR       R1, =0x1000ffef
        STR       R1, [R0]

        LDR       R0, =BCFG3
        LDR       R1, =BCFG3_SET
        STR       R1, [R0]
……
```

(13) 由于 MiniGUI 程序最终是在片外 RAM 上运行(片外 RAM 地址为 0x81000000 开始)，所以需要将向量表复制片内 RAM 的 0x40000000 地址上，然后进行 REMAP 操作。修改工程 target 组的 target.c 文件中的 TargetResetInit 函数，删除原来的 REMAP 部分代码，编写新的 REMAP 操作代码，如程序清单 9.8 所

列。

程序清单 9.8　REMAP 操作代码

```
extern    void Reset(void);
volatile uint32    *RAM_Vectors;
volatile uint32    *CODE_Vectors;

void TargetResetInit(void)
{       int    i;

    RAM_Vectors = (void*)0x40000000;
    CODE_Vectors = (void*)Reset;
    for(i=0; i<16; i++)
    {    *RAM_Vectors = *CODE_Vectors;
         RAM_Vectors++;
         CODE_Vectors++;
    }
    MEMMAP = 0x2;
    ……
```

(14) 选用 DebugInExram 生成目标，修改 mem_b.scf 分散加载文件，如程序清单 9.9 所列。然后打开菜单 Edit→DebugInExRam Settings…，在“ARM Linker”项的 Options 页中设置程序入口地址为 0x81000000。

程序清单 9.9　mem_b.scf 文件内容

```
ROM_LOAD 0x81000000
{
    ROM_EXEC 0x81000000
    {
        Startup.o (vectors, +First)
        * (+RO)
    }

    IRAM 0x40000100
    {
        Startup.o (MyStacks)
    }

    STACKS_BOTTOM +0 UNINIT
    {
        Startup.o (StackBottom)
    }
    STACKS 0x40004000 UNINIT
    {
```

```
        Startup.o (Stacks)
    }

    ERAM 0x81300000
    {
        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        Startup.o (Heap)
    }
    HEAP_BOTTOM 0x81800000 UNINIT
    {
        Startup.o (HeapTop)
    }
}
```

(15) 配置 uC/OS-II 操作系统。配置工程的 os_cfg.h 文件，将宏 OS_MAX_TASKS 定义为 60，将 OS_MAX_EVENTS 的值定义为 32 或更大。

(16) 删除半主调试代码。在 GUI_Sample 工程的 Startup.s 文件中，删除“IMPORT __use_no_semihosting_swi ”代码；在 target.c 文件中，删除“#pragma import(__use_no_semihosting_swi)”代码，删除__rt_div0、fputc、fgetc、_sys_close、_sys_write、_sys_read、_ttywrch、_sys_istty、_sys_seek、_sys_ensure、_sys_flen、_sys_tmpnam、_sys_exit 和_sys_command_string 函数的定义。

(17) 至此，MiniGUI-STR for uC/OS-II 的应用工程已经设置完成，使用鼠标单击 GUI_Sample 工程窗口的 Make 图标按钮，编译连接 GUI_Sample 工程。

(18) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(19) 启动 AXD 调试第一个 MiniGUI 程序。如果一切正确，液晶屏幕上将会显示一个消息框，标题为“Hello”，消息内容为“Hello MiniGUI!”，消息框只有一个“确定”按钮，如图 9.6 所示。



图 9.6 messageb.c 程序运行效果

(20) 使用开发板上的模拟鼠标功能按键移动光标到对话框的“确定”按钮上，然后按下开发板上的功能键 LB，即单击 MiniGUI 消息框中的“确定”按钮，则消息框将被关闭，程序运行结束此时液晶屏上没有任何东西显示。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。

(21) 使用 EasyJTAG 下载 MiniGUI 应用程序到目标板需要的时间较长，不利于程序的调试，所以建议用户先在 PC 机上使用 MiniGUI 的 Win32 平台开发包调试程序，或在 RedHat Linux 9.0 操作系统上使用 MiniGUI-STR for Linux 软件包来调试程序，最后再移值到 MiniGUI-STR for uC/OS-II 上。移植时通常只需要修改的两个地方，即头文件包含部分代码和 dti.c 文件包含部分代码，分别如下：

Linux 版程序

```
/* 包含 MiniGUI 头文件 */
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
……
#ifndef _LITE_VERSION
    #include <minigui/dti.c>
#endif
```

uC/OS-II 版程序

```
#include "MiniGUI_config.h"
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"
……
#ifndef _LITE_VERSION
    #include "dti.c"
#endif
```

说明：MiniGUI 的 Win32 平台开发包在产品光盘上可以找到。

(22) 另外，用户也可以使用 SmartARM2200 教学实验开发平台上的 ZLG/BOOT 将 MiniGUI 应用程序下载到开发板上的 NAND Flash 中，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

首先，在 GUI_Sample 工程中打开菜单 Edit→DebugInExRam Settings…，在“Target Settings”项的“Post-linker”中选择“ARM formELF”。

然后，在“ARM formELF”项中设置项目输出 bin 文件，文件名为 minigui.bin，相关设置如图 9.7 所示。

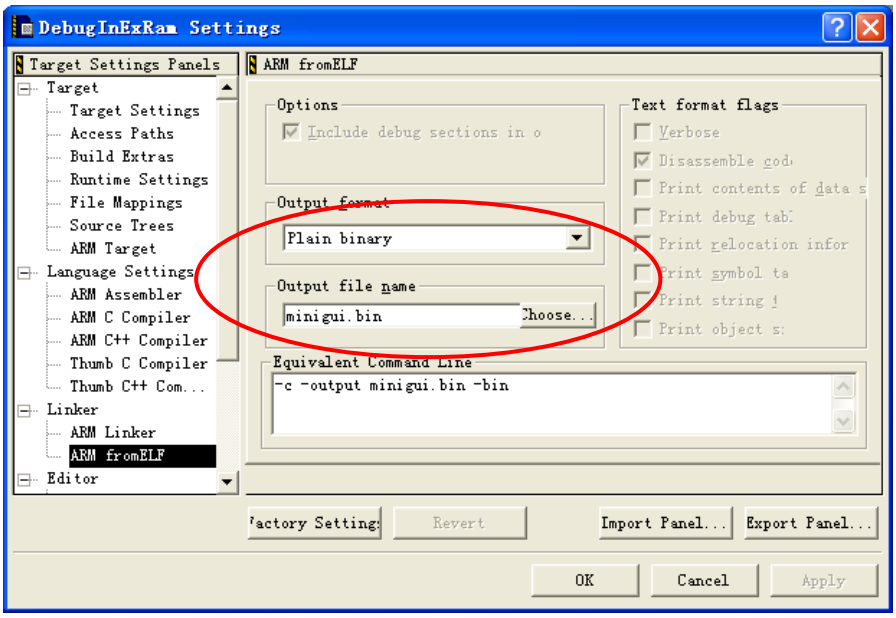


图 9.7 设置输出 minigui.bin 文件

接着，重新编译链接 GUI_Sample 工程，在 GUI_Sample\GUI_Sample_Data\DebugInExRam 目录下会生成 minigui.bin 文件。参考产品光盘上的文档《SmartARM2200 出厂程序的烧写.pdf》，将 minigui.bin 文件下载到开发板上的 boot 目录下。新建一个 ZLG_BOOT.INI 文本文件，编写文件内容如程序清单 9.10 所列，然后将 ZLG_BOOT.INI 也下载到开发板上的 boot 目录下。

程序清单 9.10 启动 MiniGUI 程序配置

```
#run MiniGUI program for SmartARM2200
load 0x81000000 a:\boot\minigui.bin
set PC=0x81000000
run
```

重新复位开发板，使用超级终端选择“run MiniGUI program for SmartARM2200”菜单项，ZLG/BOOT 将会加载 minigui.bin 文件到片外 RAM 运行。

说明：至此，MiniGUI 编译和开发板上的运行环境已经搭建完成，如果用户需要编写其它功能的 MiniGUI 应用程序，可以在 GUI_Sample 工程的 src 目录下建立一个目录，然后在此目录下新建一个源文件。新建的源文件参考 messageb.c 进行编写，然后将这个源文件添加到 GUI_Sample 工程中，当然原来的 messageb.c 就要从工程中删除掉。

7. 实验参考程序

MiniGUI for uC/OS-II 移植实验的参考代码见程序清单 9.11。

程序清单 9.11 MiniGUI for uC/OS-II 移植实验参考程序

```
/*************************************************************************
* 文件名：Messageb.C
* 功能：MiniGUI 应用例子。
```

```
*          使用消息框(MessageBox)来实现"Hello MiniGUI!"的显示。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*******************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/*******************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc      参数个数
*           argv      参数字符串指针
* 出口参数：返回 0。
*******************************************************************************/
int MiniGUIMain(int argc, const char *argv[])
{
#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif

    MessageBox(HWND_DESKTOP, "Hello MiniGUI!", "Hello", MB_OK | MB_ICONHAND);

    return(0);
}

#ifndef _LITE_VERSION
    #include "dti.c"
#endif
```

8． 思考

使用 MiniGUI-STR for uC/OS-II 时，通过配置哪一个文件来使用 dummy 输入引擎？

9.2 MiniGUI 消息处理实验

1． 实验目的

(1) 了解事件驱动编程的概念，了解窗口的概念和窗口过程函数的作用。
(2) 掌握 MiniGUI 消息循环(代码)的编写。

(3) 掌握 MiniGUI 消息的种类和几个重要的 MiniGUI 消息。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

使用 MiniGUI 创建一个主窗口，窗口过程函数中定义一个用于 MSG_PAINT 消息次数的计数变量，并在 MSG_PAINT 消息处理中显示出当前 MSG_PAINT 消息的计数值，比如“MSG_PAINT 消息次数: 2”。当接收到鼠标左键按下消息(MSG_LBUTTONDOWN)时，使用 UpdateWindow 函数使窗口重绘(这样又产生了 MSG_PAINT 消息)。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解 MiniGUI 窗口和消息的相关内容。

5. 实验原理

消息是 MiniGUI 程序运行的基础，应用程序通过接收消息来和外界交互。消息由系统或应用程序产生，比如创建窗口时系统会产生 NCREATE、CREATE 消息，用户在窗口中的各种操作会产生一系列相应消息。以下简单介绍一下消息的知识，更详细的说明请参考 MiniGUI 的应用资料。

1) 消息的数据结构

MiniGUI 的消息数据结构定义如程序清单 9.12 所示，其中窗口句柄、消息标识号都是一个数值，用来表示接收消息的窗口，以及此消息的类型(比如 MSG_CREATE)；两个消息参数都是 32 位的值，对于不同的消息类型它们的意义也不同。

程序清单 9.12 MiniGUI 的消息数据结构定义

```
typedef struct _MSG
{
    HWND        hwnd;       // 窗口句柄
    int         message;    // 消息标识号
    WPARAM      wParam;     // 消息参数
    LPARAM      lParam;     // 消息参数
    unsigned int time;      // 消息产生的时间
#ifndef _LITE_VERSION
    void*       pAdd;
#endif
```

```
} MSG;
```

2) 消息的类型

MiniGUI 的消息类型是以“MSG_”开头，比如 MSG_PAINT、MSG_LBUTTONDOWN，消息类型的定义参考 libminigui-str-1.6.2/include/window.h 文件。以下介绍用户程序经常用到的消息：

- MSG_CREATE 窗口建立消息。当窗口成功创建后系统会产生此消息，用户程序可以在此消息中处理一些变量的初始化、加载资源、建立子窗口等等。
- MSG_PAINT 窗口绘图消息。当窗口需要重绘时系统会产生此消息，用户程序可以在此消息中处理窗口的文字或图形显示。
- MSG_COMMAND 命令消息。当选择下拉菜单功能、按下快捷键、单击按钮时系统会产生此消息，用户程序要做相应功能的操作。
- MSG_INITDIALOG 模态对话框初始化消息。当模态对话框创建窗口和所有控件之后将会产生此消息，该消息的 lParam 参数值为 DialogBoxIndirectParam 函数的第四个参数的值，用户程序可以在此消息中处理一些变量的初始化或控件设置等，最后直接 return(1)。
- MSG_CHAR 字符消息。当有按键输入时，由消息循环的 TranslateMessage 函数产生，用户程序可以在此消息中做相应的字符输入处理。
- MSG_LBUTTONDOWN 鼠标左键按下消息。当鼠标在窗口中按下左键时将会产生此消息，用户程序可以在此消息中进行所需的操作(如选择操作对象)。
- MSG_MOUSEMOVE 鼠标移动消息。当移动鼠标时系统将会产生此消息，用户程序可以在此消息中进行所需的操作(如画图操作)。
- MSG_COLSE 关闭窗口命令消息。当用户单击窗口右上角的“关闭”按钮时将产生此消息，用户程序可以在此消息调用 DestroyMainWindow 函数销毁主窗口并调用 PostQuitMessage 函数发送 MSG_QUIT 消息(以便于结束消息循环)。
- MSG_DESTROY 窗口销毁消息。当用户程序使用 DestroyMainWindow 函数或 DestroyWindow 函数后，系统将会产生此消息，用户程序在此消息中销毁窗口中的资源，比如使用 DestroyIcon 函数或 DestroyAllControls 函数等。

3) 消息的发送

用户程序可以使用 SendMessage 函数或 PostMessage 向窗口或控件发送消息，SendMessage 函数发送一条消息到窗口，并等待该窗口完成此消息的处理之后才返回；PostMessage 函数只是窗口消息队列投寄消息，然后直接返回，不去等待窗口处理此消息。

6. 实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 paint 子目录，并在 paint 目录下新建 MiniGUI 应用程序 paint.c 并编写实验代码。

(2) 再把 paint.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(3) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(4) 启动 AXD 调试 MiniGUI 程序。如果一切正确，开发板的 TFT 液晶屏上将会显示一个窗口，窗口中显示有 MSG_PAINT 消息的产生次数。程序运行结果参考如图 9.8 所示。



图 9.8　paint.c 程序运行结果

(5) 操作开发板上的模拟鼠标按键移动光标到窗口中，然后按下功能键 LB(模拟鼠标左键)，窗口中显示的 MSG_PAINT 消息计数将会加 1。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

MiniGUI 消息处理实验的参考程序见程序清单 9.13。

程序清单 9.13　MiniGUI 消息处理实验参考程序

```
/*****************************************************************************
* 文件名：Paint.C
* 功能：MiniGUI 应用例子。
*       创建一个主窗口，对窗口的 MSG_PAINT 消息进行计数，并在窗口
*       中显示计数值。鼠标左键按下时，让窗口重绘。
* 说明：使用 MiniGUI v1.61 for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/* 主窗口起始位置及大小 */
#define   MWINDOW_LX  10        /* 窗口左边框的 x 值 */
#define   MWINDOW_TY  50        /* 窗口上边框的 y 值 */
#define   MWINDOW_RX  230       /* 窗口右边框的 x 值 */
#define   MWINDOW_BY  180       /* 窗口下边框的 y 值 */

HWND hMainWnd;          // 主窗口句柄
```

```
/*****************************************************************************
* 名称：WinProc()
* 功能：主窗口过程函数。
*       处理 MSG_PAINT 消息，先对计数变量进行加一计数，然后将计数值输
*       出到窗口中显示。
* 入口参数：hWnd          窗口句柄
*           message       消息
*           wParam        消息附加参数 1
*           lParam        消息附加参数 2
* 出口参数：消息已处理则返回 0。
*****************************************************************************/
static int   WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{    HDC   hdc;
     char   disp_buf[50];
     static int   no = 0;

     switch(message)
     {    case MSG_PAINT:
               hdc = BeginPaint(hWnd);
               no++;
               sprintf(disp_buf, "MSG_PAINT 消息次数:%d", no);
               TextOut(hdc, 10, 35, disp_buf);
               EndPaint(hWnd, hdc);
               break;

          case MSG_LBUTTONDOWN:                    // 鼠标左键按下消息
               //InvalidateRect(hWnd, NULL, TRUE);  // 重绘整个窗口 (可以用 UpdateWindow 函数)
              UpdateWindow(hWnd, TRUE);
               break;

          case MSG_CLOSE:
               DestroyMainWindow(hWnd);
               PostQuitMessage(hWnd);
               break;

          default:
               return(DefaultMainWinProc(hWnd, message, wParam, lParam));
     }
     return(0);
}


/*****************************************************************************
```

```
* 名称：InitMainWindow()
* 功能：建立主窗口。
* 入口参数：无
* 出口参数：建立成功返回 1，否则返回 0。
*****************************************************************************/
int InitMainWindow(void)
{    MAINWINCREATE    window_info;

     window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
     window_info.dwExStyle = WS_EX_NONE;
     window_info.spCaption = "MSG_PAINT 消息";
     window_info.hMenu = 0;
     window_info.hCursor = GetSystemCursor(0);
     window_info.hIcon = GetSmallSystemIcon(IDI_APPLICATION);
     window_info.MainWindowProc = WinProc;
     window_info.lx = MWINDOW_LX;
     window_info.ty = MWINDOW_TY;
     window_info.rx = MWINDOW_RX;
     window_info.by = MWINDOW_BY;
     window_info.iBkColor = COLOR_lightwhite;
     window_info.dwAddData = 0;
     window_info.hHosting = HWND_DESKTOP;
     hMainWnd = CreateMainWindow(&window_info);
     if(hMainWnd == HWND_INVALID) return(0);
       else return(1);
}


/*****************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc      参数个数
*           argv      参数字符串指针
* 出口参数：返回 0。
*****************************************************************************/
int   MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
     SetDesktopRect(0,0, 800,600);
#endif

     InitMainWindow();
     ShowWindow(hMainWnd, SW_SHOWNORMAL);
```

```
    while(GetMessage(&Msg, hMainWnd))
    {   TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }

    MainWindowThreadCleanup(hMainWnd);
    return(0);
}

#ifndef _LITE_VERSION
    #include "dti.c"
#endif
```

8. 思考

在 MiniGUI 的消息循环中，函数 GetMessage、TranslateMessage 和 DispatchMessage 的功能分别是什么？

9.3 对话框应用编程实验

1. 实验目的

(1) 掌握 MiniGUI 的模态对话框应用编程。
(2) 了解 MiniGUI 控件的基本应用方法。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

使用 MiniGUI 创建一个模态对话框，要求包含“OK”、“CANCEL”两个按钮，并且在对话框中显示“This is MiniGUI dlg!”(使用 MiniGUI 的“static”控件控件显示字符串)。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解 MiniGUI 的对话框编程方法。

5. 实验原理

在 MiniGUI 中，对话框是一类特殊的主窗口，这种窗口一般只关注与用户的交互(主要是参数输入)。对话框可分为模态对话框和非模态对话框，模态对话框显示之后，用户就不能再切换到其它主窗口进行工作，而非模态对话框就没有这样的限制。MiniGUI 使用了结构 DLGTEMPLATE 表示对话框模板，结构 CTRLDATA 表示控件模板(用于对话框的控件)。用户需要使用这两个结构来定义对话框，而且定义的控件要挂接到对话框上。

模态对话框使用 InitDialogBox 函数启动(显示)，然后在对话框的回调函数(即窗口过程函数)实现对控件消息的处理，包含结束对话框(使用 EndDialog 函数)。

以下简要的说明模态对话框的建立过程，更详细的说明请参考 MiniGUI 的应用资料。

1) 定义资源 ID

首先要定义对话框内的控件的 ID 值，ID 值应大于 100，且不要重复，如程序清单 9.14 所示。MiniGUI 在 windows.h 文件中也定义了一些标准控件的 ID，如 IDOK、IDCANCEL 和 IDABORT 等等，用户可以直接使用这些 ID。

程序清单 9.14 对话框内的控件 ID 值定义

```
#define   IDC_DISP1  1001
...
```

2) 定义对话框模板

使用 DLGTEMPLATE 来定义一个对话框对象，并设置对话框的风格、大小、标题以及控件个数等等，如程序清单 9.15 所示。程序清单 9.15 中，图标和菜单变量设置为 0，表明不使用图标和菜单；控件个数设置为 3，而控件对象的地址设置为 NULL，所以在调用 InitDialogBox 函数之前必须给它正确赋值。

程序清单 9.15 对话框定义示范

```
static   DLGTEMPLATE   DlgInitProgress =
{    WS_BORDER | WS_CAPTION,   /*  风格  */
     WS_EX_NONE,               /*  扩展风格  */
     100, 100, 200, 130,       /*  对话框起始坐标，宽度、高度  */
     "Dlg demo",               /*  对话框标题  */
     0,                        /*  图标  */
     0,                        /*  菜单  */
     3,                        /*  内含控件个数  */
     NULL,                     /*  控件对象的地址  */
     0                         /*  附加数据  */
};
```

3) 定义对话框内的控件

控件是一个子窗口，每一类控件仅用于实现某些较为固定的功能，比如静态框(“static”类控件)就用于文字显示输出，对话框常常使用控件来实现提示或设置等功能。

定义控件示范如程序清单 9.16 所示，使用了 MiniGUI 的“static” 类控件(静态框)和“button”类控件(按钮)。

程序清单 9.16 对话框内的控件定义示范

```
static CTRLDATA    CtrlInitData[] =
{
    {       "static",                           /*  控件类名  */
            WS_VISIBLE | SS_SIMPLE,             /*  控件风格  */
            50,30,    100, 16,                  /*  控件的起始坐标，宽度、高度  */
            IDC_DISP1,                          /*  控件 ID */
            "This is MiniGUI dlg!",             /*  控件标题  */
            0,                                  /*  附加参数  */
            WS_EX_NONE                          /*  扩展风格  */
    },
    {       "button",
            WS_TABSTOP | WS_VISIBLE | BS_DEFPUSHBUTTON,
            40,70,    50,25,
            IDOK,
            "OK",
            0,
            WS_EX_NONE
    },
    …
};
```

4) 编写对话框过程函数

使用对话框时，用户需要编写相应的过程函数，主要处理的消息为 MSG_INITDIALOG 和 MSG_COMMAND 消息，处理 MSG_INITDIALOG 消息时要返回 1，即“return(1);”；而 MSG_COMMAND 消息要根据命令进行处理，结束对话框使用 EndDialog 函数，如程序清单 9.17 所示。

程序清单 9.17　对话框过程函数示范

```
static int    DialogBoxProc(HWND hdlg, int message, WPARAM wParam, LPARAM lParam)
{     switch(message)
      {     case MSG_INITDIALOG:
                  return(1);
            case MSG_COMMAND:
                  switch(wParam)
                  {     case IDOK:
                        case IDCANCEL:
                              EndDialog(hdlg, wParam);
                              DestroyAllControls(hdlg);
                              break;
                  }
                  break;
      }
    return(DefaultDialogProc(hdlg, message, wParam, lParam));
}
```

5) 启动对话框

如前面所介绍，用户需要调用 InitDialogBox 函数启动对话框，调用此函数时需要一个对话框对象(使用 DLGTEMPLATE 来定义的对话框对象)，如程序清单 9.18 所示。

程序清单 9.18　启动对话框示范

```
DlgInitProgress.controls = CtrlInitData;                /* 设置控件对象地址 */
/* 启动对话框"DlgInitProgress" */
DialogBoxIndirectParam(&DlgInitProgress, hWnd, DialogBoxProc, 0L);
```

6．　实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 dlg 子目录，并在 dlg 目录下新建 MiniGUI 应用程序 dlg.c 并编写实验代码。

(2) 再把 dlg.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(3) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(4) 启动 AXD 调试 MiniGUI 程序。如果一切正确，开发板的 TFT 液晶屏上将会显示一个窗口如图 9.9 所示。

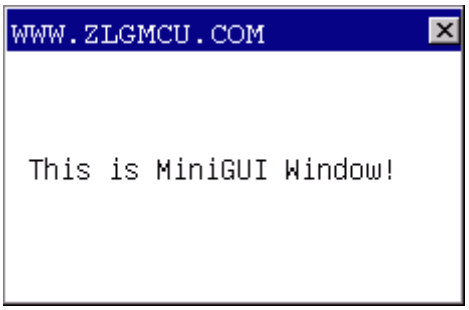


图 9.9　dlg.c 程序运行结果—主窗口

(5) 操作开发板上的模拟鼠标按键移动光标到窗口中，然后按下功能键 LB(模拟鼠标左键)，即会弹出一个模态对话框，如图 9.10 所示。



图 9.10　dlg.c 程序运行结果—对话框

(6) 移动光标到对话框的 OK”按钮或“CANCEL”按钮上，按下开发板上的功能键 LB，对话框将会关闭。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。可以使用 ZLG/BOO 将工

程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

对话框应用编程实验的参考程序见程序清单 9.19。

程序清单 9.19 对话框应用编程实验参考程序

```
/*****************************************************************************
* 文件名：Dlg.C
* 功能：MiniGUI 应用例子。
*       建立一个模式对话框，对话框包括 2 个命令按钮和 1 个静态文本框。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/* 主窗口起始位置及大小 */
#define   MWINDOW_LX        5         /* 窗口左边框的 x 值 */
#define   MWINDOW_TY        50        /* 窗口上边框的 y 值 */
#define   MWINDOW_RX        235       /* 窗口右边框的 x 值 */
#define   MWINDOW_BY        200       /* 窗口下边框的 y 值 */

/* 定义静态文本框的 ID */
#define   IDC_DISP1  1001

HWND   hMainWnd;       // 主窗口句柄

/* 对话框属性设置 */
static   DLGTEMPLATE   DlgInitProgress =
{     WS_BORDER | WS_CAPTION,  // 风格
      WS_EX_NONE,        // 扩展风格
      10, 10, 200, 130,        // 对话框起始坐标，宽度、高度
      "Dlg demo",              // 对话框标题
      0,                       // 图标
      0,                       // 菜单
      3,                       // 内含控件个数
      NULL,                    // 控件对象的地址
```

```
    0                       // 附加数据
};

/* 对话框内的控件定义 */
static CTRLDATA CtrlInitData[] =
{
    {       "static",
            WS_VISIBLE | SS_SIMPLE,
            10,30,   160, 16,
            IDC_DISP1,
            "This is MiniGUI dlg!",
            0,
            WS_EX_NONE
    },
    {       "button",
            WS_TABSTOP | WS_VISIBLE | BS_DEFPUSHBUTTON,
            30,70,   60,25,
            IDOK,
            "OK",
            0,
            WS_EX_NONE
    },
    {       "button",
            WS_TABSTOP | WS_VISIBLE,
            110,70, 60,25,
            IDCANCEL,
            "CANCEL",
            0,
            WS_EX_NONE
    }
};


/*****************************************************************************
* 名称：DialogBoxProc()
* 功能：对话框过程函数。
*       处理 MSG_COMMAND 消息，执行命令按钮的功能。
* 入口参数：hWnd          窗口句柄
*           message       消息
*           wParam        消息附加参数 1
*           lParam        消息附加参数 2
* 出口参数：返回消息处理结果。
*****************************************************************************/
static int   DialogBoxProc(HWND hdlg, int message, WPARAM wParam, LPARAM lParam)
```

```
{    switch(message)
     {    case MSG_INITDIALOG:
               return(1);

          case MSG_COMMAND:
               switch(wParam)
               {    case IDOK:
                    case IDCANCEL:
                         EndDialog(hdlg, wParam);
                         DestroyAllControls(hdlg);
                         break;
               }
               break;
     }

     return(DefaultDialogProc(hdlg, message, wParam, lParam));
}


/*****************************************************************************
* 名称：InitDialogBox()
* 功能：初始化对话框，然后启动对话框。
* 入口参数：hWnd              父窗口句柄
* 出口参数：无
*****************************************************************************/
static void    InitDialogBox(HWND hWnd)
{    DlgInitProgress.controls = CtrlInitData;       // 设置控件对象地址

     /* 启动对话框"DlgInitProgress" */
     DialogBoxIndirectParam(&DlgInitProgress, hWnd, DialogBoxProc, 0L);
}


/*****************************************************************************
* 名称：WinProc()
* 功能：主窗口过程函数。
*          当接收到 MSG_LBUTTONDOWN 消息，打开模式对话框。
* 入口参数：hWnd          窗口句柄
*               message        消息
*               wParam        消息附加参数 1
*               lParam         消息附加参数 2
* 出口参数：消息已处理则返回 0。
*****************************************************************************/
static int    WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
```

```
{    HDC    hdc;

     switch(message)
     {     case MSG_PAINT:
                 hdc = BeginPaint(hWnd);
                 TextOut(hdc, 10, 50, "This is MiniGUI Window!");
                 EndPaint(hWnd, hdc);
                  break;

           case MSG_CLOSE:
                 DestroyMainWindow(hWnd);
                 PostQuitMessage(hWnd);
                 break;

           case MSG_LBUTTONDOWN:
                InitDialogBox(hWnd);
                break;

           default:
                 return(DefaultMainWinProc(hWnd, message, wParam, lParam));
     }
     return(0);
}


/*************************************************************************
* 名称：InitMainWindow()
* 功能：建立主窗口。
* 入口参数：无
* 出口参数：建立成功返回 1，否则返回 0。
*************************************************************************/
int InitMainWindow(void)
{    MAINWINCREATE    window_info;

     window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
     window_info.dwExStyle = WS_EX_NONE;
     window_info.spCaption = "WWW.ZLGMCU.COM";
     window_info.hMenu = 0;
     window_info.hCursor = GetSystemCursor(0);
     window_info.hIcon = 0;
     window_info.MainWindowProc = WinProc;
     window_info.lx = MWINDOW_LX;
     window_info.ty = MWINDOW_TY;
     window_info.rx = MWINDOW_RX;
```

```
    window_info.by = MWINDOW_BY;
    window_info.iBkColor = COLOR_lightwhite;
    window_info.dwAddData = 0;
    window_info.hHosting = HWND_DESKTOP;
    hMainWnd = CreateMainWindow (&window_info);
    if (hMainWnd == HWND_INVALID) return(0);
      else   return(1);
}


/*************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc      参数个数
*           argv      参数字符串指针
* 出口参数：返回 0。
*************************************************************************/
int   MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif

    InitMainWindow();
    ShowWindow(hMainWnd, SW_SHOWNORMAL);
    while (GetMessage(&Msg, hMainWnd))
    {    TranslateMessage(&Msg);
         DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup (hMainWnd);
    return(0);
}

#ifndef _LITE_VERSION
     #include "dti.c"
#endif
```

8. 思考

(1) 请描述 MiniGUI 对话框模板(数据结构)各成员变量的作用。

(2) 如果要将实验参考程序的对话框标题改为 Hello，只需要一个“OK”按钮，应如何修改？

9.4 控件应用编程实验

1. 实验目的

(1) 掌握 MiniGUI 的模态对话框应用编程。
(2) 掌握 MiniGUI 控件的基本应用。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

使用 MiniGUI 的静态框、按钮和编辑框控件，实现一个 Login 系统登录对话框。对话框中要求输入用户名和密码，如果输入正确的用户名和密码(如用户名为“51”，密码为“888888”)则进入系统(显示一个主窗口)，否则弹出错误对话框，并要求用户再次输入用户名和密码。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。
(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。
(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解 MiniGUI 的模态对话框和基本控件应用等相关知识。

5. 实验原理

MiniGUI 提供了众多的系统控件，如静态框、按钮、列表框、编辑框等等，对话框中通常是利用这些控件实现各种显示输出、设置输入等功能。以下简要的介绍静态框、按钮和编辑框控件，更详细的说明请参考 MiniGUI 的应用资料。

静态框

静态框是最常用的控件之一，适用于在窗口的特定位置显示文字、数字等信息，还可以用来静态的图片信息。静态框可以使用 WS_xxx 等子窗口风格修饰，同时它也具有多种自身的风格控制选项，常用的风格如下：

- SS_SIMPLE 单行文本显示。即文本不会自动换行显示，文本永远是左对齐的。
- SS_LEFT 多行文本显示，左对齐。
- SS_CENTER 多行文本显示，居中对齐。
- SS_RIGHT 多行文本显示，右对齐。
- SS_BITMAP 位图形静态框，显示一幅位图。
- SS_ICON 位图形静态框，显示一幅图标。

使用静态框，可以在定义(对话框的)控件时指定要显示的文字，或者调用 CreateWindow

函数时指定显示的文字，如程序清单 9.20、程序清单 9.21 所示。若在程序运行过程中要更改静态框的显示文字，可以使用 SetWindowText 函数实现。

程序清单 9.20 对话框中的静态框定义

```
{    "static",                  /* 控件类名 */
     WS_VISIBLE | SS_SIMPLE,    /* 控件风格 */
     65,10,  200, 16,           /* 控件的起始坐标，宽度、高度 */
     IDC_SLOGIN,                /* 控件 ID */
     "请输入用户名和密码.",      /* 控件标题 */
     0,                         /* 附加参数 */
     WS_EX_NONE                 /* 扩展风格 */
}
```

程序清单 9.21 使用 CreateWindow 函数创建静态框

```
CreateWindow(  CTRL_STATIC,
               "请输入用户名和密码.",
               WS_CHILD | WS_VISIBLE | SS_SIMPLE,
               IDC_SLOGIN,
               65,10,  200, 16,
               hWnd,
               0  );
```

控钮

键钮也是常用的控件之一，用来为用户提供开关选择。MiniGUI 的按钮可划分为普通按钮、复选框和单选钮。

普通按钮是一个矩形，按钮中显示的文字一般在定义(对话框的)控件时指定，或者调用 CreateWindow 函数时指定，对话框中最常见的“确定/OK”、“取消/CANCEL”按钮都是普通按钮。

普通按钮可以使用 WS_xxx 等子窗口风格修饰，如 WS_VISIBLE、WS_TABSTOP。同时它也具有自身的风格控制选项，常用以下风格：

- BS_PUSHBUTTON 普通按钮。
- BS_DEFPUSHBUTTON 默认普通按钮。

编辑框

编辑框可用来显示文本并接收用户输入文本，但一般只用于接收输入文本。MiniGUI 中提供了三种类型的编辑框，简单编辑框(类名为“edit”)、单行编辑框(类名为“sledit”)和多行编辑框(类名为“mledit”)。

编辑框可以使用 WS_xxx 等子窗口风格修饰，同时它也具有自身的风格控制选项，常用以下风格：

- ES_UPPERCASE 编辑框只显示大写字母。
- ES_LOWERCASE 编辑框只显示小写字母。
- ES_PASSWORD 编辑框用于密码输入，即用“*”显示输入字符
- ES_AUTOWRAP 自动换行(用于多行编辑框)。

可以通过向编辑框发送 MSG_GETTEXTLENGTH 消息来获取文本的长度，发送 MSG_GETTEXT 消息来获取编辑框的文本，发送 MSG_SETTEXT 设置编辑框中的文本内容。

当然，也可以直接使用 GetWindowTextLength、GetWindowText 和 SetWindowText 三个函数来实现对应的功能。

6. 实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 login 子目录，并在 login 目录下新建 MiniGUI 应用程序 login.c 并编写实验代码。

(2) 再把 login.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(3) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(4) 启动 AXD 调试 MiniGUI 程序。正确运行 login 程序后在开发板的 TFT 液晶屏上应显示“登录”对话框。程序运行结果参考如图 9.11 所示。

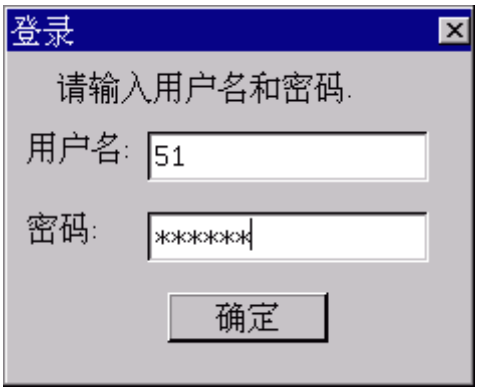


图 9.11　“登录”对话框

(5) 使用开发板上的功能键 0～9 输入用户名和密码，使用模拟鼠标功能键移动光标，选择输入焦点和确定输入(单击“确定”按钮)。当用户名为“51”，密码为“888888”时，登录成功，“登录”对话框被关闭，并显示出一个 MiniGUI 主窗口。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

控件应用编程实验的参考程序见程序清单 9.22。

程序清单 9.22　控件应用编程实验参考程序

```
/*****************************************************************************
* 文件名：Login.C
* 功能：MiniGUI 应用例子。
*       使用 MiniGUI 的静态框、按钮和编辑框控件，实现一个 Login 系统登录对话框。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"
```

```
/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

#include <string.h>

HWND hMainWnd;

#define IDC_SLOGIN 300
#define IDC_SUSER301
#define IDC_SPASS 302
#define IDC_EUSER 401
#define IDC_EPASS 402

// 定义对话框
static DLGTEMPLATE MyDlg =
{ WS_BORDER | WS_CAPTION,
  WS_EX_NONE,
  2, 50, 235, 190,
  "登录",
  0,
  0,
  6,
  NULL,
  0
};

// 定义对话框中的控件
static CTRLDATA CtrlInitData[] =
{
  { "static",
    WS_VISIBLE | SS_SIMPLE,
    25,10, 200, 16,
    IDC_SLOGIN,
    "请输入用户名和密码.",
    0,
    WS_EX_NONE
  },
  { "static",
    WS_VISIBLE | SS_SIMPLE,
```

```
            10,40,    60, 16,
            IDC_SUSER,
            "用户名:",
            0,
            WS_EX_NONE
    },
    {       "static",
            WS_VISIBLE | SS_SIMPLE,
            10,80,    60, 16,
            IDC_SPASS,
            "密码:",
            0,
            WS_EX_NONE
    },
    {       "edit",
            WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,
            70,40,    140,25,
            IDC_EUSER,
            "",
            0,
            WS_EX_NONE
    },
    {       "edit",
            WS_CHILD | WS_VISIBLE | WS_BORDER | ES_PASSWORD | WS_TABSTOP,
            70,80, 140,25,
            IDC_EPASS,
            "",
            0,
            WS_EX_NONE
    },
    {       "button",
            WS_VISIBLE | WS_TABSTOP | BS_DEFPUSHBUTTON,
            80,120,    80,25,
            IDOK,
            "确定",
            0,
            WS_EX_NONE
    }
};

#define    USER_NO   3
static char *g_user[USER_NO] = {"root",
                                "51",
                                "WXM"
```

```
                                        };
static char *g_pass[USER_NO] = {"******",
                                        "888888",
                                        "2046"
                                        };

static BOOL CheckUser(char *user, char *pass)
{   int   i;

    if(USER_NO==0) return(TRUE);    // 不启用密码
    for(i=0; i<USER_NO; i++)
    {       if(strcmp(user, g_user[i]) == 0)
            {   if(strcmp(pass, g_pass[i]) == 0) return(TRUE);
                else   return(FALSE);
            }
    }
    return(FALSE);
}


static int   MyDlgProc(HWND hDlg, int message, WPARAM wParam, LPARAM lParam)
{    char   user[30];
     char   pass[30];

     switch(message)
     {      case MSG_INITDIALOG:
                return(1);

            case MSG_COMMAND:
                switch(LOWORD(wParam))
                {    case IDC_EUSER:                    // 处理控件的通知消息(EN_ENTER)
                     case IDC_EPASS:
                          if(HIWORD(wParam) != EN_ENTER) break;
                     case IDOK:
                          // 读取编辑框的输入
                          GetWindowText(GetDlgItem(hDlg, IDC_EUSER), user, 22);
                          GetWindowText(GetDlgItem(hDlg, IDC_EPASS), pass, 22);
                          if(CheckUser(user, pass))
                          {     EndDialog(hDlg, wParam);
                                DestroyAllControls(hDlg);
                          }
                          else
                          {     MessageBox(hDlg, "密码错误!", "校验错误", MB_OK | MB_ICONHAND);
                                SetWindowText(GetDlgItem(hDlg, IDC_EUSER), "");
```

```
                                SetWindowText(GetDlgItem(hDlg, IDC_EPASS), "");
                            }
                            break;
                        default:
                            break;
                    }
                    break;

            default:
                    break;
        }
        return(DefaultDialogProc(hDlg, message, wParam, lParam));
}


static void    LoginBox(HWND hWnd)
{     MyDlg.controls = CtrlInitData;
      DialogBoxIndirectParam(&MyDlg, hWnd, MyDlgProc, 0L);
}


static char *hello_str = "欢迎登录系统!";
static int    WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{     HDC    hdc;

        switch(message)
        {       case MSG_PAINT:
                    hdc = BeginPaint(hWnd);
                    TextOut(hdc, 50, 50, hello_str);
                    EndPaint(hWnd, hdc);
                    break;
                case MSG_CLOSE:
                    DestroyMainWindow(hWnd);
                    PostQuitMessage(hWnd);
                    break;
                default:
                    return(DefaultMainWinProc(hWnd, message, wParam, lParam));
        }
        return(0);
}


int InitMainWindow(void)
{     MAINWINCREATE    window_info;
```

```
    window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
    window_info.dwExStyle = WS_EX_NONE;
    window_info.spCaption = "MiniGUI";
    window_info.hMenu = 0;
    window_info.hCursor = GetSystemCursor(0);
    window_info.hIcon = 0;
    window_info.MainWindowProc = WinProc;
    window_info.lx = 2;
    window_info.ty = 50;
    window_info.rx = 238;
    window_info.by = 200;
    window_info.iBkColor = COLOR_lightwhite;
    window_info.dwAddData = 0;
    window_info.hHosting = HWND_DESKTOP;
    hMainWnd = CreateMainWindow (&window_info);
    if (hMainWnd == HWND_INVALID) return(0);
      else    return(1);
}


int    MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif

    LoginBox(HWND_DESKTOP);
    InitMainWindow();
    ShowWindow(hMainWnd, SW_SHOWNORMAL);
    while (GetMessage(&Msg, hMainWnd))
    {    TranslateMessage(&Msg);
         DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup (hMainWnd);
    return(0);
}

#ifndef _LITE_VERSION
     #include "dti.c"
#endif
```

8. 思考

如果要求程序具有输入密码次数限制功能(比如只允许用户输入 3 次)，应如何修改实验参考程序？

9.5 自定义控件实验

1. 实验目的

掌握 MiniGUI 自定义控件的基本设计方法。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

设计一个简单的静态文本框控件，要求在控件上直接显示“my control”字符串，然后在 MiniGUI 的窗口中使用此控件。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解自定义控件的相关内容。

5. 实验原理

控件是实现某些较为固定功能的子窗口，使用控件可以减轻主程序的设计复杂性。MiniGUI 提供了众多的系统控件，如静态框、按钮、列表框、编辑框等等，用户也可以定义自己的控件(称为自定义控件)，然后在主窗口或对话框使用它。以下简要的说明自定义控件的编程步骤，更详细的说明请参考 MiniGUI 的应用资料。

1) 定义控件类型

使用 WNDCLASS 数据结构类型来定义一个控件类型变量，然后设置控件的相关设置参数，如类型名、窗口风格、光标、窗口底色等，如程序清单 9.23 所示。

程序清单 9.23 定义控件类型示例

```
WNDCLASS    MyClass;

MyClass.spClassName = MYC_STATIC;              // 类型名，字符串表达式
MyClass.dwStyle       = WS_NONE;               // 窗口风格
MyClass.dwExStyle     = WS_EX_NONE;            // 窗口扩展风格
```

```
    MyClass.hCursor       = GetSystemCursor (IDC_BUSY);// 窗口光标
    MyClass.iBkColor      = COLOR_lightwhite;          // 窗口底色
    MyClass.WinProc       = MyControlProc;             // 窗口过程函数
```

2) 注册控件类型

使用 RegisterWindowClass 函数注册用户的自定义控件类型，如程序清单 9.24 所示。

程序清单 9.24 注册控件类型示例

```
    ……
    RegisterWindowClass (&MyClass);
```

3) 编写控件的窗口过程函数

控件的窗口过程函数与主窗口过程函数的结构相似，但对于用户不处理的消息，要调用 DefaultControlProc 函数进行处理，如程序清单 9.25 所示。

用户还要根据控件的功能来提供相应的控件消息处理，以便于父窗口程序对其进行控制操作。控件消息可以是用户自己定义的，控件消息的值一般要大于 700，如程序清单 9.26 所示。

程序清单 9.25 控件的窗口过程函数示例

```
static int   MyControlProc(HWND hwnd, int message, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;

    switch (message)
    {    case MSG_PAINT:
            hdc = BeginPaint (hwnd);
            TextOut (hdc, 0, 0, "my control");
            EndPaint (hwnd, hdc);
            Return(0);
    }
    return(DefaultControlProc (hwnd, message, wParam, lParam));
}
```

程序清单 9.26 控件消息值定义示例

```
#define   RTC_SETTIME       800
#define   RTC_GETTIME       801
#define   RTC_SETHOUR       802
#define   RTC_SETMIN        803
#define   RTC_SETSEC        804
```

4) 使用控件

当控件注册后，可以在对话框的控件定义中使用，也可以使用 CreateWindow 函数直接创建并显示控件，如程序清单 9.27 所示。

程序清单 9.27 使用 CreateWindow 创建控件

```
……
CreateWindow( MYC_STATIC,                    // 窗口类型
              "",                            // 窗口标题
              WS_VISIBLE | WS_BORDER,        // 风格
              IDC_STATIC1,                   // ID
              100, 100, 100, 20,             // 窗口左上角坐标以及宽度、高度
              hWnd,                          // 父窗口句柄
              0);                            // 窗口附加参数
```

5) 注销控件

控件的销毁可以通过 DestroyWindow 函数实现，也可以通过 DestroyAllControls 函数销毁所有控件。

6. 实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 control 子目录，并在 control 目录下新建 MiniGUI 应用程序 control.c 并编写实验代码。

(2) 再把 control.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(6) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(7) 启动 AXD 调试 MiniGUI 程序。如果一切正确，在开发板的 TFT 液晶屏上可以观察到窗口中间的自定义静态文本框。程序运行结果参考如图 9.12 所示。

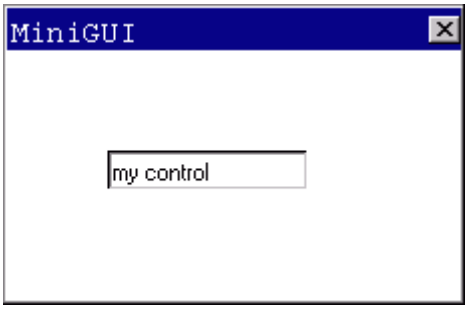


图 9.12　control.c 程序运行结果

说明：可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

自定义控件实验的参考程序见程序清单 9.28。

程序清单 9.28　自定义控件实验参考程序

```
/*****************************************************************************
* 文件名：Control.C
* 功能：MiniGUI 应用例子。
*       设计一个简单的静态文本框(自定义)控件，然后在 MiniGUI 的窗口中使用此控件。
```

```
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*********************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

#define MYC_STATIC        "MYSTATIC1"
#define IDC_STATIC11000

HWND hMainWnd;

static int MyControlProc(HWND hwnd, int message, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;

    switch (message)
    {   case MSG_PAINT:
            hdc = BeginPaint (hwnd);
            TextOut (hdc, 0, 0, "my control");
            EndPaint (hwnd, hdc);
            return(0);
    }
    return(DefaultControlProc (hwnd, message, wParam, lParam));
}


static BOOL RegisterMyControl(void)
{
    WNDCLASS MyClass;

    MyClass.spClassName = MYC_STATIC;
    MyClass.dwStyle         = WS_NONE;
    MyClass.dwExStyle     = WS_EX_NONE;
    MyClass.hCursor         = GetSystemCursor (IDC_BUSY);
    MyClass.iBkColor        = COLOR_lightwhite;
    MyClass.WinProc         = MyControlProc;
    return(RegisterWindowClass (&MyClass));
}
```

```
static void UnregisterMyControl(void)
{
    UnregisterWindowClass (MYC_STATIC);
}

static int    WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {    case MSG_CREATE:
            CreateWindow(MYC_STATIC, "", WS_VISIBLE | WS_BORDER, IDC_STATIC1,
                                50, 50, 100, 20, hWnd, 0);
            break;

        case MSG_DESTROY:
            DestroyAllControls(hWnd);
            break;

        case MSG_CLOSE:
            DestroyMainWindow(hWnd);
            PostQuitMessage(hWnd);
            break;

        default:
            return(DefaultMainWinProc(hWnd, message, wParam, lParam));
    }
    return(0);
}


int InitMainWindow(void)
{    MAINWINCREATE    window_info;

    window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
    window_info.dwExStyle = WS_EX_NONE;
    window_info.spCaption = "MiniGUI";
    window_info.hMenu = 0;
    window_info.hCursor = GetSystemCursor(0);
    window_info.hIcon = 0;
    window_info.MainWindowProc = WinProc;
    window_info.lx = 5;
    window_info.ty = 50;
    window_info.rx = 235;
    window_info.by = 200;
```

```
    window_info.iBkColor = COLOR_lightwhite;
    window_info.dwAddData = 0;
    window_info.hHosting = HWND_DESKTOP;
    hMainWnd = CreateMainWindow (&window_info);
    if (hMainWnd == HWND_INVALID) return(0);
      else   return(1);
}


int   MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif

    RegisterMyControl();
    InitMainWindow();
    ShowWindow(hMainWnd, SW_SHOWNORMAL);

    while (GetMessage(&Msg, hMainWnd))
    {    TranslateMessage(&Msg);
         DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup (hMainWnd);
    UnregisterMyControl();
    return(0);
}

#ifndef _LITE_VERSION
     #include "dti.c"
#endif
```

8． 思考

请完善参考实验程序中的静态文本框控件的功能，使文本框中显示的文字为窗口(控件窗口)的标题，并实现更新文本框内容的消息 STM_SETIMAGE。

9.6 简易编辑器实验

1． 实验目的

掌握 MiniGUI 字符消息的相关知识和基本应用。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

使用 MiniGUI 创建主窗口，在窗口过程函数中处理字符消息，实现一个简易编辑器的功能。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解键盘输入和字符消息的相关内容。

5. 实验原理

键盘是基本的输入设备之一，MiniGUI 程序通过处理窗口的键盘事件来接收按键输入，键盘事件有 MSG_KEYDOWN(键按下)、MSG_KEYUP(键释放)、MSG_SYSKEYDOWN(系统击键消息，即 ALT 键)、MSG_SYSKEYUP(系统击键消息)等等。

一般窗口用户程序不直接处理键盘事件，而是处理 MSG_CHAR、MSG_SYSCHAR 事件，即字符消息。字符消息是由消息循环代码中的 TranslateMessage 函数产生，TranslateMessage 函数是将键盘消息 MSG_KEYDOWN、MSG_SYSKEYDOWN 转换成字符消息的，同时也将按键扫描码转换成为相应的字符编码(字符消息中的 wParam 参数)。

对于字符消息的处理，用户只需要在窗口过程函数中添加相应的处理代码即可，更详细的说明请参考 MiniGUI 的应用资料。字符消息处理示范代码如程序清单 9.29 所示。

程序清单 9.29 字符消息处理示范

```
case   MSG_CHAR:                    /* 字符消息 */
       if(wParam==127)              /* 判断是否 Back Space 键，若是则删除前一个字符 */
       {    if(no>0)
            {   no--;
                str_buf[no] = ' ';
            }
       }
       else                         /* 不是 Back Space 键，添加字符 */
       {    str_buf[no++] = wParam; /* 取得字符，并加入 str_buf 字符串 */
            str_buf[no] = '\0';
       }

       if( ((no%30) == 0) ||        /* 以 30 个字符为一行 */
```

```
            (wParam == 0x0D )           /* 若有回车键，则换行 */
        )
    {   if(no>0)
        {   y = y + char_h;
            no = 0;
            sprintf(str_buf, "");
        }
    }

    hdc = BeginPaint(hWnd);
    TextOut(hdc, 0, y, str_buf);        /* 显示当前行字符 */
    EndPaint(hWnd, hdc);
    SetCaretPos(hWnd, no * char_w, y);  /* 设置插入符位置，即光标位置 */
    break;
```

6. 实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 sedit 子目录，并在 sedit 目录下新建 MiniGUI 应用程序 sedit.c 并编写实验代码。

(2) 再把 sedit.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(3) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(4) 启动 AXD 调试 MiniGUI 程序。正确运行 sedit 程序后在开发板的 TFT 液晶屏上将会显示一个简易编辑窗口。程序运行结果参考如图 9.13 所示。

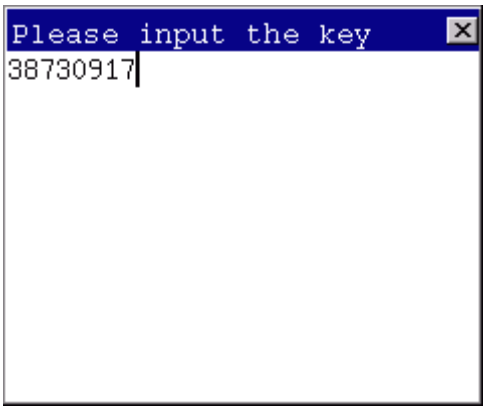


图 9.13 sedit.c 程序运行结果

(5) 使用开发板上的功能键 0～9 输入数字，简易编辑窗口上将会显示所输入的字符。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

简易编辑器实验的参考程序见程序清单 9.30。

程序清单 9.30　简易编辑器实验参考程序

```
/*****************************************************************************
* 文件名：Sedit.C
* 功能：MiniGUI 应用例子。
*       使用 MiniGUI 创建主窗口，在窗口过程函数中处理字符消息，实现一个简
*       易编辑器的功能。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

HWND hMainWnd;

static int    WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{    HDC    hdc;
     static char str_buf[1000];
     static int    no;
     static int    y;
     static int    char_w, char_h;

     switch(message)
     {    case MSG_CREATE:
               CreateCaret(hWnd, NULL, 2, GetSysCharHeight());       // create the caret is 2(width)
               SetCaretBlinkTime(hWnd, 500);
               char_w = GetSysCharWidth();
               char_h = GetSysCharHeight();
               sprintf(str_buf, "");
               no = 0;
               y = 0;
               break;

          case MSG_PAINT:
               hdc = BeginPaint(hWnd);
               TextOut(hdc, 0, y, str_buf);
               EndPaint(hWnd, hdc);
               SetCaretPos(hWnd, 0,0);
               break;
```

```
case MSG_SETFOCUS:
    ShowCaret(hWnd);
    break;

case MSG_KILLFOCUS:
    HideCaret(hWnd);
    break;

case MSG_CLOSE:
     DestroyCaret(hWnd);                    // destroy the caret
     DestroyMainWindow(hWnd);
     PostQuitMessage(hWnd);
     break;

case MSG_CHAR:
    if(wParam==127)
    {    if(no>0)
         {   no--;
             str_buf[no] = ' ';
         }
    }
    else
    {    str_buf[no++] = wParam;
         str_buf[no] = '\0';
    }

    if( ((no%30) == 0) ||
         (wParam == 0x0D )
      )
    {    if(no>0)
         {    y = y + char_h;
              no = 0;
              str_buf[0] = '\0';
         }
    }

    hdc = BeginPaint(hWnd);
    TextOut(hdc, 0, y, str_buf);
    EndPaint(hWnd, hdc);
    SetCaretPos(hWnd, no * char_w, y);
    break;

default:
```

```
            return(DefaultMainWinProc(hWnd, message, wParam, lParam));
    }
    return(0);
}


int InitMainWindow(void)
{   MAINWINCREATE   window_info;

    window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
    window_info.dwExStyle = WS_EX_IMECOMPOSE;
    window_info.spCaption = "Please input the key";
    window_info.hMenu = 0;
    window_info.hCursor = GetSystemCursor(0);
    window_info.hIcon = 0;
    window_info.MainWindowProc = WinProc;
    window_info.lx = 0;
    window_info.ty = 50;
    window_info.rx = 239;
    window_info.by = 250;
    window_info.iBkColor = COLOR_lightwhite;
    window_info.dwAddData = 0;
    window_info.hHosting = HWND_DESKTOP;
    hMainWnd = CreateMainWindow (&window_info);
    if (hMainWnd == HWND_INVALID) return(0);
      else   return(1);
}


int   MiniGUIMain(int argc, const char *argv[])
{   MSG Msg;

#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif

    InitMainWindow();
    ShowWindow(hMainWnd, SW_SHOWNORMAL);
    while (GetMessage(&Msg, hMainWnd))
    {   TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup (hMainWnd);
    return(0);
```

```
}

#ifndef _LITE_VERSION
    #include "dti.c"
#endif
```

8. 思考

请简单描述一下键盘消息和字符消息的差别？

9.7 位图显示实验

1. 实验目的

了解 MiniGUI 位图操作的相关 API 函数，并能够使用它们来实现位图的显示。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

创建一个 MiniGUI 主窗口，在窗口中间显示 Linux 吉祥物小企鹅(使用 MiniGUI-STR (for uLinux)的资源文件 logo256-s.bmp)。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解位图文件的装载和显示输出的相关内容。

5. 实验原理

MiniGUI 提供了众多的位图操作函数，支持单色、16 色、256 色和 24 位色的 BMP 文件、JPG 文件、GIF 文件和 PCX 文件等等，MiniGUI 可以从 File(使用文件系统读取文件)或 Memory(内嵌在工程代码中的数据，即内建式资源)中取得位图文件数据。

MiniGUI for uC/OS-II 只能使用内建式资源，如果要显示一个 BMP 图，则先要将此 BMP 文件(二进制值)转换成 C 语言数组，并加入应用工程的源文件中进行编译。然后使用 LoadBitmapFromMemory 函数读取 BMP 文件的数据，使用 FillBoxWithBitmap 函数将 BMP 图输出液晶屏显示。最后，使用 UnloadBitmap 函数清除已加载的位图文件。

更详细的说明请参考 MiniGUI 的应用资料。

6. 实验步骤

(8) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 dispbmp 子目录，并在 dispbmp 目录下新建 MiniGUI 应用程序 DispBmp.c 并编写实验代码，再把此文件加入到 GUI_Sample 工程的 usr 组中。

(9) 取得要显示的 BMP 文件 logo256-s.bmp，将其复制到 D 盘根目录下。使用一个名为 BintoArray.exe 的转换软件打开 D:\ logo256-s.bmp，如图 9.14 所示。单击“Change”按钮，即可生成 D:\logo256-s.bmp.h 文件，此文件如程序清单 9.31 所列。

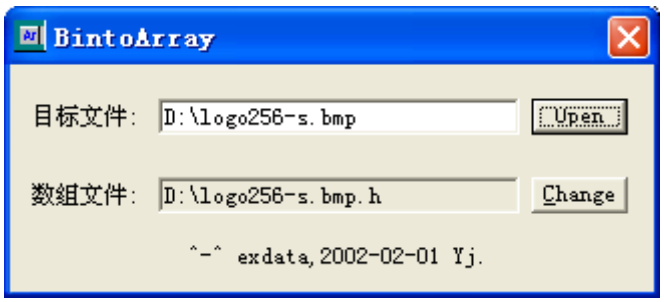


图 9.14　将 logo256-s.bmp 转换为数组

程序清单 9.31　BMP 文件转换生成的数组

```
/*      File D:\logo256-s.bmp.h      */
/*      Byte's number(DEC)=8816      */
unsigned char array[]={
      0x42, 0x4d, 0x70, 0x22, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x36, 0x04, 0x00, 0x00, 0x28, 0x00, 0x00, 0x00, 0x78, 0x00,
      ……
```

(10)将 D:\logo256-s.bmp.h 复制到 GUI_Sample 工程的 src\dispbmp 目录下，并将文件名改为 logo256-s.c。

(11)修改 logo256-s.c 文件，如程序清单 9.32 所列，其中 bmp1_data 为 BMP 文件的数据，sz_bmp1 为 bmp1_data 数组的大小(即 BMP 文件的大小)。

程序清单 9.32　修改后的 logo256-s.c 文件

```
/*      File D:\logo256-s.bmp.h      */
/*      Byte's number(DEC)=8816      */
const unsigned int     sz_bmp1 = 8816;
const unsigned char    bmp1_data[]={
      0x42,0x4d,0x70,0x22,0x00,0x00,0x00,0x00,0x00,0x00,
      0x36,0x04,0x00,0x00,0x28,0x00,0x00,0x00,0x78,0x00,
      ……
```

(12)将 logo256-s.c 添加到 GUI_Sample 工程的 usr 组中，编译连接工程。

(13)连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置

为 Bank0-FLASH 和 Bank1-RAM。

(14)启动 AXD 调试 MiniGUI 程序。如果一切正确，开发板的 TFT 液晶屏上将会显示一个可爱的小企鹅，程序运行结果参考如图 9.15 所示。



图 9.15　DispBmp.c 程序运行结果

说明：可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

位图显示实验的参考程序见程序清单 9.33。

程序清单 9.33　位图显示实验参考程序

```
/*****************************************************************************
* 文件名：DispBmp.C
* 功能：MiniGUI 应用例子。
*       使用 LoadBitmapFromMem 装载一个 BMP 文件数据(内建资源)，然后使用
*       FillBoxWithBitmap 在主窗口上显示该图片。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/

/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/* 主窗口起始位置及大小 */
#define   MWINDOW_LX        10        /* 窗口左边框的 x 值 */
#define   MWINDOW_TY        10        /* 窗口上边框的 y 值 */
```

```
#define   MWINDOW_RX        220       /* 窗口右边框的 x 值 */
#define   MWINDOW_BY        220       /* 窗口下边框的 y 值 */

/* 声明 bmp 文件大小及数据指针 */
extern const unsigned int     sz_bmp1;
extern const unsigned char    bmp1_data[];

HWND   hMainWnd;         // 主窗口句柄

/*************************************************************************
* 名称：WinProc()
* 功能：主窗口过程函数。
*       处理 MSG_PAINT 消息，在窗口中显示 BMP 图片。
* 入口参数：hWnd          窗口句柄
*           message       消息
*           wParam        消息附加参数 1(对于不同的消息，有不同的用途)
*           lParam        消息附加参数 2(对于不同的消息，有不同的用途)
* 出口参数：消息已处理则返回 0。
*************************************************************************/
static int   WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{   static BITMAP   s_bmp;
    HDC   hdc;                  // 定义一个图形设备上下文对象，输出字符时需要。

    switch(message)
    {       case MSG_CREATE:
                if(LoadBitmapFromMem( HDC_SCREEN,   // 设备上下文
                                      &s_bmp,       // BITMAP 变量地址
                                      bmp1_data,    // 图形文件数据
                                      sz_bmp1,      // 文件大小
                                      "bmp") )      // 文件类型
                {   return(-1);
                }
                break;

            case MSG_PAINT:
                hdc = BeginPaint(hWnd);             // 获得设备 hdc
                FillBoxWithBitmap (hdc, 10, 10, 120, 141, &s_bmp);   // 显示图片
                EndPaint(hWnd, hdc);                // 释放设备 hdc
                break;

            case MSG_CLOSE:
                UnloadBitmap (&s_bmp);
                DestroyMainWindow(hWnd);            // 销毁主窗口
                PostQuitMessage(hWnd);              // 发送 MSG_QUIT 消息，通知"消息循环"结束
```

```
                break;

            default:
                return(DefaultMainWinProc(hWnd, message, wParam, lParam));
        }
        return(0);
}

/*************************************************************************
* 名称：InitMainWindow()
* 功能：建立主窗口。
* 入口参数：无
* 出口参数：建立成功返回 1，否则返回 0。
*************************************************************************/
int InitMainWindow(void)
{   MAINWINCREATE   window_info;

    window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
    window_info.dwExStyle = WS_EX_NONE;
    window_info.spCaption = "MiniGUI";
    window_info.hMenu = 0;
    window_info.hCursor = GetSystemCursor(0);
    window_info.hIcon = 0;
    window_info.MainWindowProc = WinProc;
    window_info.lx = MWINDOW_LX;
    window_info.ty = MWINDOW_TY;
    window_info.rx = MWINDOW_RX;
    window_info.by = MWINDOW_BY;
    window_info.iBkColor = COLOR_lightwhite;
    window_info.dwAddData = 0;
    window_info.hHosting = HWND_DESKTOP;

    hMainWnd = CreateMainWindow(&window_info);
    if(hMainWnd == HWND_INVALID) return(0);
        else return(1);
}

/*************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc      参数个数
*           argv      参数字符串指针
* 出口参数：返回 0。
*************************************************************************/
```

```
int    MiniGUIMain(int argc, const char *argv[])
{     MSG Msg;

/*  虽然 MiniGUI for uC/OS-II 不支持"MiniGUI-Lite 模式"，
    但为了保持代码的移植性，此段不要删除     */
#ifdef _LITE_VERSION
      SetDesktopRect(0,0, 800,600);
#endif

      InitMainWindow();              // 建立主窗口
      ShowWindow(hMainWnd, SW_SHOWNORMAL);         // 显示主窗口

      /*  消息循环  */
      while(GetMessage(&Msg, hMainWnd))
      {     TranslateMessage(&Msg);
            DispatchMessage(&Msg);
      }

      MainWindowThreadCleanup(hMainWnd);
      return(0);
}

/*  定义桌面接口函数  */
#ifndef _LITE_VERSION
            #include "dti.c"
#endif
```

8． 思考

在MiniGUI-STR for uC/OS-II中，如何设置桌面背景图片？(提示：在MiniGUI_Lib\libmin igui-str-1.6.2-ucosii\src\sysres 目录下的 mgetc.c 文件中，将 BGPICTURE_VALUES[]的值定义为“center”即可使用背景图片，图片资源在相同目录下的 bmps-3d.c 文件中)

9.8 GDI 绘图实验

1． 实验目的

熟悉 MiniGUI 的绘图属性，掌握基本 GDI 绘图函数的应用。

2． 实验设备

- 硬件：PC 机　　一台
  SmartARM2200 教学实验开发平台　　一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3. 实验内容

使用 MiniGUI 创建主窗口，然后在窗口中以客户区中心点为中心，画出随机颜色、随机长度和随机角度的 50 根直线。

4. 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200 开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解 MiniGUI 的 GDI 绘图函数的使用方法。

5. 实验原理

MiniGUI 提供了众多的 GDI 绘图函数，包括基本的二维图形绘制函数，矩形操作，区域操作，位图显示和文字输出等等。进行 GDI 绘图操作之前，首先要获得设备 hdc(即设备上下文)，然后调用相应的 GDI 函数进行设置及绘图操作，最后还需要释放设备 hdc。注意，程序必须在处理单个消息期间获取设备 hdc 和释放设备 hdc。一般用户可以通过 BeginPaint、GetDC 函数来取得设备 hdc，然后使用 EndPaint、ReleaseDC 函数释放设备 hdc。BeginPaint 是只针对窗口的客户区，而 GetDC 则是针对整个窗口。函数 BeginPaint 和 EndPaint 常用于窗口过程函数中的 MSG_PAINT 消息处理中。

更详细的说明请参考 MiniGUI 的应用资料。

6. 实验步骤

(1) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 line 子目录，并在 line 目录下新建 MiniGUI 应用程序 line.c 并编写实验代码。

(2) 再把 line.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

说明：要先将工程 usr 组中旧的实验程序文件(如 messageb.c)从工程中删除掉。

(3) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(4) 启动 AXD 调试 MiniGUI 程序。如果一切正确，在开发板的 TFT 液晶屏上可以观察到画有随机直线的窗口。程序运行结果参考如图 9.16 所示。

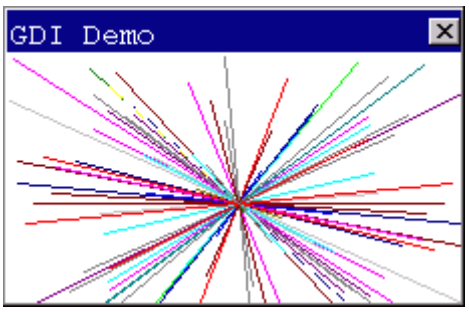


图 9.16 line.c 程序运行结果

说明：可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

GDI 绘图实验的参考程序见程序清单 9.34。

程序清单 9.34　GDI 绘图实验参考程序

```
/*****************************************************************************
* 文件名：Line.C
* 功能：MiniGUI 应用例子。
*       创建一个主窗口，并在窗口中间画出随机颜色、随机长度和随机角度
*       的 50 根直线。
* 说明：使用 MiniGUI for uC/OS-II，使用 ADS 1.2 编译器。
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>

/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/* 定义 random()函数 */
#define   random()      rand()

/* 主窗口起始位置及大小 */
#define   MWINDOW_LX      5       /* 窗口左边框的 x 值 */
#define   MWINDOW_TY      50      /* 窗口上边框的 y 值 */
#define   MWINDOW_RX      235     /* 窗口右边框的 x 值 */
#define   MWINDOW_BY      200     /* 窗口下边框的 y 值 */

/* 定义窗口宽度、高度的宏 */
#define   MWINDOW_W    (MWINDOW_RX - MWINDOW_LX)
#define   MWINDOW_H    (MWINDOW_BY - MWINDOW_TY)

HWND hMainWnd;          // 主窗口句柄

/*****************************************************************************
* 名称：WinProc()
* 功能：主窗口过程函数。
*       处理 MSG_PAINT 消息，在窗口中间画出随机颜色、随机长度和随机角度
*       的 50 根直线。
```

```
* 入口参数：hWnd          窗口句柄
*           message       消息
*           wParam        消息附加参数 1
*           lParam        消息附加参数 2
* 出口参数：消息已处理则返回 0。
*********************************************************************************/
static int    WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{    HDC    hdc;
     int    x0, y0;      // 直线第一点坐标变量
     int    x1, y1;      // 直线第二点坐标变量
     int    no;

     switch(message)
     {     case MSG_PAINT:
               hdc = BeginPaint(hWnd);

               for(no=0; no<50; no++)                    // 画 50 条随机直线
               {     SetPenColor(hdc, SysPixelIndex[random()%16]);       // 取得随机颜色
                     x0 = random()%MWINDOW_W;     // 随机取得直线第一点坐标值
                    y0 = random()%MWINDOW_H;

                    /* 以中心点对称，计算出直线第二点坐标值。
                        设 x0 在中点左边，中点值为 W/2，则中点到 x0 的长度 Mx = W/2 - x0，
                        所以 x1 = W/2 + Mx = W/2 + W/2 - x0 = W - x0。
                        若 x0 在中点右边，也会得到同要的结果(x1=W - x0)。
                    */
                    x1 = MWINDOW_W - x0;
                    y1 = MWINDOW_H - y0;
                    MoveTo(hdc, x0, y0);            // 移动 hdc 当前位置到直线第一点处
                    LineTo(hdc, x1, y1);            // 画直线到直线第二点
               }
               EndPaint(hWnd, hdc);
               break;

          case MSG_CLOSE:
               DestroyMainWindow(hWnd);
               PostQuitMessage(hWnd);
               break;

          default:
               return(DefaultMainWinProc(hWnd, message, wParam, lParam));
     }
     return(0);
}
```

```
/*************************************************************************
* 名称：InitMainWindow()
* 功能：建立主窗口。
* 入口参数：无
* 出口参数：建立成功返回 1，否则返回 0。
*************************************************************************/
int InitMainWindow(void)
{    MAINWINCREATE    window_info;

     window_info.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
     window_info.dwExStyle = WS_EX_NONE;
     window_info.spCaption = "GDI Demo";
     window_info.hMenu = 0;
     window_info.hCursor = GetSystemCursor(0);
     window_info.hIcon = 0;
     window_info.MainWindowProc = WinProc;
     window_info.lx = MWINDOW_LX;
     window_info.ty = MWINDOW_TY;
     window_info.rx = MWINDOW_RX;
     window_info.by = MWINDOW_BY;
     window_info.iBkColor = COLOR_lightwhite;
     window_info.dwAddData = 0;
     window_info.hHosting = HWND_DESKTOP;
     hMainWnd = CreateMainWindow(&window_info);
     if(hMainWnd == HWND_INVALID) return(0);
       else return(1);
}


/*************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc       参数个数
*           argv       参数字符串指针
* 出口参数：返回 0。
*************************************************************************/
int   MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
     SetDesktopRect(0,0, 800,600);
#endif
```

```
    srand(3721);      // 初始化随机总机

    InitMainWindow();
    ShowWindow(hMainWnd, SW_SHOWNORMAL);
    while(GetMessage(&Msg, hMainWnd))
    {    TranslateMessage(&Msg);
         DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup(hMainWnd);
    return(0);
}

#ifndef _LITE_VERSION
     #include "dti.c"
#endif
```

8． 思考

如果想要在窗口的随机位置，显示随机大小(当然还是要规定在合理的范围内)随机颜色的圆，应如何修改实验参考程序？

9.9 桌面主题实验

1． 实验目的

熟悉 MiniGUI 的窗口程序设计，能够使用 MiniGUI 的常用 API 函数来实现一些特殊窗口的设计。

2． 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：Windows98/XP/2000 系统，ADS 1.2 集成开发环境
  μC/OS-II 操作系统(V2.52)
  MiniGUI-STR(for uC/OS-II)软件

3． 实验内容

使用 MiniGUI 创建一个桌面窗口(即全屏窗口)，在桌面上显示 6 个 Windows XP 操作系统风格的图标，以及 Windows XP 操作系统风格的任务栏，并可以通过按键操作选择各图标功能项。

4． 实验预习要求

(1) 必须先做第 1 节的实验，搭建好 MiniGUI 应用工程环境，掌握交叉编译 MiniGUI 应用程序的操作方法和下载到 SmartARM2200 开发板并运行。

(2) 仔细阅读<<ARM 嵌入式系统实验教程(二)>>第 1 章的内容，了解 SmartARM2200

开发板的硬件结构，注意 TFT 液晶接口电路和键盘电路。

(3) 阅读 MiniGUI 的文档<<MiniGUI 编程指南>>，了解 MiniGUI 的窗口风格、位图显示、逻辑字体和图形设备上下文等相关知识。

5. 实验原理

1) 窗口风格

窗口风格是用来控制窗口的外观及行为方式，比如窗口的边框类型、是否有滚动条、是否有最小化/最大化按钮等等，在创建窗口时需要指定窗口的风格。

MiniGUI 窗口的风格分为普通风格和扩展风格两种，风格标识定义在 libminigui-str-1.6.2/include/window.h 文件中，它们分别以 WS_和 WS_EX 的形式开头。常用窗口风格如下：

WS_VISIBLE，窗口可见；
WS_CAPTION，窗口含标题栏；
WS_SYSMENU，窗口含系统菜单；
WS_BORDER，窗口包含窗口边框；
WS_THICKFRAME，使用厚边框；
WS_THINFRAME，使用薄边框；
WS_VSCROLL，窗口包含垂直滚动条；
WS_HSCROLL，窗口包含水平滚动条；
WS_MINIMIZEBOX，窗口带最小化按钮；
WS_MAXIMIZEBOX，窗口带最大化按钮。

2) 逻辑字体

当用户需要使用不同的字体、大小和显示风格进行文字显示时，则要建立逻辑字体，建立逻辑字体时设置相应的字体类型和风格等参数，然后设置设备 hdc 使用新建的逻辑字体，即可输出相应的文字。

建立逻辑字体的函数为 CreateLogFont、CreateLogFontByName 和 CreateLogFontIndirect；获取系统逻辑字体的函数为 GetSystemFont；获取指定设备 hdc 使用的逻辑字体的函数为 GetCurFont；设置设备 hdc 使用指定逻辑字体的函数为 SelectFont。

更详细的说明请参考 MiniGUI 的应用资料。

6. 实验步骤

(1) 在 PC 机上找到一些 Windows XP 操作系统的图标，规格为 48×48 点真彩色。
(2) 由于 MiniGUI 只提供对单色和 16 色 Windows 图标的支持，而本实验需要的图标为真彩色的，所以要将图标转换为 BMP 图并保存(文件名为 no01.bmp～no06.bmp)，以便于使用 BMP 图片的形式进行显示输出。
(3) 在 PC 机上使用绘图工具制作一个大小为 240×37 的类似于 Windows XP 的任务栏，保存为 BMP 文件(文件名为 start.bmp)。
(4) 使用一个名为 BintoArray.exe 的转换软件，将这些 BMP 文件转换为数组，保存文件名分别为 no01.c～no06.c、start.c。然后修改转换文件中的数组名并定义相应的数据大小，修改点参考如程序清单 9.15 所示，其中 xxx_data 为 BMP 文件的数据，sz_xxx 为 xxx_data 数组的大小(即 BMP 文件的大小)。

程序清单 9.35　BMP 文件转换生成的数组

```
/*       File D:\res_bmp\no01.bmp.h       */
/*       Byte's number(DEC)=6966       */
const unsigned int      sz_no01 = 6966;
const unsigned char     no01_data[]={
        0x42, 0x4d, 0x36, 0x1b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x36, 0x00, 0x00, 0x00, 0x28, 0x00, 0x00, 0x00, 0x30, 0x00,
        0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x01, 0x00, 0x18, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x1b, 0x00, 0x00, 0x00, 0x00,
        ......
```

(5) 使用 GUI_Sample 工程，在工程目录的 src 目录下建立一个 desktop 子目录，并在 desktop 目录下新建 MiniGUI 应用程序 desktop.c 并编写实验代码。

(6) 将文件 no01.c～no06.c、start.c 复制到 GUI_Sample 工程的 src\desktop 目录下。

(7) 再把 desktop.c 文件加入到 GUI_Sample 工程的 usr 组中，编译连接工程。

(8) 连接好 SmartARM2200 教学实验开发平台硬件(JTAG 仿真器、TFT4267 PACK 板)，将 JP3 跳线断开，JP1、JP6 跳线短接，JP9 跳线设置为 OUTSIDE，JP10 跳线设置为 Bank0-FLASH 和 Bank1-RAM。

(9) 启动 AXD 调试 MiniGUI 程序。正确运行 desktop 程序后在开发板的 TFT 液晶屏上可以观察到 XP 主题的桌面。程序运行结果参考如图 9.17 所示。



图 9.17　desktop.c 运行结果参考图

(10)使用开发板上的功能键 1～3、6 控制图标菜单的选择(键 1 表示←，键 2 表示↓，键 3 表示→，键 6 表示↑)，使用功能键 8 确定选择。

说明：SmartARM2200 开发板上的功能键定义如图 9.2 所示。可以使用 ZLG/BOO 将工程输出文件 minigui.bin 文件下载到开发板上的 boot 目录下，然后再由 ZLG/BOOT 加载程序到片外 RAM 并运行程序。

7. 实验参考程序

桌面主题实验的参考程序见程序清单 9.36。

程序清单 9.36 桌面主题实验参考程序

```
/*****************************************************************************
* 文件名：desktop.c
* 功能：MiniGUI 应用例子。
*       创建一个无边框和标题的桌面窗口，然后在桌面上显示 6 个图标按钮，
*       使用按键进行选择操作。
*****************************************************************************/
/* 包含 MiniGUI 的配置头文件(编译配置选项) */
#include "MiniGUI_config.h"

/* 包含 MiniGUI 头文件 */
#include "common.h"
#include "minigui.h"
#include "gdi.h"
#include "window.h"
#include "control.h"

/* 主窗口起始位置及大小 */
#define   MWINDOW_LX        0          /* 窗口左边框的 x 值 */
#define   MWINDOW_TY        0          /* 窗口上边框的 y 值 */
#define   MWINDOW_RX        239        /* 窗口右边框的 x 值 */
#define   MWINDOW_BY        319        /* 窗口下边框的 y 值 */

/* 包含各图片资源文件 */
#include   "no01.c"
#include   "no02.c"
#include   "no03.c"
#include   "no04.c"
#include   "no05.c"
#include   "no06.c"
#include   "start.c"

HWND hMainWnd;          // 主窗口句柄

const char*   menu_hz[] = { "我的电脑",
                            "回收站",
                            "网络邻居",
                            "IE 浏览器",
                            "媒体播放机",
                            "数码相机"
                            };
```

```
/*************************************************************************
* 名称：WinProc()
* 功能：主窗口过程函数。
*       处理 MSG_PAINT 消息，在窗口中显示 6 个图标。
*       处理 MSG_CHAR 消息，进行图标菜单选择操作。
* 入口参数：hWnd          窗口句柄
*           message       消息
*           wParam        消息附加参数 1(对于不同的消息，有不同的用途)
*           lParam        消息附加参数 2(对于不同的消息，有不同的用途)
* 出口参数：消息已处理则返回 0。
*************************************************************************/
static int   WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{    static BITMAP   s_startbmp;
     static BITMAP   s_bmp[6];
     static PLOGFONT s_font, s_font1;
     static int s_sel = 0;
     static int s_selbak = 0;
     int     i;
     HDC   hdc;    // 定义一个图形设备上下文对象，输出字符时需要。

     switch(message)
     {    case MSG_CREATE:
               // 装载图片资源
               LoadBitmapFromMem(HDC_SCREEN, &s_startbmp, start_data, start_no, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[0], no01_data, sz_no01, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[1], no02_data, sz_no02, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[2], no03_data, sz_no03, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[3], no04_data, sz_no04, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[4], no05_data, sz_no05, "bmp");
               LoadBitmapFromMem(HDC_SCREEN, &s_bmp[5], no06_data, sz_no06, "bmp");

               // 建立逻辑字体
               s_font = CreateLogFont( NULL, "song", "GB2312",
               FONT_WEIGHT_BOLD, FONT_SLANT_ITALIC, FONT_SETWIDTH_NORMAL,
               FONT_SPACING_CHARCELL, FONT_UNDERLINE_NONE, FONT_STRUCKOUT_NONE,
                                   16, 0);
               s_font1 = CreateLogFont( NULL, "song", "GB2312",
               FONT_WEIGHT_BOLD, FONT_SLANT_ROMAN, FONT_SETWIDTH_NORMAL,
               FONT_SPACING_CHARCELL, FONT_UNDERLINE_NONE, FONT_STRUCKOUT_NONE,
                                   16, 0);
               break;

          case MSG_PAINT:
               hdc = BeginPaint(hWnd);              // 获得设备 hdc
```

```
        // 显示各图标
        for(i=0; i<4; i++)
        {       FillBoxWithBitmap(hdc, 10, i*70+10, 48,48, &s_bmp[i]);
        }
        for(i=0; i<2; i++)
        {       FillBoxWithBitmap(hdc, 80, i*70+10, 48,48, &s_bmp[i+4]);
        }

        // 显示窗口"开始"菜单
        FillBoxWithBitmap(hdc, 0, 283, 240, 37, &s_startbmp);
        SetBkMode(hdc, BM_TRANSPARENT);
        SetTextColor(hdc, COLOR_lightwhite);
        SelectFont(hdc, s_font);
        TextOut(hdc, 30, 295, "开 始");

        // 设置选中的菜单项(选择边框)
        SetPenColor(hdc, COLOR_black);
        if(s_sel<4)
        {       Rectangle(hdc, 8,s_sel*70+8, 60,s_sel*70+60);
        }
        else
        {       Rectangle(hdc, 78,(s_sel-4)*70+8, 130,(s_sel-4)*70+60);
        }
        SelectFont(hdc, s_font1);
        TextOut(hdc, 130, 295, menu_hz[s_sel]);
        EndPaint(hWnd, hdc);          // 释放设备 hdc
        break;

case MSG_CHAR:
        i = 0;                              // 使用 i 作为更新显示标志
        switch(wParam)
        {       case '1':               // 左移
                        if(s_sel>3)
                        {       s_selbak = s_sel;
                                s_sel = s_sel-4;
                                i = 1;
                        }
                        break;
                case '2':               // 下移
                        if(s_sel<5)
                        {       s_selbak = s_sel;
                                s_sel++;
                                i = 1;
```

```
                }
                else
                {       s_selbak = 0;
                        s_sel = 5;
                }
                break;
        case '3':               // 右移
                if(s_sel<4)
                {       s_selbak = s_sel;
                        s_sel += 4;
                        if(s_sel>5) s_sel = 5;
                        i = 1;
                }
                break;
        case '6':               // 上移
                if(s_sel<=0)
                {       s_selbak = 5;
                        s_sel = 0;
                }
                else
                {       s_selbak = s_sel;
                        s_sel--;
                        i = 1;
                }
                break;

        case '8':               // 确定选择，执行相应的功能
                MessageBox(hMainWnd, "功能正在建设...", "提示", MB_OK);
                break;

        default:
                break;
}
if(i==1)
{       hdc = BeginPaint(hWnd);                 // 获得设备 hdc

        // 清除旧的选择项(选择边框)
        SetPenColor(hdc, COLOR_lightgray);
        if(s_selbak<4)
        {       Rectangle(hdc, 8,s_selbak*70+8, 60,s_selbak*70+60);
        }
        else
        {       Rectangle(hdc, 78,(s_selbak-4)*70+8, 130,(s_selbak-4)*70+60);
        }
```

```
                    // 设置新的选择项(选择边框)
                    SetPenColor(hdc, COLOR_black);
                    if(s_sel<4)
                    {     Rectangle(hdc, 8,s_sel*70+8, 60,s_sel*70+60);
                    }
                    else
                    {     Rectangle(hdc, 78,(s_sel-4)*70+8, 130,(s_sel-4)*70+60);
                    }

                    // 重新显示"开始"菜单
                    FillBoxWithBitmap(hdc, 0, 283, 240, 37, &s_startbmp);
                    SetBkMode(hdc, BM_TRANSPARENT);
                    SetTextColor(hdc, COLOR_lightwhite);
                    SelectFont(hdc, s_font);
                    TextOut(hdc, 30, 295, "开 始");
                    SelectFont(hdc, s_font1);
                    TextOut(hdc, 130, 295, menu_hz[s_sel]);

                    EndPaint(hWnd, hdc);                    // 释放设备 hdc
                }
                break;

        case MSG_CLOSE:
                UnloadBitmap(&s_startbmp);
                for(i=0; i<6; i++) UnloadBitmap(&s_bmp[i]);
                DestroyLogFont(s_font);
                DestroyLogFont(s_font1);
                DestroyMainWindow(hWnd);              // 销毁主窗口
                PostQuitMessage(hWnd);                // 发送 MSG_QUIT 消息，通知"消息循环"结束
                break;

        default:
                return(DefaultMainWinProc(hWnd, message, wParam, lParam));
    }
    return(0);
}


/*************************************************************************
* 名称：InitMainWindow()
* 功能：建立主窗口。
* 入口参数：无
* 出口参数：建立成功返回 1，否则返回 0。
```

```
*************************************************************************/
int InitMainWindow(void)
{    MAINWINCREATE    window_info;

     window_info.dwStyle = WS_VISIBLE;
     window_info.dwExStyle = WS_EX_NONE;
     window_info.spCaption = "";
     window_info.hMenu = 0;
     window_info.hCursor = GetSystemCursor(0);
     window_info.hIcon = 0;
     window_info.MainWindowProc = WinProc;
     window_info.lx = MWINDOW_LX;
     window_info.ty = MWINDOW_TY;
     window_info.rx = MWINDOW_RX;
     window_info.by = MWINDOW_BY;
     window_info.iBkColor = COLOR_lightgray;
     window_info.dwAddData = 0;
     window_info.hHosting = HWND_DESKTOP;
     hMainWnd = CreateMainWindow(&window_info);
     if(hMainWnd == HWND_INVALID) return(0);
         else return(1);
}


/*************************************************************************
* 名称：MiniGUIMain()
* 功能：MiniGUI 程序入口点。
* 入口参数：argc      参数个数
*           argv      参数字符串指针
* 出口参数：返回 0。
*************************************************************************/
int   MiniGUIMain(int argc, const char *argv[])
{    MSG Msg;

#ifdef _LITE_VERSION
     SetDesktopRect(0,0, 800,600);
#endif

     InitMainWindow();              // 建立主窗口
     ShowWindow(hMainWnd, SW_SHOWNORMAL);      // 显示主窗口

     /* 消息循环 */
     while(GetMessage(&Msg, hMainWnd))
     {    TranslateMessage(&Msg);
```

```
        DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup(hMainWnd);
    return(0);
}

/* 定义桌面接口函数 */
#ifndef _LITE_VERSION
        #include "dti.c"
#endif
```

8. 思考

请简单描述一下实验参考程序是通过什么方法产生桌面窗口(全屏应用的窗口)，又是如何实现桌面上的图标功能(显示和选择操作)？

μClinux 篇

第10章　μCLinux 开发平台构建

10.1 宿主机和目标机

由于嵌入式 Linux 的开发板资源有限，不可能在开发板上运行开发和调试工具。通常需要**交叉编译调试**的方式进行，即“**宿主机＋目标机（开发板）**”的形式。

目标机和宿主机一般采用串口连接，亦可同时通过网口连接，如图 10.1 所示。

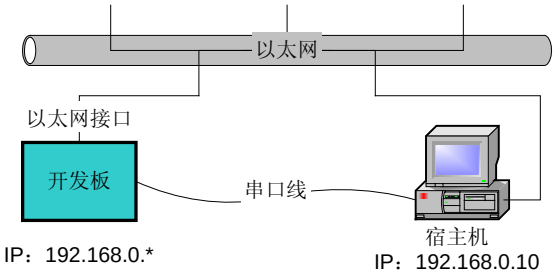


图 10.1　嵌入式 Linux 应用程序调试模型

宿主机是运行 Linux 的 PC，可以是安装了 Linux 操作系统的本地机，亦可以是 Linux 服务器（模型假定其 IP 为 192.168.0.10）。

宿主机和目标机的处理器通常情况下都不会相同。宿主机需要建立适合于目标机的交叉编译环境。程序在宿主机上编译－连接－定位，得到的可执行文件则在目标机运行。

在开发调试过程中，通常的操作界面如图 10.2 所示。

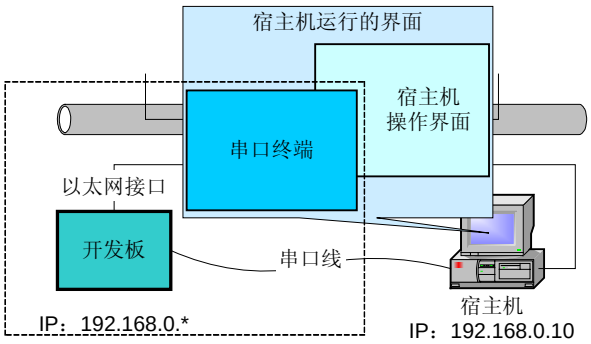


图 10.2　嵌入式 Linux 开发调试界面

在宿主机上一般需要运行 2 个窗口：宿主机本机操作窗口和串口终端窗口。

- 宿主机操作窗口可以是本机的操作终端（安装 Linux 的本地机），亦可以是 Windows 下通过远程登录软件如 Telnet 登录到 Linux 服务器的操作界面。宿主机只能编译－链接程序，不能运行产生的 flat 文件（GCC 模式的交叉编译调试方式）。
- 串口终端，可以是 Linux 的 minicom，亦可以是 Windows 的超级终端。目标机可以看成一台计算机，串口终端就相当于这台计算机的显示器，作为人机交互界面。在宿主机编译链接后得到的可执行文件，下载到目标机上运行。

10.2 建立交叉开发环境

10.2.1　建立开发环境

宿主机是运行 Linux 的 PC 或者服务器。由于 Linux 版本众多，各版本之间存在差异，

出于兼容性考虑，推荐安装 RedHat Linux 9.0。在安装 RedHat Linux 的时候，建议选择 Custom（定制）安装，在选择软件 Package 时，选择 Everything，即安装所有软件包。

由于目标板平台处理器是 ARM，因而需要安装 GCC 针对 ARM 的编译器。适用于 μCLinux 的编译器为 arm-elf-gcc。

arm-elf-gcc 安装步骤：

（1）将光盘附带的文件 arm-elf-tools-20040427.sh 拷贝到目录如/usr/src 下，为 arm-elf-tools-20040427.sh 增加可执行权限（如果已经具有可执行权限，这步可省略）。注意：增加权限必须具有 root 权限，若没有 root 权限，请先为用户增加 root 权限（命令 su）。

```
#chmod 755 arm-elf-20040427.sh
```

（2）安装 arm-elf-gcc。执行 arm-elf-tools-20040427.sh。

```
./arm-elf-tools-20040427.sh
```

安装完毕后，查看/usr/local/bin 目录下是否存在 arm-elf-gcc 等文件，如果存在，基本可以确定交叉编译器安装成功。

（3）添加交叉编译器的路径。一般情况下安装过程会自动执行，并在下次启动还有效。

```
export PATH=$PATH:/usr/local/arm-elf/bin
```

10.2.2 安装μCLinux

1：将光盘附带的μCLinux 源码包 uClinux-dist-20040408.tar.gz 拷贝到目录如/usr/src 下，并解压。

```
tar xzvf uClinux-dist-20040408.tar.gz
```

解压后，在当前目录下会多出文件夹 uClinux-dist，即μCLinux 源代码目录。

2：为 SmartARM2200 开发板和 LPC2200 芯片打补丁。将光盘附带的补丁文件 uClinux-dist-20040408-lpc-chy-cmj.patch 拷贝到当前目录下，并执行：

```
cat uClinux-dist-20040408-lpc-chy-cmj.patch | patch –p1 –d uClinux-dist
```

为 LPC2200 打补丁后，在μCLinux 平台内核配置的时候将会有 LPC2200 的选项。

10.2.3 配置μCLinux

（1）配置方式

可以通过 3 种方式配置μCLinux，分别是 make config、make menuconfig 和 make xconfig。各命令使用范围不一样，但效果是相同的。各命令说明如下：

```
make config          －命令行方式，适用于控制台和图形终端
make menuconfig      －文本菜单方式，适用于控制台和图形终端
make xconfig         －X 窗口图形界面方式，仅适用于图形终端
```

由于文本菜单的配置方式，既可在终端控制台使用，亦可在图形终端下使用，再由于其配置直观性，因而下面介绍这种配置方式。

（2）配置μCLinux 和文件系统

1：新建终端或者在控制台（CTRL＋ALT＋F1～F6 进入 6 个控制台的 1 个）下，进入 μCLinux 源代码所在的目录，如/usr/src/uClinux-dist。

```
cd /usr/src/uClinux-dist
```

2：如果不是进行第一次编译，需要先清除以前编译产生的辅助文件和目标文件，以保证编译的正确性。如果是第一次编译则可省略。

```
make distclean
```

3：通过文本菜单方式配置μCLinux。

```
make menuconfig
```

执行 make menuconfig 语句后，将会出现如图 10.3 所示的界面。

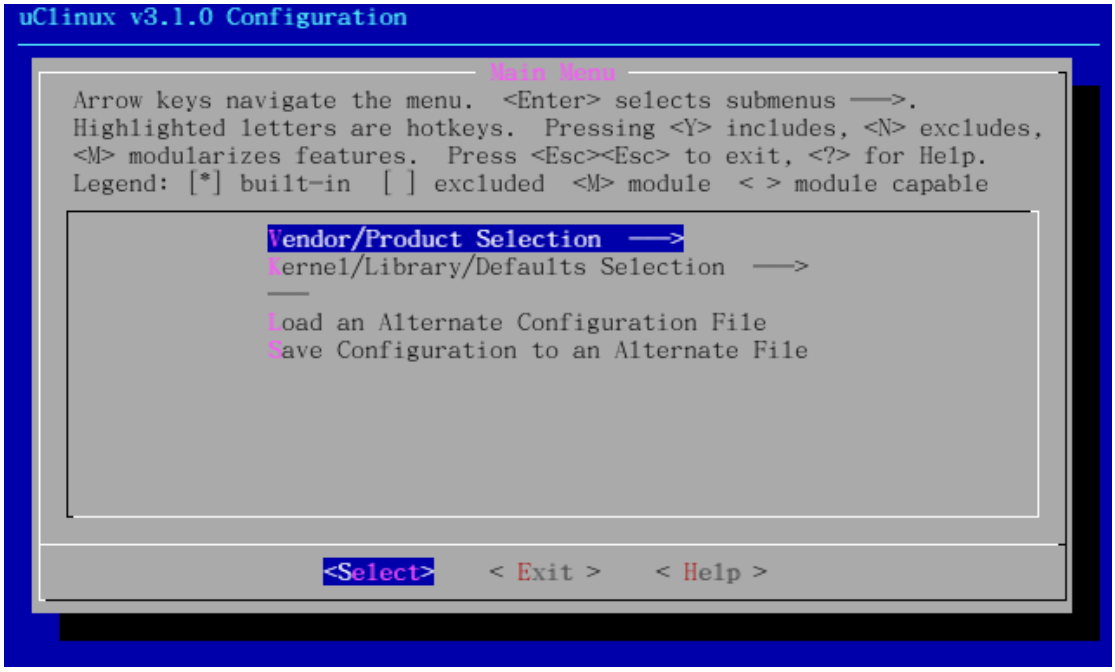


图 10.3 配置目标平台

界面中有提示：使用“↑”“↓”选择配置模块，使用空格键选择或者取消，使用“←”“→”选择命令，使用回车键 Enter 执行命令。

第一项可进行厂商/产品设置，选择第二项则进行内核版本和函数库设置。各选项的含义以及设置如表 10.1 所示。

表 10.1 平台选择设置

选项	描述	设置
Vendor/Product	厂商/产品选择	PHILIPS/lpc2200
Kernel Version	内核版本选择	Linux-2.4.x
Libc Verson	函数库版本选择	uClibc
Default all Settings	默认设置	恢复默认设置时选择（保留补丁信息）
Customize Kernel Settings	定制内核	根据需要选择
Customize Vendor/User Settings	定制厂商和用户设置	根据需要选择
Update Default Vendor Settings	更改设置为默认厂商配置	根据需要选择，一般不用

使用光盘提供的补丁文件后，只需对平台进行基本设置，最后选择 Default all Settings 后退出保存即可。

如果要对内核或者厂商等进行特殊设置，请选中 Customize Kernel Settings 或者 Customize Vendor/User Settings。使用光盘附带的μCLinux -dist 包，可以不进行特殊设置，不过，用户可选中这 2 个选项，查看内核和用户程序的具体配置，参考。

如果用户在配置过程中进行了不恰当的配置，导致编译出错，选择 Default all Settings 后进行编译即可。选择该项，为 LPC2200 打的补丁信息依然保留。

如果用户不需要对内核或者厂商进行特殊设置，选择 Exit 后则直接跳转到第 4 步。

如果用户在平台配置过程中选中了 Customize Kernel Settings，在平台配置完毕即可进入 Linux 内核配置界面，如图 10.4 所示。

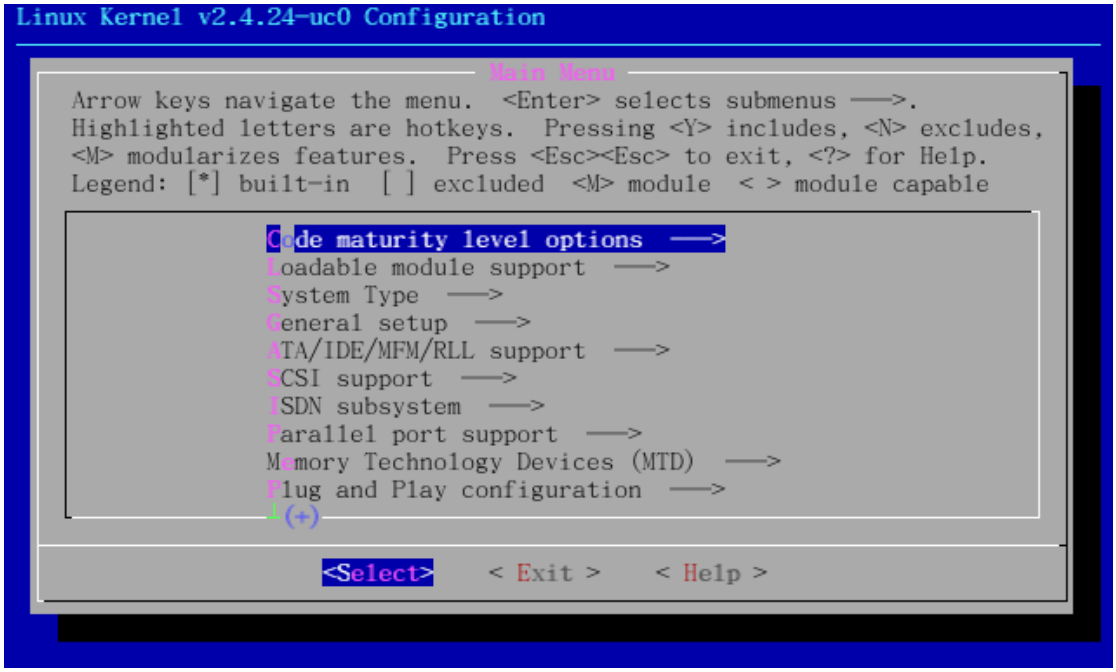


图 10.4 Linux 内核配置界面

配置完毕后选择 Exit，然后在提示是否保存的时候选择保存。

如果用户在平台配置过程中选择了 Customize Vendor/User Settings，则可进入用户程序配置界面，如图 10.5 所示。

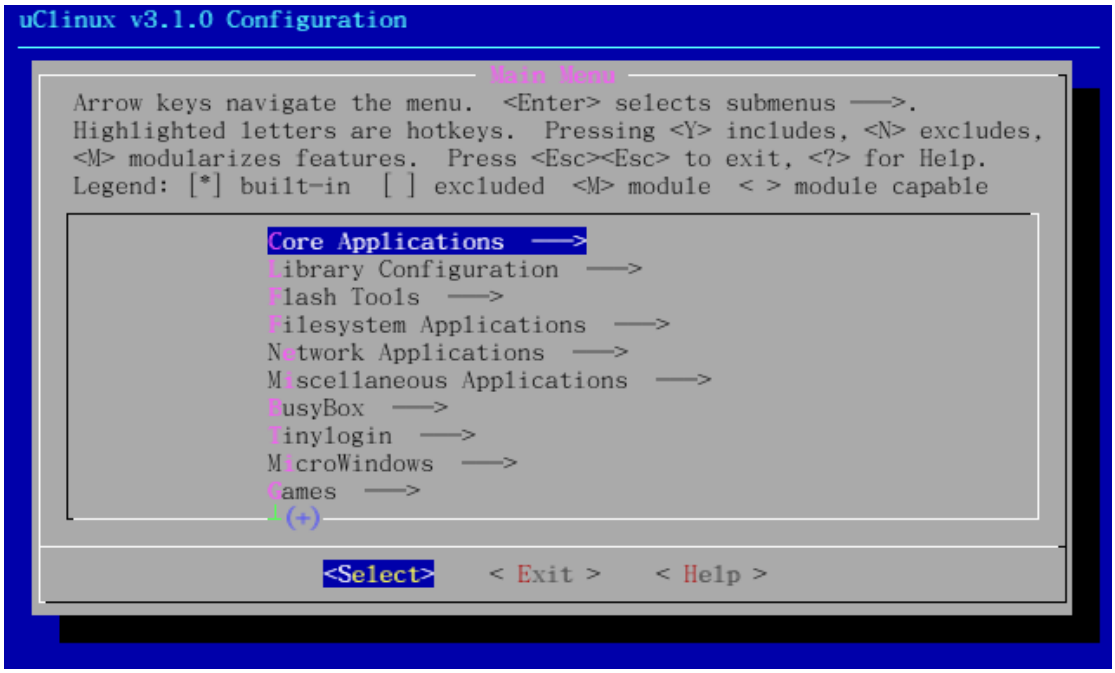


图 10.5 用户程序配置界面

配置完毕后，选择 Exit，在对话提示选择保存。

4：建立源代码文件依赖关系（第一次必须执行）。

```
make dep
```

5：编译内核。

```
make
```

这两步的时间都比较长。编译结束，会在μCLinux 安装目录下建立 romfs 和 images 两个目录。其中 romfs 为 romfs 文件系统的临时存放目录，里面有很多实用程序；images 为生成的目标代码目录，其中 ronfs.img 为 romfs 文件系统的映象文件，image.bin 为文件系统和内核代码的目标文件。在 linux-2.4.x 目录下会有一个 linux 文件，是编译好的内核文件，为 elf 格式。这些文件可用于进行μCLinux 内核实验。

至此，宿主机的交叉编译环境已经建立完成。

10.3 下载μCLinux 内核到开发板

下载 ZLG_BOOT 需要在 Windows 下进行，并且需要安装 ADS1.2 软件；下载 μCLinux 内核以及文件系统等操作可在 Windows 下进行，亦可在 Linux 下进行。

10.3.1 文件结构

光盘附带的用于下载到开发板的文件和目录如表 10.2 所示。

表 10.2 文件和结构

文件（夹）	描述	说明
bin\	文件夹	
Rtl8019	网络接口芯片 RTL8019 驱动	在 rc 文件中调用
boot\		
init.img	RAM 盘映象文件	
K9f2808.bin	NAND FLASH K9F2808 驱动	
linux.bin	μCLinux 内核文件	
zlg_boot.ini	ZLG_BOOT 管理文件	
etc\	文件夹	
rc	脚本文件	
\		
zlg_ftp.ini	设定开发板 FTP 下载的 IP、MacID 等地址	

下面对几个可编辑文件进行详细描述。

1）boot\zlg_boot.ini

ZLG_BOOT 通过 zlg_boot.ini 文件进行文件管理和加载。Zlg_boot.ini 是一个文本文件，可以进行根据需要进行编辑。它的每一行为一条命令，zlg_boot.ini 顺序执行这些命令。对于不能识别的命令，ZLG_BOOT 将忽略。典型的 zlg_boot.ini 内容如程序清单 10.1 所示。

程序清单 10.1 zlg_boot.ini 文件内容

```
#run uCLinux for SmartARM2200                          （1）
load 0x81008000 a:\boot\linux.bin                      （2）
load 0x81700000 a:\boot\init.img                       （3）
load 0x40003000 a:\boot\k9f2808.bin                    （4）
```

```
SFR 0xE002C000 0x00050005                                    （5）
#PINSEL0=0x00050005, 使用 UART0 和 UART0                     （6）
set PC=0x81008000                                            （7）
run                                                          （8）
```

其中：

（1）为 ZLG_BOOT 的启动菜单选项，参考图 10.31；

（2）将 a:\boot\linux.bin（二进制格式）加载到 0x81008000 处；

（3）将 a:\boot\init.img（二进制格式）加载到 0x81700000 处；

（4）将 a:\boot\k9f2808.bin（二进制格式）加载到 0x40003000 处；

（5）命令，向 0xE002C000 单元写入 0x00050005；

（6）对（5）的注释；

（7）设置寄存器 PC 的值为 0x81008000，也就是开始执行的地址；

（8）将所有寄存器复制到物理寄存器（包括 PC），实际就是执行代码了；

注意：

1）所有命令项格开始，命令不区分大小写；

2）以“换行”（0x0d）、“回车”（0x0a）结束；

3）数字以 16 进制表示，加“0x”前缀；

4）不能识别的命令不被执行，不正确的格式可能导致执行错误；

5）run 命令之后的命令不会被执行；

6）没有 run 命令将会导致程序复位；

7）不允许存在空行。

2）zlg_ftp.ini

当使用 FTP 和开发板相连的时候，开发板使用 zlg_ftp.ini 所设定的 IP 地址等信息，在图 10.31 的界面中，具体体现为用户 IP。开发板默认 IP 为 192.168.15.115。zlg_ftp.ini 文件内容如程序清单 10.2 所示。

程序清单 10.2 zlg_ftp.ini 文件内容

```
IP=192.168.15.15                                             （1）
GateWay=192.168.0.1                                          （2）
IPMark=255.255.0.0                                           （3）
MacID=0x52,0x54,0x4f,0x19,0xf7,0x73                          （4）
```

其中：

（1）为用于 FTP 下载时的用户 IP；

（2）FTP 下载时的用户设定网关；

（3）FTP 下载时的用户设定子网掩码；

（4）FTP 下载时的用户设定网卡 MacID。

用户可根据需要对以上设置进行修改。其中，MacID 地址的第 1 个字节必须为偶数。

3）etc\rc

μCLinux 启动后，将会运行 rc 文件。rc 是文本文件，里面每行是一个命令，μCLinux 将会顺序执行该文件。典型的 rc 文件内容如程序清单 10.3 所示。

程序清单 10.3 rc 文件内容

```
insmod /usr/bin/rtl8019                          （1）
ifconfig eth0 192.168.15.15                      （2）
ifconfig eth0 netmask 255.255.0.0                （3）
ifconfig eth0 down                               （4）
ifconfig eth0 hw ether 52:54:4f:19:f7:73         （5）
ifconfig eth0 up                                 （6）
route add default gw 192.168.0.254               （7）
```

μCLinux 启动后：

（1）加载 bin 目录下的 rtl8019 驱动程序；

（2）设定开发板运行μCLinux 的 IP 地址；

（3）设定子网掩码；

（4）关闭网卡；

（5）设置网卡 MacID 地址；

（6）打开网卡；

（7）设定网关信息。

用户需要对此文件进行修改，但是注意：MacID 地址的第 1 字节必须为偶数。如果用户需要μCLinux 启动后自动运行某些命令或者程序，可修改 rc 文件，如后面会提到的自动进行 NFS 连接等。如果在 rc 文件末尾添加命令：

```
portmap&
mount -t nfs 192.168.41.42:/home/armwork /mnt -o nolock
```

则μCLinux 启动后自动与 IP 地址为 192.168.41.42 的主机进行 NFS 连接。

10.3.2 下载 ZLG_BOOT

出厂的开发板已经下载了完整的 Boot Loader 程序和 μCLinux 内核以及文件系统等程序。如果用户在使用过程中没有擦除 NOR FLASH（SST39VF160/1601），则不需要进行下载 Boot Loader 的下载工作。

SmartARM2200 主芯片为 LPC2210，LPC2210 芯片芯片没有内部 Flash，无法进行 ISP 下载，需要额外的 DownLoad 程序。SmartARM2200 提供了 2 种方式用于下载 ZLG_BOOT：串口下载和以太网下载，串口下载较慢，以太网下载较快，用户可以根据自身情况进行选择。下面分别对这两种方式进行介绍。

注意:

(1) 下载 ZLG_BOOT 需要 ADS1.2 集成开发环境，如果用户的电脑上没有安装 ADS1.2，请先安装。

(2) 光盘提供了 2 个 ADS 工程分别用于串口下载和以太网下载，分别在文件夹 FlashDown 和 FlashDown_net 中。

1. 通过串口下载

（1）建立 FlashDown 超级终端

打开超级终端，设置波特率 38400、8 位数据位、1 位停止位、无奇偶校验、无流控制，命名为 FlashDown 后保存，运行界面如图 10.6 所示。

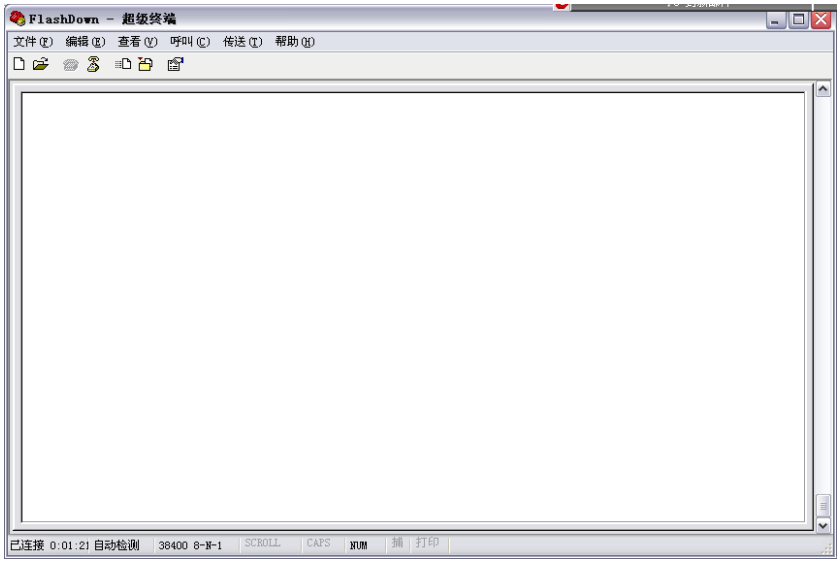


图 10.6　FlashDown 超级终端界面

（2）运行 FlashDown 工程

将 EasyJTAG 和开发板相连。打开用于串口下载的 FlashDown 工程，选择 debug 生成目标(在 RAM 中运行)，进入 AXD 调试环境运行，正确设置仿真器驱动，全速运行 FlashDown，启动下载程序，在超级终端得到如图 10.7 所示的 FlashDown 界面。

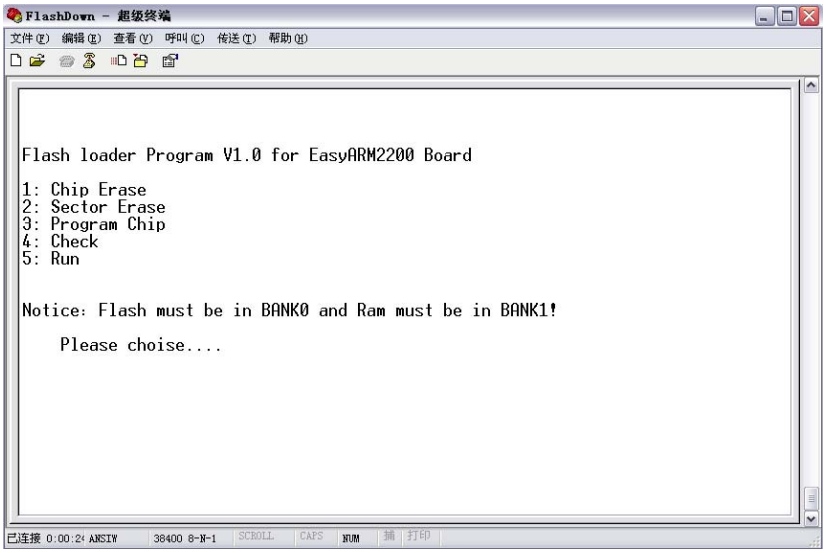


图 10.7　启动下载程序

（3）下载 zlg_boot.hex

按照提示进行 BANK 跳线设置：BANK0－FLASH，BANK1－RAM。选择 1，先对芯片进行全片擦除，擦除成功后的界面如图 10.8 所示。

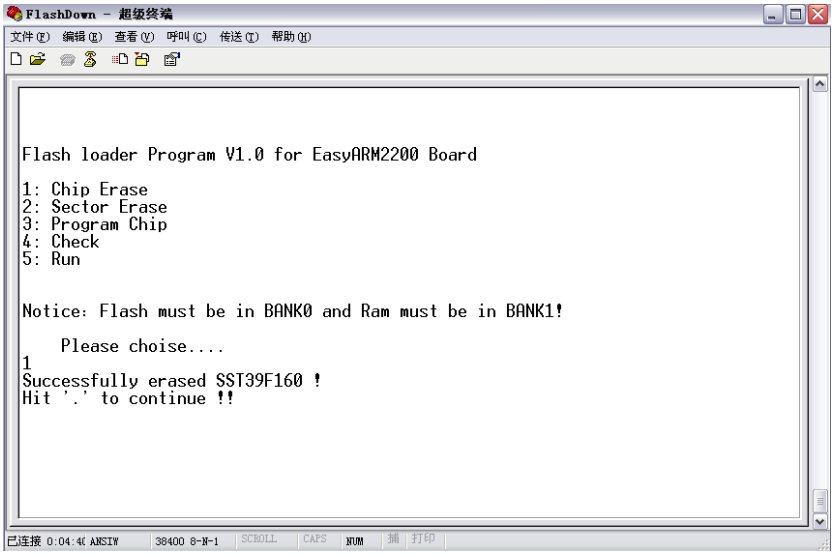


图 10.8 擦除成功

输入“.”，打开选择提示菜单，输入 3，选择编程，会有选择 hex 文件的提示，如图 10.9 所示。

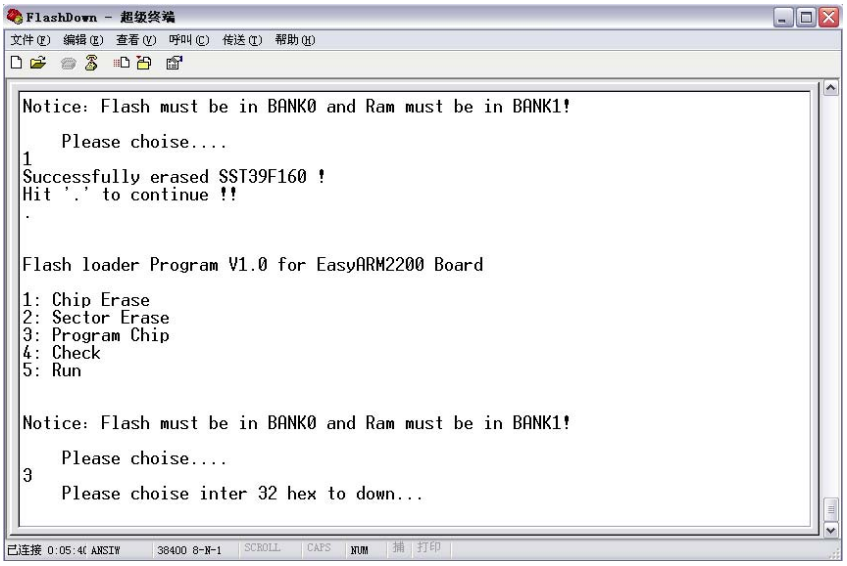


图 10.9 选择“编程”

选择“传送—>发送文本文件”，如图 10.10 所示。

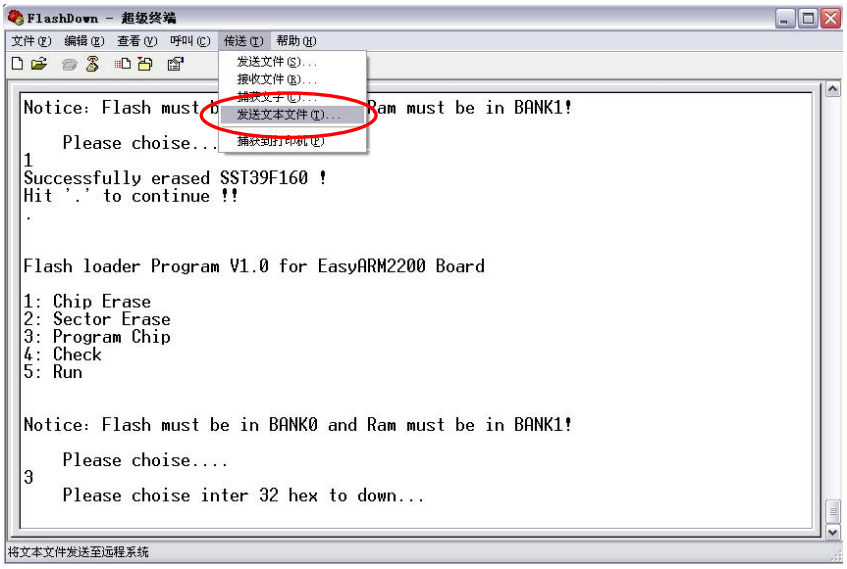


图 10.10 选择“发送文本文件”

在弹出的对话框中，指向 zlg_boot.hex 文件所在的目录，在文件类型中选择“所有文件(*.*)”，即可看到 zlg_boot.hex 文件，选择该文件，如图 10.11 所示。

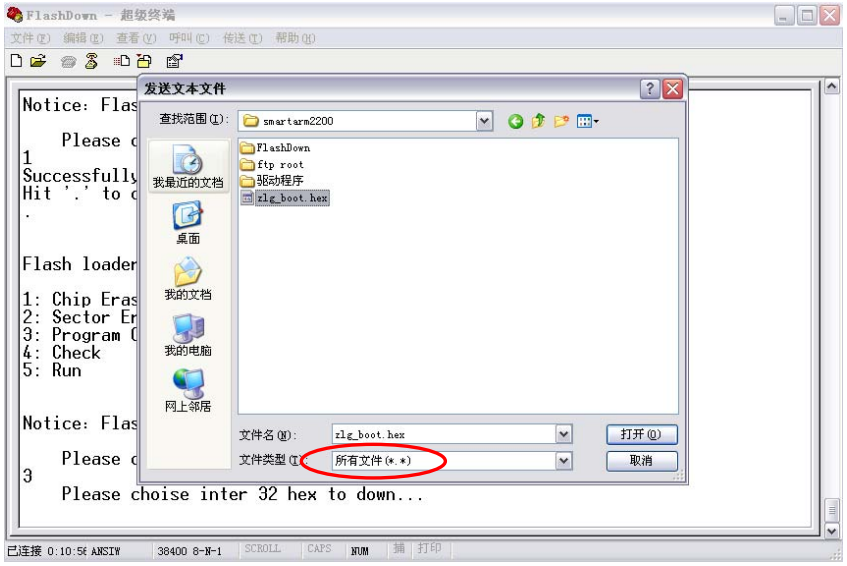


图 10.11 选择 zlg_boot.hex 文件

选择 zlg_boot.hex 文件，点击“打开”，即开始传送，文件传送界面如图 10.12 所示。

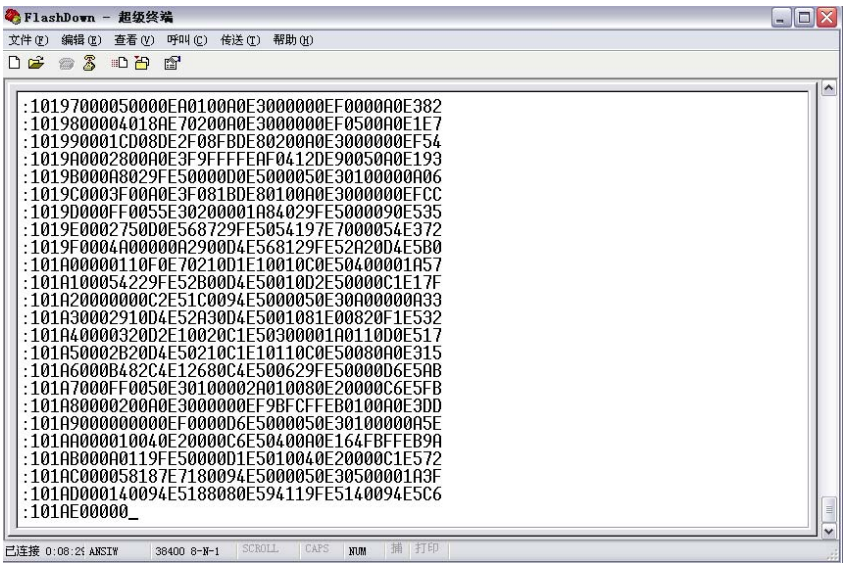


图 10.12 文件传送界面

一般情况下，整个下载过程大概需要 20 分钟左右。下载成功后的界面如图 10.13 所示。

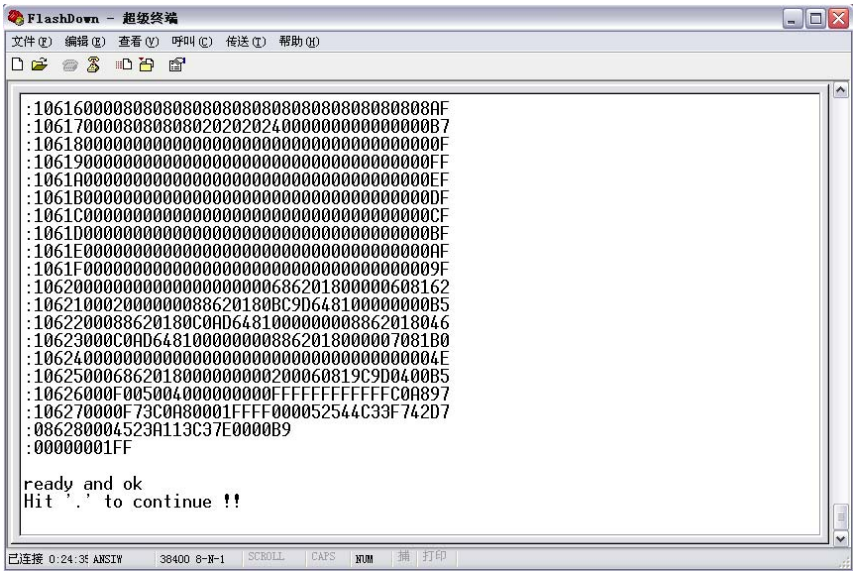


图 10.13 文件传送成功

此时，可以输入“.”打开选择提示菜单，进行校验等其它操作。

2. 通过以太网下载

通过以太网下载的操作和通过串口下载的操作方法很类似，因而在这里仅重点介绍以太网下载不同的地方。

（1）运行 FlashDown 工程

将 EasyJTAG 和开发板相连。打开 FlashDown_net 文件夹下的 FlashDown 工程，选择 DebugInExtRam。设置跳线 BANK0－Flash，BANK1－RAM，点击 Debug 按钮，进入 AXD 调试环境，全速运行程序。

注意：该工程基于μC/OS-II 操作系统，但工程中没有μC/OS-II 源代码，用户需将

μC/OS-II V2.52 的源代码放到 source 目录下，方可进行编译调试。

（2）建立以太网下载终端

打开超级终端，建立以太网下载终端。选择使用 TCP/IP，并设置主机 IP 地址 192.168.15.25，端口号 21，如图 10.14 所示。



图 10.14 以太网下载终端设置

其中，主机 IP 地址可以在 FlashDown_net 文件夹下的 FlashDown 工程中的 main.c 文件中进行更改。

设置完毕后保存，并启动终端，将得到和图 10.7 一样的 FlashDown 界面。

注意：如果先运行了终端，再运行 FlashDown 程序，有可能出现无法打开主机端口的警告，只要再次启动终端即可。

（3）下载 ZLG_BOOT

按照“通过串口下载”中介绍的方法进行下载，不到 1 分钟，即可完成整个下载过程。

10.3.3 下载μCLinux 内核和文件系统

1. 在 Windows 下进行 FTP 下载

这里介绍在 Windows 下利用 FlashFXP 软件进行下载的方法。

（1）建立μCLinux 超级终端

打开超级终端，设置波特率 115200、8 位数据位、1 位停止位、无奇偶校验、无流控制，命名为 uCLinux 后保存。

（2）启动 ZLG_BOOT

将 SmartARM2200 开发板 CZ2（UART0）与 PC 的串口相连。按开发板复位键 RST，将得到如图 10.15 所示的界面，界面上有操作选择菜单。

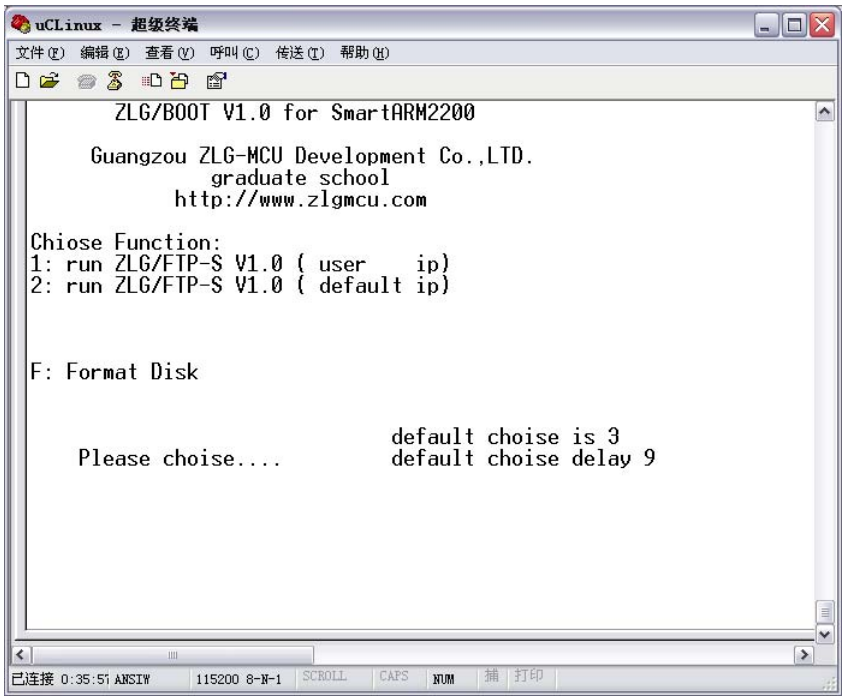


图 10.15 ZLG_BOOT 启动界面

选择 2，开发板将以默认 IP：192.168.15.115 进行 FTP 连接（如果文件系统 A 盘下没有用户的 ini 文件，则 1 和 2 等价）。

选择 F，则对 NAND Flash 进行格式化，出厂的开发板已经格式化过，因而这一步可选。

（3）运行 FlashFXP

运行 FlashFXP（如果系统已经安装了 FlashFXP，则先安装），其运行界面如图 10.16 所示。

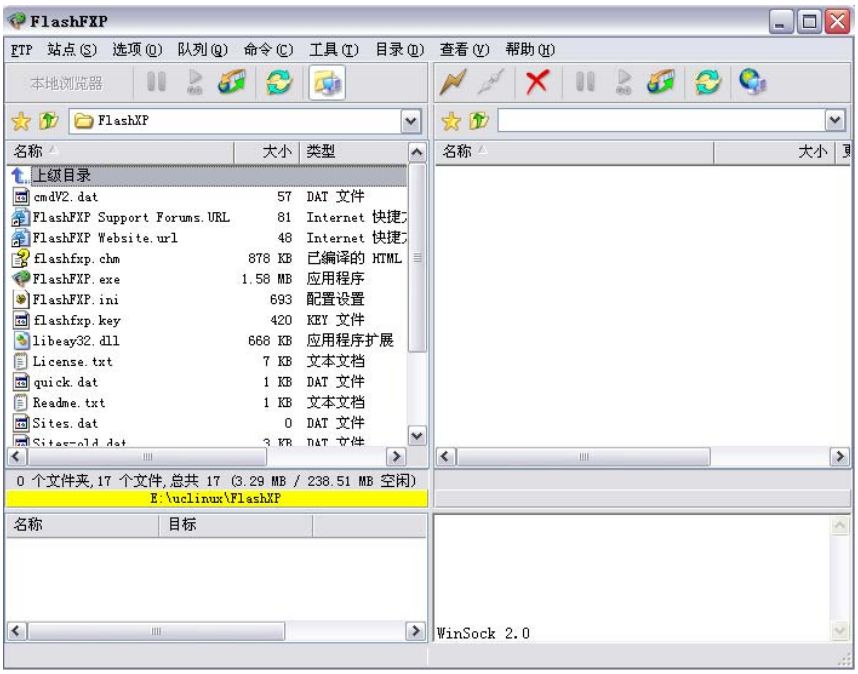


图 10.16 FlashFXP 运行界面

执行“File”—>“快速连接”，在弹出的窗口的服务器或 URL 栏输入开发板的 IP 地址，

如选择默认 IP 则输入 192.168.15.115，并输入端口号 21，如图 10.17 所示。

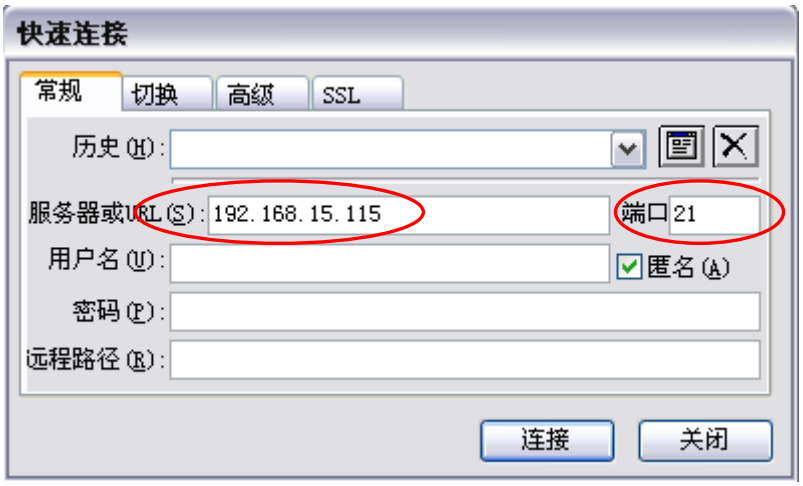


图 10.17　设置 IP 地址和端口

（4）执行 FTP 连接

通过网线将开发板接入局域网，或者通过网络交叉线直接将 SmartARM2200 和电脑相连。点击“连接”，成功后将得到类似于图 10.18 的界面。

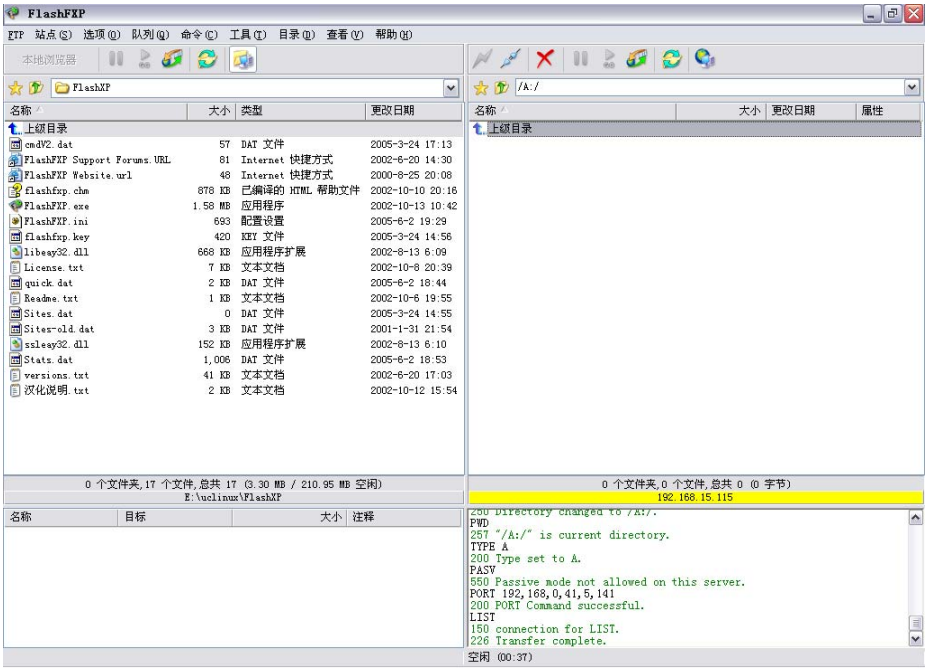


图 10.18　连接成功

选择将光盘提供的内核文件，将目录 bin\、boot\、etc\下的文件以及 zlg_ftp.ini 下载到开发板中，参考图 10.19。大约 1 分钟后传输完成，传输完成后的界面如图 10.20 所示。

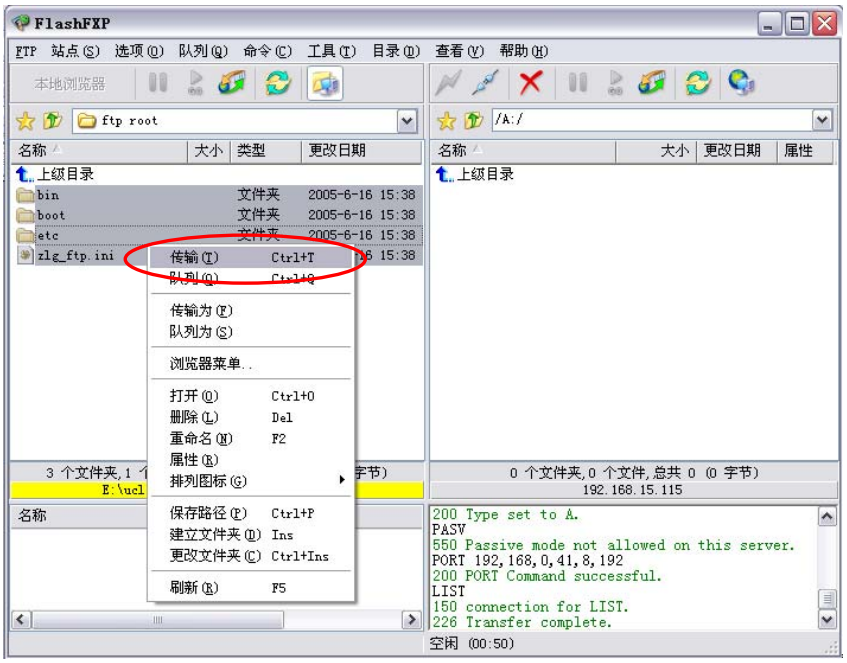


图 10.19　选择传输文件

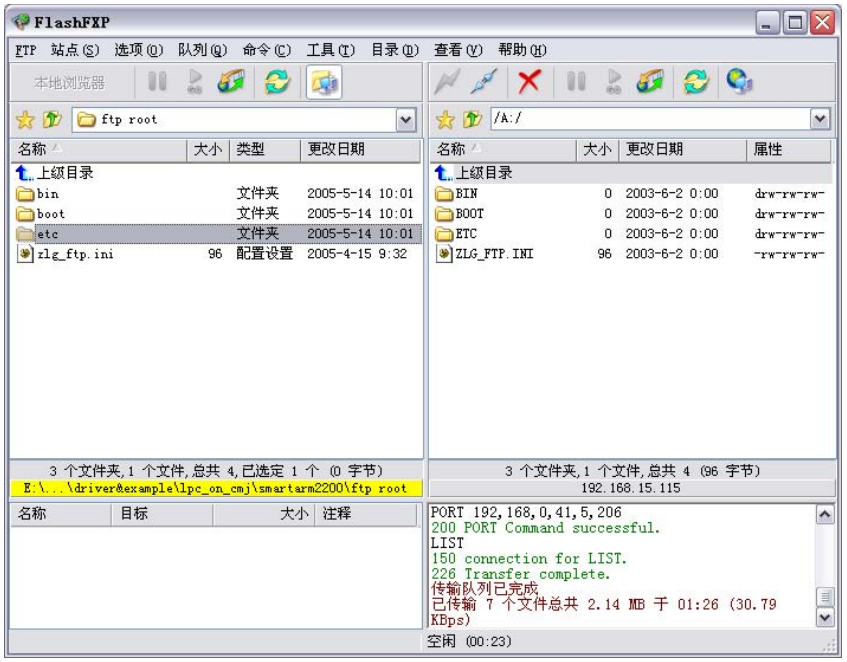


图 10.20　文件传输完成

2．　在 Linux 下进行 FTP 下载

（1）设置 minicom

在 Linux 下，打开终端（亦可在控制台进行），启动 minicom：

```
# minicom
```

成功启动后的 minicom 运行界面如图 10.21 所示。

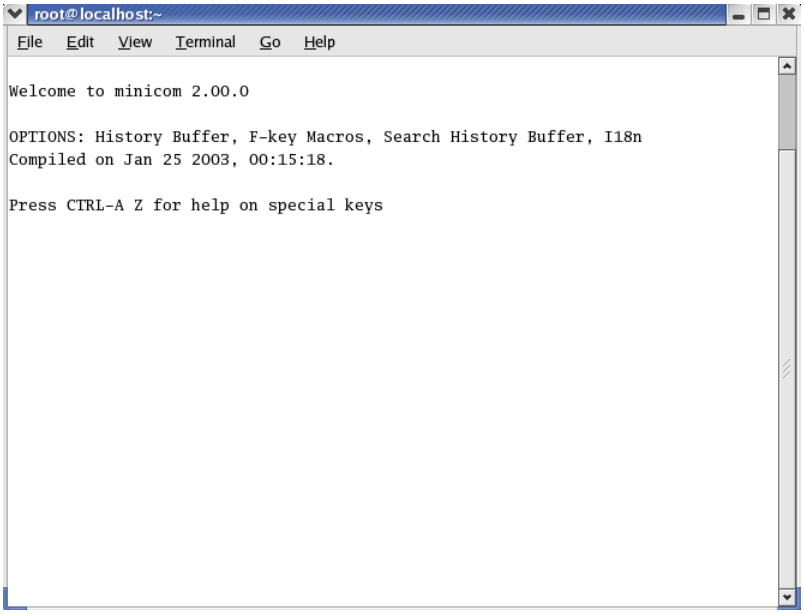


图 10.21 Minicom 界面

根据 minicom 的提示，按 CTRL+A，松开后再按 Z，进入 monicom 配置界面，如图 10.22 所示。

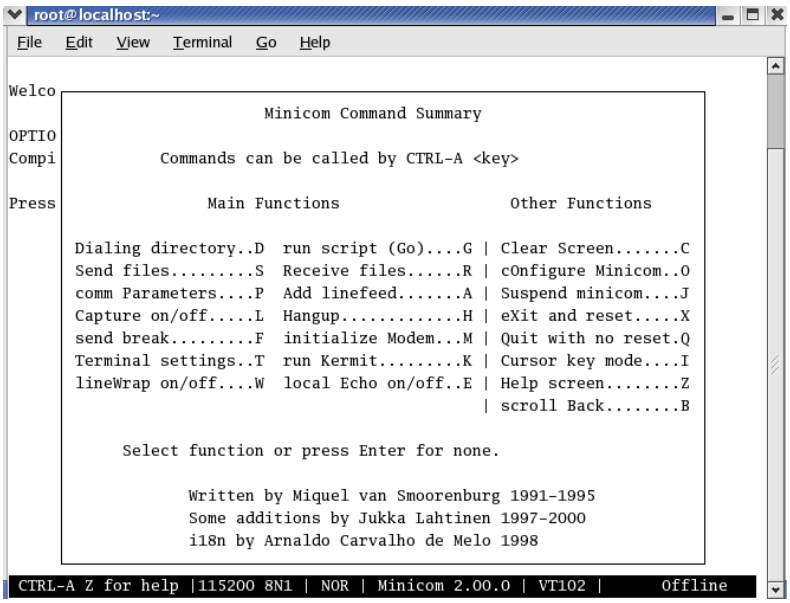


图 10.22 monicom 配置界面

根据提示，按 O，选择 cOnfigure Minicom，在弹出的界面选择 Serial Port Setup，如图 10.23 所示。

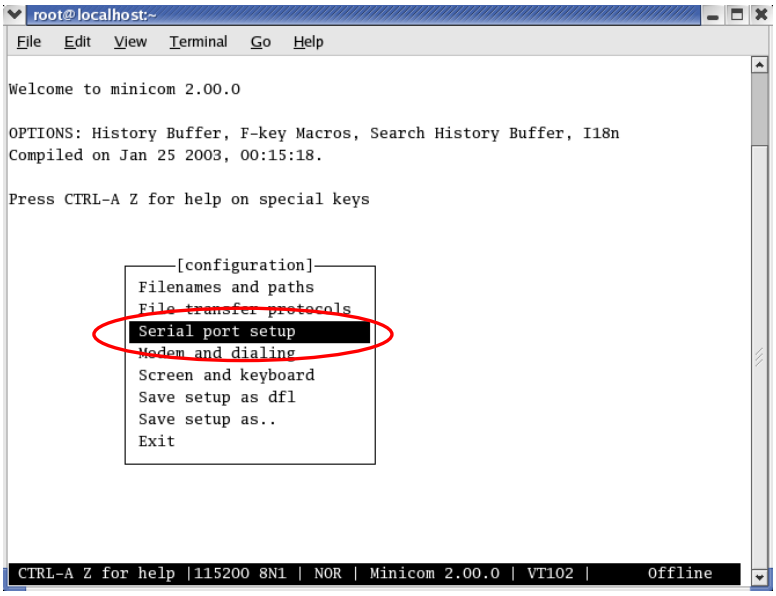


图 10.23 选择串口设置

对串口进行正确设置：波特率 115200、8 位数据位、1 位停止位、无奇偶校验、无流控制，如图 10.24 所示。

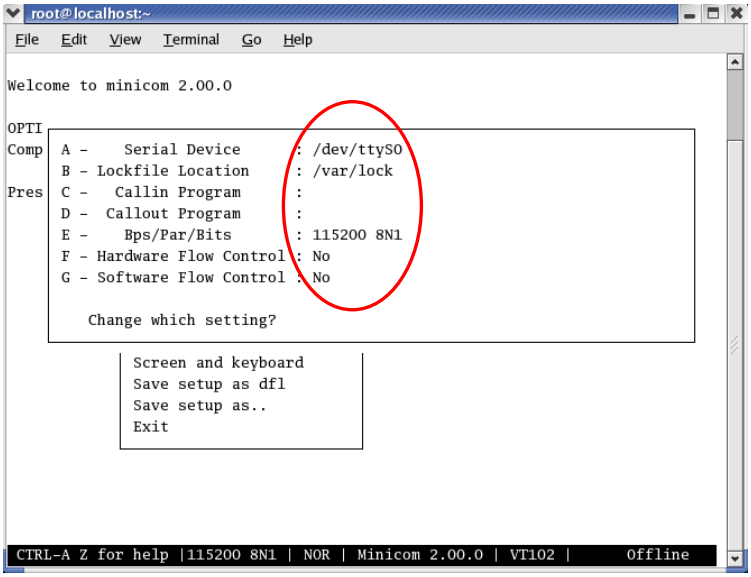


图 10.24 串口设置

设置好串口后，选择更改，在退回图 10.23 的界面，选择 Save Setup as dfl 保存，并退出。

（2）启动 ZLG_BOOT

进行正确的串口设置后，将开发板串口 0 和 PC 串口相连，给开发板上电，将进入 ZLG_BOOT 界面，如图 10.25 所示。

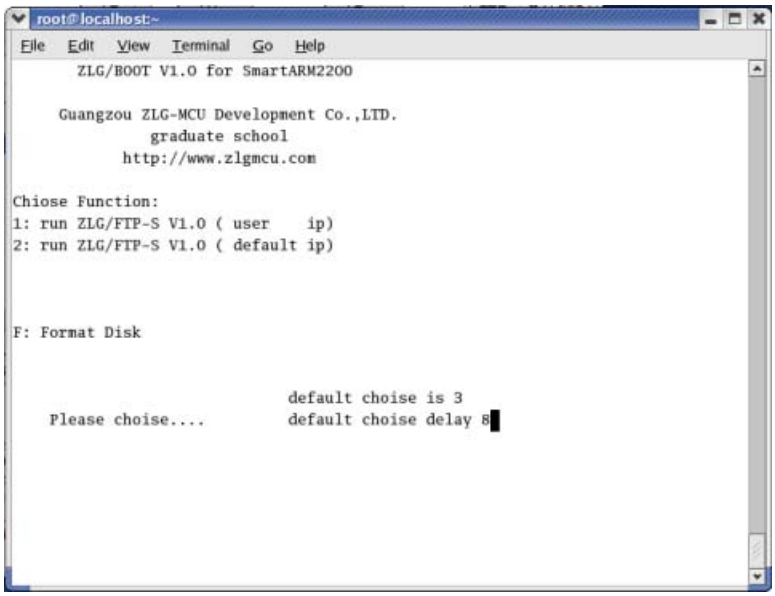


图 10.25 ZLG_BOOT 启动界面

选择 1，则使用用户 IP 进行 FTP 连接；

选择 2，使用默认 IP（192.168.15.115）进行 FTP 连接；

输入 F，则对 NAND Flash 进行格式化。

现在选择 2，使用默认 IP 进行 FTP 连接。

（3）启动 gFTP

在 Linux 下，可以通过命令行和图形界面进行 FTP 下载，使用图形界面相对直观和简单，因而在这里介绍使用 gFTP 软件进行下载的方法。

在 Linux 下新建终端，输入 gftp，启动 gFTP 软件，如图 10.26 所示。

```
# gftp
```

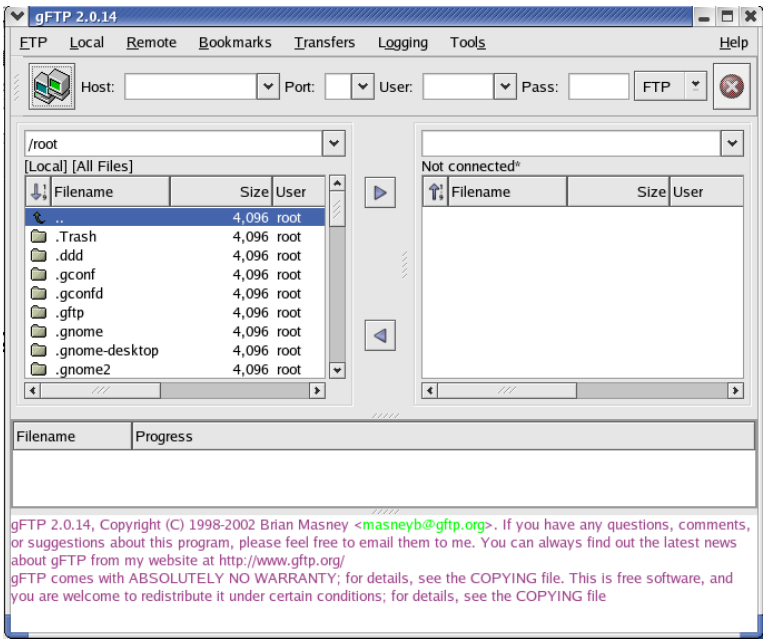


图 10.26 gFTP 软件界面

通过网线将开发板接入局域网，或者通过网络交叉线直接将 SmartARM2200 和电脑相连。在 gftp 中进行 FTP 设置：在 Host 填入正确的 IP 地址，如在图 10.25 的界面选择 2（使用默认 IP）则填入 192.168.15.115，并输入端口号 21。连接成功后的界面如图 10.27 所示。

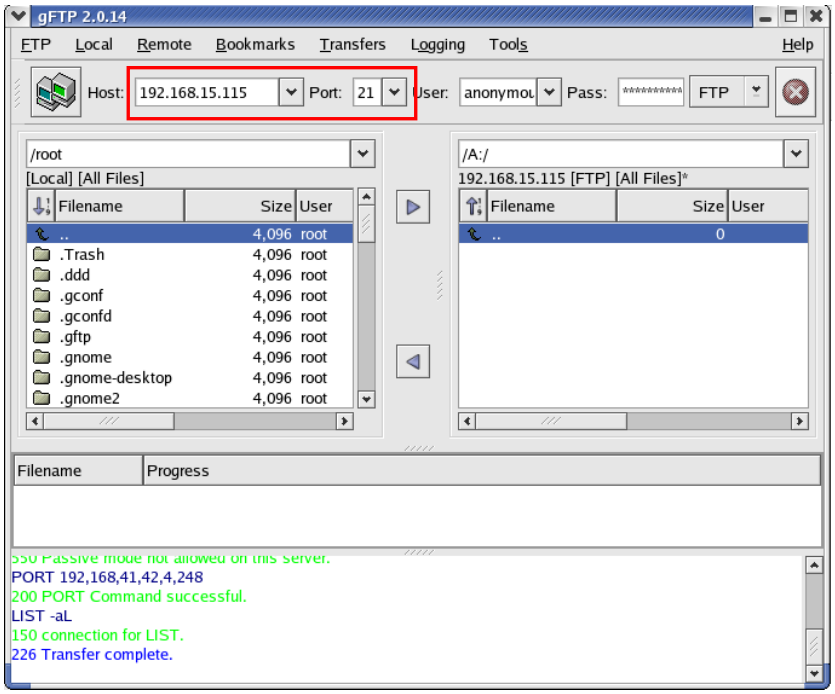


图 10.27 gFTP 连接成功

亦可在 Remote 菜单，选择 Open URL，在弹出的对话框中填入开发板 IP 地址，然后点击 Connect 按钮进行连接，如图 10.28 所示。

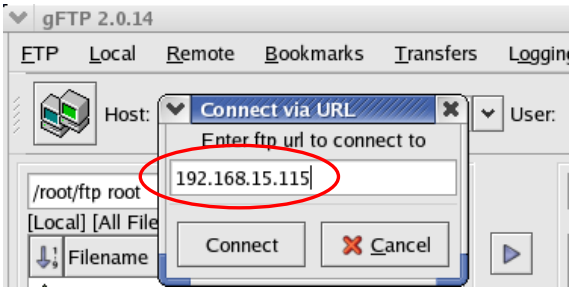


图 10.28 FTP 连接设置

进入存放内核和文件系统的目录，选择要下载的文件，然后点击传输按钮进行 FTP 上传，如图 10.29 所示。

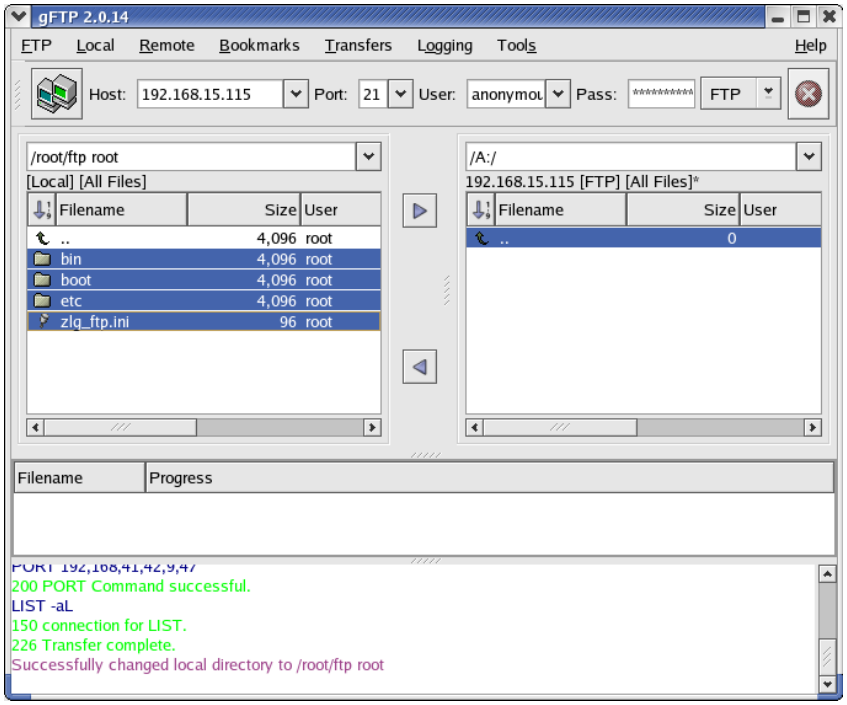


图 10.29 选择下载文件

大约一分钟后，传输完毕。传送成功后的界面如图 10.30 所示。

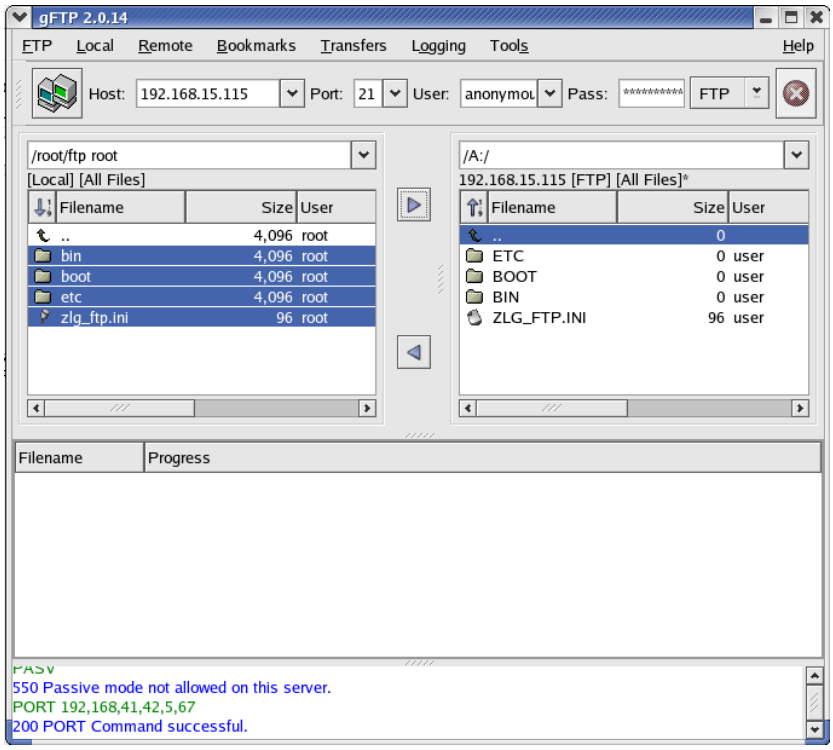


图 10.30 文件传输完成

10.4 启动μCLinux

10.4.1 在 Widows 下通过超级终端进行

将串口 0 和 PC 机串口相连，打开μCLinux 超级终端，按开发板复位键 RST，在超级终端将会得到如图 10.31 所示的选择界面。

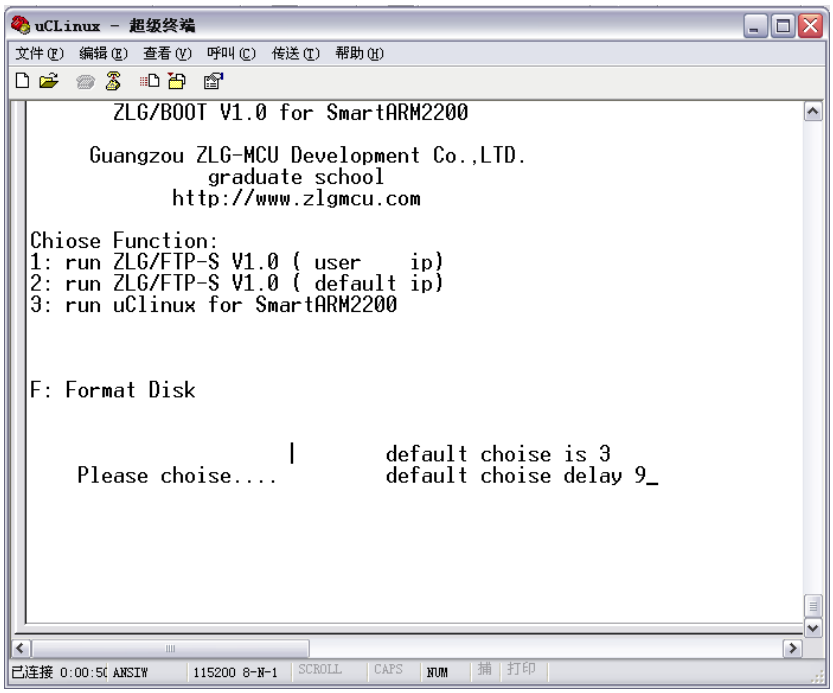


图 10.31 启动选择界面

选择 1 以用户自行设定的 IP 进行 FTP 连接，可用于下载文件。

选择 2 以出厂默认 IP（192.168.15.115）进行 FTP 连接，可用于下载文件。

选择 3，启动μCLinux。

在启动默认选项 3 之前，有 9 秒钟的时间供用户选择，如果在这段时间内用户没有输入，程序将自动启动μCLinux。μCLinux 启动成功的界面如图 10.32 所示。

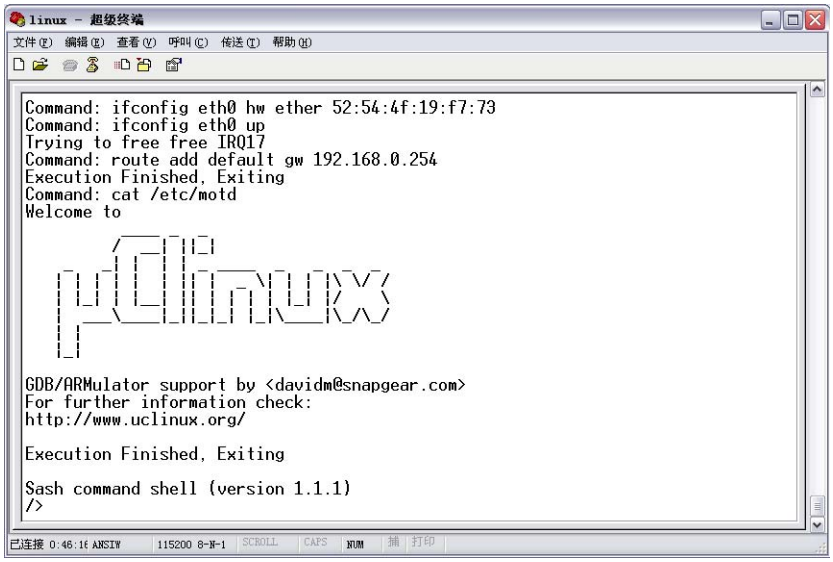


图 10.32 μCLinux 界面

在命令提示符“/>”下，可输入各种μCLinux 的命令。如用户可以输入查看命令 ls，查看一下开发板所建立的文件系统大致结构。

```
Sash command shell (version 1.1.1)
/> ls
```

```
bin
dev
etc
home
lib
mnt
proc
sbin
tmp
usr
var
/>
```

10.4.2　在 Linux 下通过 Minicom 进行

将开发板的串口 0 和 minicom 所设定的串口 ttyS0（对应于 COM1）或者 ttyS1（COM2）相连，按复位键，启动开发板，将得到如图 10.33 所示的界面。

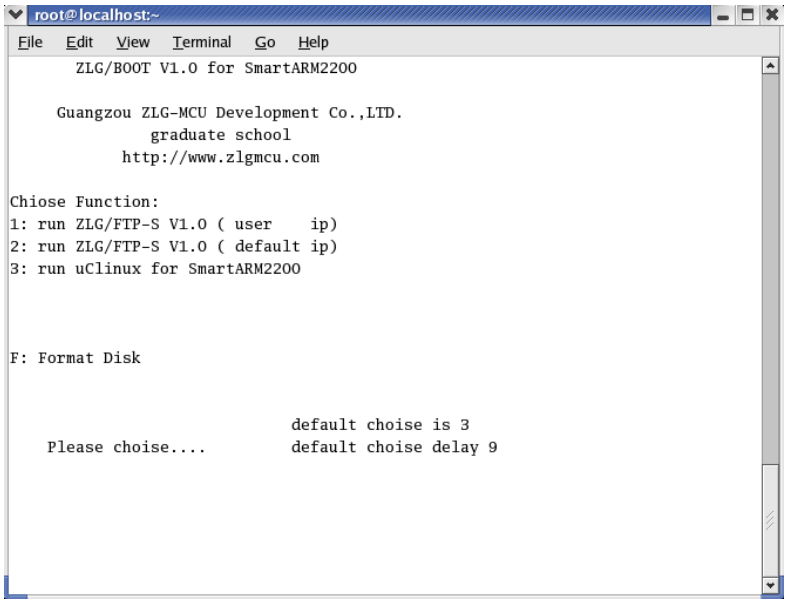


图 10.33　启动界面

选择 3，启动μCLinux。

在启动默认选项 3 之前，有 9 秒钟的时间供用户选择，如果在这段时间内用户没有输入，程序将选择 3。

μCLinux 成功启动后，将会得到如图 10.34 所示的界面。

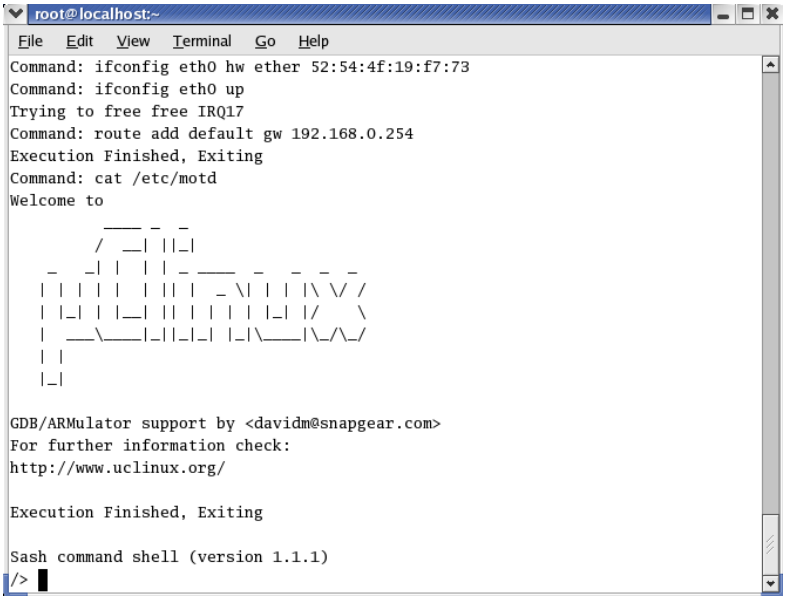


图 10.34 μCLinux 启动界面

在命令提示符“/>”下输入μCLinux 支持的命令，即可对其进行操作。

10.5 NFS 设置

该节重点介绍使用 NFS(Network FileSystem)调试μCLinux 应用程序的方法。

要想进行 NFS 调试，需要在宿主机启动 NFS 服务以及主机进行一些设置。

（1）网卡设置

从 RedHat Linux 开始菜单，选择“系统设置”—>“网络”，打开网络配置界面，如所示。

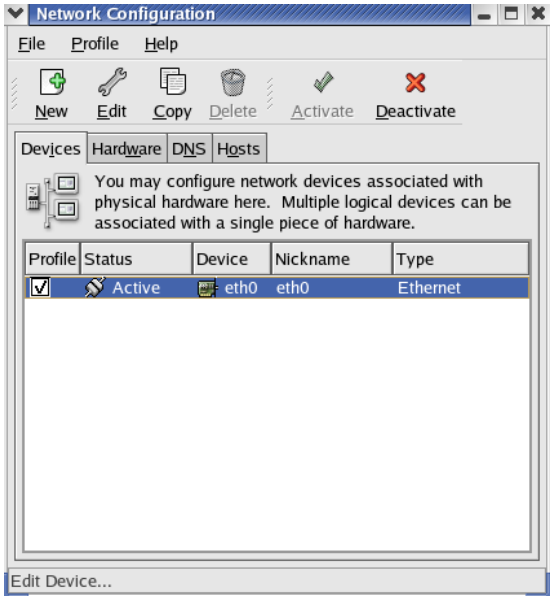


图 10.35 RedHat Linux 网卡配置界面

双击 eth0，在弹出的界面进行具体的网络设置，如 IP 地址、子网掩码和网关等信息，

如图 10.36 所示。不同局域网的设置不一样，详细的网络设置请参考 RedHat Linux 操作系统的使用教材。

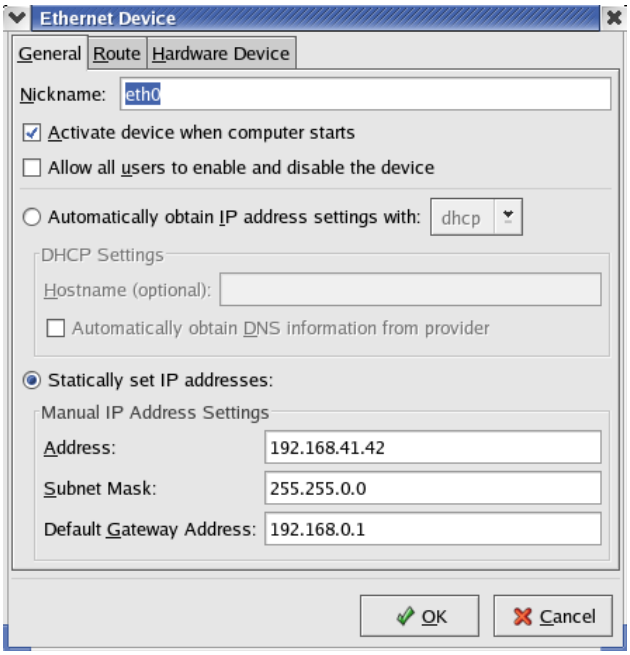


图 10.36 网络设置

（2）关闭防火墙

由于 RedHat Linux 默认安装时启动了防火墙，它将拒绝其它所有外来 IP 对它的访问。要使用 NFS 服务，必须将防火墙关闭。从 RedHat Linux 开始菜单，选择“系统设置”—>“安全级别”，打开 RedHat Linux 安全级别设置界面，将其安全级别设置为“无防火墙”，如图 10.37 所示。

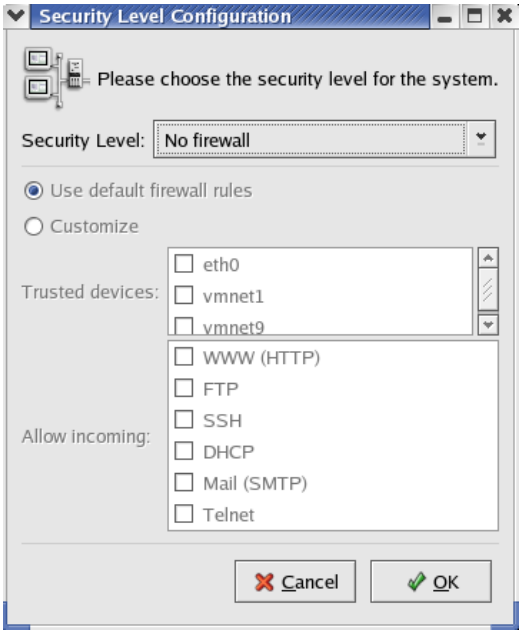


图 10.37 关闭防火墙

（3）去掉 iptables 服务

从 RedHat Linux 开始菜单，选择“系统设置”—>“服务器设置”—>“服务”，在弹出的界面中，将 iptables 前面的“√”去掉，如图 10.38 所示。

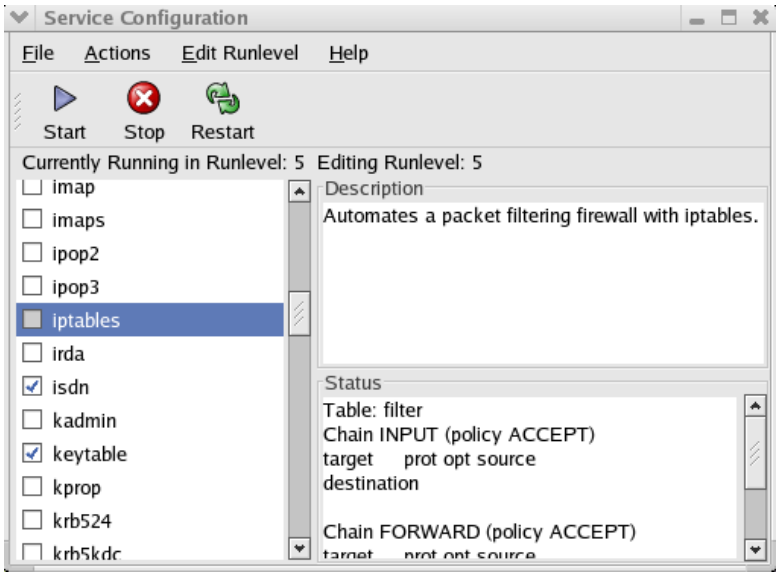


图 10.38　去掉 iptables 服务

（4）启动 NFS 服务

如果用户在安装 RedHat Linux 9.0 的时候选择了 Everything，则从开始菜单，选择“系统设置”—>“服务器设置”，在其下一级将会有“NFS 服务”的子菜单，如图 10.39 所示。选择点击，将会弹出如所示的界面。

如果不是完全安装，则可能没有“NFS 服务”子菜单，那样的话，请插入 RedHat Linux 的光盘，安装 NFS 服务。

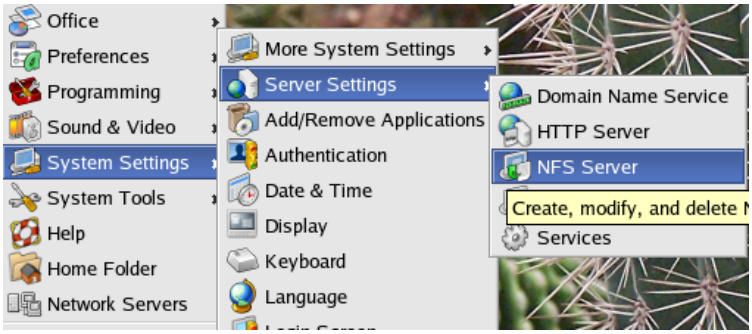


图 10.39　NFS 菜单

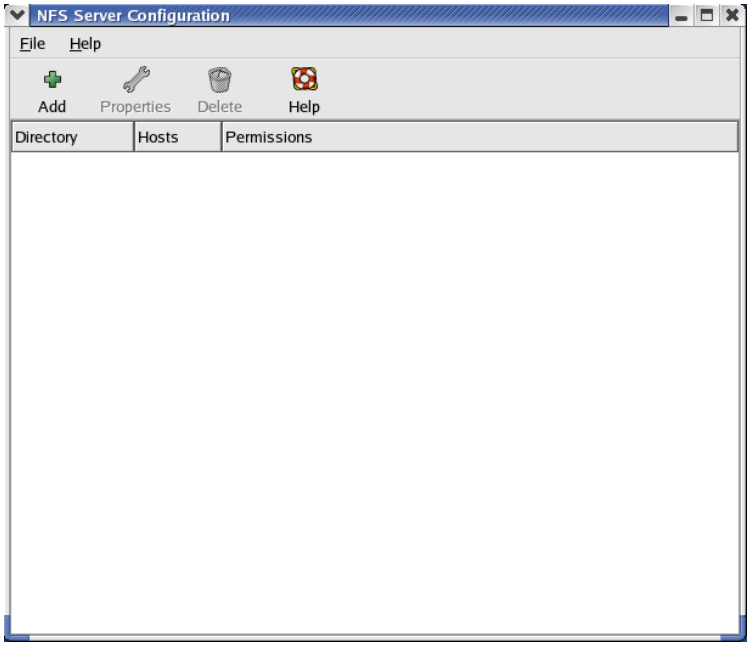


图 10.40　NFS 服务器配置

点击“Add”按钮，将会弹出如图 10.41 所示的界面。在相应栏分别填入共享目录以及可以访问该共享目录的主机的 IP。

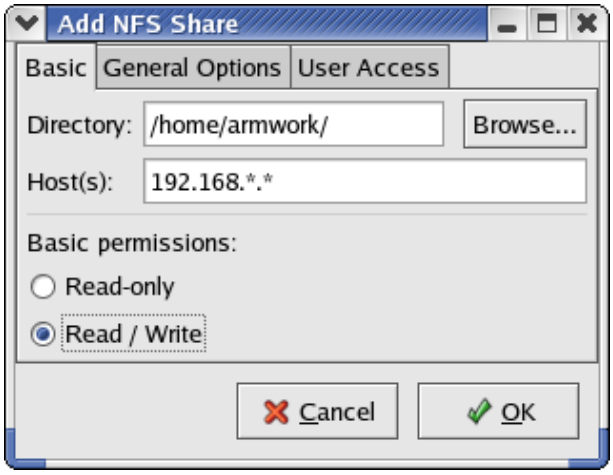


图 10.41　设置 NFS 共享目录

示例中设置主机 IP 为“192.168.*.*”，表示处于网段 192.168.X.X 的主机均可对共享目录“/home/armwork”进行访问。

至于一些权限，如“读/写”等，可以根据需要进行设置。设置后的文件保存在/etc/exports 文件中。用户亦可直接修改该文件进行共享设置。

测试 NFS 是否设置成功。可以将主机上的共享目录 mount 到自己另外一个目录下，如

```
# mount 192.168.41.42:/home/armwork /mnt
```

然后查看/mnt 目录下是否有/home/armwork 目录下的文件，如果有的话，则 NFS 启动成功。

10.6 GDB 调试

在很多情况下，用户需要对一个应用程序进行反复调试，特别是复杂的程序。调试 Linux 程序，常用的方法是进行 GDB 调试。GDB 调试既可通过网口进行，亦可通过串口进行调试。采用 GDB 方法调试，由于嵌入式系统资源有限性，一般不可能直接在目标系统上进行调试，通常采取 gdb＋gdbserver 的方式进行调试。gdbserver 在目标系统中运行，gdb 则在宿主机上运行。

要进行 GDB 调试，目标系统必须包含 gdbserver 程序，宿主机也必须安装 gdb 程序。开发板的内核已经包含了 gdbserver 程序；如果用户的宿主机没有安装 gdb 程序，请先安装。产品光盘上有安装文件（arm-elf-binutils-2.11-3.i386.rpm 和 arm-elf-gdb-5.0-1.i386.rpm）。

进行 GDB 调试，首先要在目标系统上启动 gdbserver 服务。先进入要调试的程序所在目录，如 hello，然后输入命令：

```
/hello> gdbserver :2000 hello &
```

上述命令表示在目标系统的 2000 端口开启了一个调试进程，hello 就是要调试的程序。

然后在宿主机上启动 gdb 程序。先进入要调试的程序所在的目录如 hello，然后启动调试器 arm-elf-gdb。

```
# arm-elf-gdb
```

启动 arm-elf-gdb 后，在 gdb 命令提示符状态下输入“file＋文件名”，装入调试程序的调试信息。

```
(gdb) file hello.gdb
```

如果通过网口进行 GDB 调试，输入命令，与目标系统进行连接。

```
(gdb) target remote  开发板 IP:2000
```

注意，端口号要和目标系统开启的端口号一致。

连接成功后，宿主机会输出提示信息，如：

```
remote debugging using  (开发板 IP):2000
0x16100050 in ?? ()
(gdb)
```

在目标系统串口端将会输出提示信息，如：

```
remote debugging using :2000
```

连接成功后，即可输入各种 GDB 命令如 list、run、next、step、break 等进行程序调试了。

第11章 基于μCLinux 基础实验

11.1 CAT1025 读写实验

1. 实验目的

(1) 掌握 I^2C 器件操作方法；
(2) 掌握 I^2C 驱动程序的使用方法。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：RedHat Linux 9.0 操作系统
  Windows 98/2000/XP 操作系统（可选）
  μCLinux -2.4.x 发行包和 arm-elf-gcc 交叉编译环境

3. 实验内容

向 SmartARM2200 上的 CAT1025 写入 8 字节数据，然后读回校验。

4. 实验预习要求

熟悉μCLinux 应用程序调试方法。
熟悉 CAT1025 的操作时序。
仔细阅读本书第 1 章的内容，了解 SmartARM2200 的硬件结构，注意 CAT1025 的电路。

5. 实验原理

CAT1025 的电路请参考硬件结构部分的介绍。

6. 实验步骤

（1）打开终端或者进入虚拟控制台，进入 arm 实验目录，为本实验新建工作目录 cat1025。

```
# cd /home/armwork
# mkdir cat1025
```

（2）使用自己熟悉的编辑器如 vi 建立文件 cat1025.c，编写实验代码并保存。

```
# vi cat1025.c
```

（3）将光盘提供的 Makefile 文件拷贝到当前目录下，进行修改，适合于本实验，如果编译器路径和光盘提供的不一致，还需修改编译器路径。

```
EXEC = cat1025
OBJS = cat1025.o
SRC = cat1025.c
```

（4）将光盘提供的 i2c.h 文件拷贝到 header 目录下。

（5）将光盘提供的 config.h 文件拷贝到当前目录下，修改该文件，增加文件包含信息。

```
#include "../header/i2c.h"
```

（6）编译程序，生成可执行代码 cat1025。

```
# make
```

（7）短接跳线 JP6。启动μCLinux，进行 NFS 连接，进入 I^2C 驱动所在目录，加载 I^2C

驱动。

```
/> ./loadi2c
```

（8）进入 cat1025 目录，运行实验程序。

```
/> ./cat1025
```

7. 实验参考程序

程序首先打开设备 CAT1025，如果打开失败则报错退出。成功打开后，先设置 I^2C 总线频率为 100K（最高 400K），然后往 0 地址开始处写入 8 字节数据，最后读回显示。实现程序如程序清单 11.1 所示。

程序清单 11.1 CAT1025 读写实验参考程序

```
#include "config.h"

int fd;
uint8 data_buf[32];
int main()
{
    uint8 i;
    int suba;
    uint8 snd[32] = "\0ABCDEFGHIJK";          // 第一个字节“0”为子地址

    fd = open("/dev/cat1025", O_RDWR);        // 打开设备
    if(fd == -1)
    {
        printf("\nCan't open CAT1025!\n");
        exit(-1);
    }
    /* 设置 I2C 总线频率 */
    ioctl(fd, I2C_SET_CLH, ((11059200 / 100000) + 1) / 2);
    ioctl(fd, I2C_SET_CLL, (11059200 / 100000) / 2);

    printf("write CAT1025:\n");
    for (i=1; i<9; i++)
    {
        printf(" %c ", snd[i]);               // 以字符形式显示
    }

    write(fd, snd, 9);                        // 写入 8 字节有效数据
    printf("\n");

    sleep(1);

    printf("Read CAT1025:\n");
```

```
    suba = 0x00;
    write(fd, &suba, 1 );                  // 写入子地址
    read(fd, data_buf, 8);                 // 读取数据

    for (i = 0; i < 8; i++)
    {
        printf(" %c ", data_buf[i]);       // 以字符形式显示
    }
    printf("\n");

    close(fd);
    return 0;
}
```

8． 思考

如果将要写入的数据跨页，该如何处理？

11.2 任务定时器

1． 实验目的

熟悉 Linux 定时器的基本应用。

2． 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：RedHat Linux 9.0 操作系统
  Windows 98/2000/XP 操作系统（可选）
  μCLinux -2.4.x 发行包和 arm-elf-gcc 交叉编译环境

3． 实验内容

使用 Linux 定时器，每秒钟将蜂鸣器状态取反。

4． 实验预习要求

熟悉μCLinux 应用程序调试方法。
了解 Linux 定时器操作方法。

5． 实验原理

Linux 为每个任务安排了 3 个内部定时器：

- ITIMER_REAL：实时定时器，不管进程在何种模式下运行（甚至在进程被挂起的时），它总在计数。定时到达，向进程发送 SIGALRM 信号。
- ITIMER_VIRTUAL：这个不是实时定时器，当进程在用户模式（即程序执行时）计算进程执行的时间。定时到达后向该进程发送 SIGVTALRM 信号。
- ITIMER_PROF：进程在用户模式（即程序执行时）和核心模式（即进程调度用时）均计数。定时到达产生 SIGPROF 信号。ITIMER_PROF 记录的时间比 ITIMER_VIRTUAL 多了进程调度所花的时间。

定时器在初始化是，被赋予一个初始值，随时间递减，递减至 0 后发出信号，同时恢复初始值。在任务中，我们可以一种或者全部三种定时器，但同一时刻同一类型的定时器只能使用一个。

6. 实验步骤

（1）打开终端或者进入虚拟控制台，进入 arm 实验目录，为本实验新建工作目录 systimer。

```
# cd /home/armwork
# mkdir systimer
```

（2）使用自己熟悉的编辑器如 vi 建立文件 systimer.c，编写实验代码并保存。

```
# vi systimer.c
```

（3）将光盘提供的 Makefile 文件拷贝到当前目录下，进行修改，适合于本实验，如果编译器路径和光盘提供的不一致，还需修改编译器路径。

```
EXEC = systimer
OBJS = systimer.o
SRC = systimer.c
```

（4）将光盘提供的 gpio.h 文件拷贝到 header 目录下。

（5）将光盘提供的 config.h 文件拷贝到当前目录下，修改该文件，增加文件包含信息。

```
#include "../header/gpio.h"
```

（6）编译程序，生成可执行代码 systimer。

```
# make
```

（7）短接跳线 JP7。启动μCLinux，进行 NFS 连接，进入 gpio 驱动所在目录，加载 gpio 驱动。

```
/> ./loadgpio
```

（8）进入 systimer 目录，运行实验程序。

```
/> ./systimer
```

7. 实验参考程序

实验使用 ITIMER_PROF 定时器，每秒钟产生一次定时信号，将蜂鸣器控制口取反，如程序清单 11.2 所示。

程序清单 11.2 任务定时器实验参考程序

```
#include "config.h"
#include <stdio.h>
#include <signal.h>
#include <sys/time.h>

#define BEEP    7                       // P0.7 控制蜂鸣器
uint8 ledcon = 0;

/* 信号处理函数 */
void led_info(int signo)
{
    int fd;
```

```
    printf("timeout!\n");                    // 输出提示信息

    ledcon = ~ledcon;
    fd = open("/dev/P0", O_RDONLY);
    if(fd == -1)
    {
        printf("\nCan't open P0!\n");
        exit(-1);
    }

    /* 取反蜂鸣器控制口 */
    ioctl(fd, GPIO_SET_PIN_OUT, BEEP);
    if (ledcon)     ioctl(fd, GPIO_SET_PIN, BEEP);
    else ioctl(fd, GPIO_CLR_PIN, BEEP);

    close(fd);
}

/* 信号初始化 */
void init_sigaction(void)
{
    struct sigaction act;

    act.sa_handler = led_info;
    act.sa_flags    = 0;
    sigemptyset(&act.sa_mask);
    sigaction(SIGPROF, &act, NULL);
}

/* 定时器初始化 */
void init_time(void)
{
    struct itimerval val;

    val.it_value.tv_sec = 1;                 // 定时时间 1 秒
    val.it_value.tv_usec = 0;
    val.it_interval = val.it_value;
    setitimer(ITIMER_PROF, &val, NULL);
}

int main(void)
{
    init_sigaction();
```

```
    init_time();
    while(1);
    exit(0);
}
```

8. 思考

另外的定时器如何使用？请用实验验证。

11.3 使用信号量作定时器

1. 实验目的

熟悉 Linux 信号量作为定时器的使用。

2. 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：RedHat Linux 9.0 操作系统
  Windows 98/2000/XP 操作系统（可选）
  μCLinux -2.4.x 发行包和 arm-elf-gcc 交叉编译环境

3. 实验内容

使用 Linux 的信号量作定时器，每秒钟将 LED1 状态取反。

4. 实验预习要求

熟悉μCLinux 应用程序调试方法。
了解 Linux 信号量的基本使用方法。

5. 实验原理

6. 实验步骤

（1）打开终端或者进入虚拟控制台，进入 arm 实验目录，为本实验新建工作目录 sigtimer。

```
# cd /home/armwork
# mkdir sigtimer
```

（2）使用自己熟悉的编辑器如 vi 建立文件 sigtimer.c，编写实验代码并保存。

```
# vi sigtimer.c
```

（3）将光盘提供的 Makefile 文件拷贝到当前目录下，进行修改，适合于本实验，如果编译器路径和光盘提供的不一致，还需修改编译器路径。

```
EXEC = sigtimer
OBJS = sigtimer.o
SRC = sigtimer.c
```

（4）将光盘提供的 gpio.h 文件拷贝到 header 目录下。

（5）将光盘提供的 config.h 文件拷贝到当前目录下，修改该文件，增加文件包含信息。

```
#include "../header/gpio.h"
```

（6）编译程序，生成可执行代码 sigtimer。

```
# make
```

（7）短接跳线 JP2，并将 JP12 的 P2.28 短接 LED1。启动μCLinux，进行 NFS 连接，进入 gpio 驱动所在目录，加载 gpio 驱动。

```
/> ./loadgpio
```

（8）进入 sigtimer 目录，运行实验程序。

```
/> ./sigtimer
```

7. 实验参考程序

主程序中，设置信号回调，并设置 2 秒的时限。如果定时时间到，则进入定时处理函数，先打开设备 P2，若打开失败则报错，并退出；打开成功的话，读取 LED1 控制口的设置状态，并取反 LED1 控制状态，同时重新设置定时时限为 1 秒。实现程序如程序清单 11.3 所示。

程序清单 11.3 使用信号量作定时器实验参考程序

```
#include "config.h"
#include <signal.h>            // 使用信号量必须包含的头文件

#define LED   28               // P2.28 控制 LED1
uint32 counter = 0;            // 计数器
// alarm 函数
static void sig_alarm(int sec)
{
    int fd;
    uint32 pin;

    counter++;
    printf("alarm\n");
    printf("%d\n", counter);
     // 打开设备 P2
    fd = open("/dev/P2", O_RDONLY);
    if(fd == -1)
    {
        printf("\nCan't open P2!\n");
        exit(-1);
    }
    ioctl(fd, GPIO_SET_PIN_OUT, LED);          // 设置 LED 控制口为输出状态

    pin = LED;                                 // 读取 LED 控制口的状态
    ioctl(fd, GPIO_READ_PORT, &pin);

     /* 取反 LED 控制口 */
    if (pin)
        ioctl(fd, GPIO_CLR_PIN, LED);
```

```
    else
        ioctl(fd, GPIO_SET_PIN, LED);

    alarm(1);                       // 如果要 alarm 再次发生，必须设置发生周期
    close(fd);
}

int main(void)
{
    signal(SIGALRM, sig_alarm);         // 设置信号回调
    alarm(2);                           // 设置 2 秒时限

    while(1);
    return 0;
}
```

8． 思考

信号量最主要的作用是什么？

11.4 多进程实验

1． 实验目的

熟悉 Linux 多进程应用程序编写。

2． 实验设备

- 硬件：PC 机 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：RedHat Linux 9.0 操作系统
  Windows 98/2000/XP 操作系统（可选）
  μCLinux -2.4.x 发行包和 arm-elf-gcc 交叉编译环境

3． 实验内容

使用 Linux 的信号量作定时器，每秒钟将 LED1 状态取反。

4． 实验预习要求

熟悉μCLinux 应用程序调试方法。
了解 Linux 多进程应用程序编写。

5． 实验原理

在μCLinux 中，vfork()函数取代了 fork()函数。创建进程成功后，需根据返回值判断是父进程还是子进程。

6． 实验步骤

（1）打开终端或者进入虚拟控制台，进入 arm 实验目录，为本实验新建工作目录

mprocess。

```
# cd /home/armwork
# mkdir mprocess
```

（2）使用自己熟悉的编辑器如 vi 建立文件 mprocess.c，编写实验代码并保存。

```
# vi mprocess.c
```

（3）将光盘提供的 Makefile 文件拷贝到当前目录下，进行修改，适合于本实验，如果编译器路径和光盘提供的不一致，还需修改编译器路径。

```
EXEC = mprocess
OBJS = mprocess.o
SRC = mprocess.c
```

（4）将光盘提供的 gpio.h 文件拷贝到 header 目录下。

（5）将光盘提供的 config.h 文件拷贝到当前目录下，修改该文件，增加文件包含信息。

```
#include "../header/gpio.h"
```

（6）编译程序，生成可执行代码 mprocess。

```
# make
```

（7）将跳线 JP12 的 P2.28 短接 LED1。启动μCLinux，进行 NFS 连接，进入 gpio 驱动所在目录，加载 gpio 驱动。

```
/> ./loadgpio
```

（8）进入 mprocess 目录，运行实验程序。

```
/> ./mprocess
```

7. 实验参考程序

程序先打开设备 P0 和 P2，将蜂鸣器控制口和 LED1 控制口设置为输出状态，并输出高电平。紧接着创建新进程，根据返回值判断是否创建成功，创建成功则继续判断是子进程还是父进程。在子进程中，让蜂鸣器响一声，父进程则让 LED1 闪烁。实现程序见程序清单 11.4。

程序清单 11.4 创建子进程实验参考程序

```
#include "config.h"

#define LED     28              // P2.28 控制 LED1
#define BEEP    7               // P0.7 控制  BEEP

int main(void)
{
    int fd, fd2;
    uint32 val;
    pid_t child;

    fd = open("/dev/P0", O_RDONLY);              // 打开设备 P0 口
    if(fd == -1)
    {
        printf("\nCan't open P0!\n");
        exit(-1);
```

```
}

fd2 = open("/dev/P2", O_RDONLY);              // 打开设备 P2 口
if(fd == -1)
{
    printf("\nCan't open P2!\n");
    exit(-1);
}

ioctl(fd, GPIO_SET_PIN_OUT, BEEP);
ioctl(fd2, GPIO_SET_PIN_OUT, LED);
ioctl(fd, GPIO_SET_PIN, BEEP);
ioctl(fd2, GPIO_SET_PIN, LED);

printf("\nTry to create new process.\n");

child = vfork();                    // 创建新进程

switch(child)
{
    case -1:                        // 创建失败
        perror("vfork error\n");
        exit(-1);

    case 0:                         // 成功，子进程
    {   // 子进程，蜂鸣器鸣叫
        printf("This is child, BEEP is running\n");
        ioctl(fd, GPIO_CLR_PIN, BEEP);
        sleep(1);
        ioctl(fd, GPIO_SET_PIN, BEEP);
        sleep(1);
        exit(0);
    }
    default:                        // 成功，父进程
    {
        printf("This is father, LED is lighting.\n");
        while(1)
        {   // 父进程，LED1 闪烁
            ioctl(fd2, GPIO_CLR_PIN, LED);
            sleep(1);
            ioctl(fd2, GPIO_SET_PIN, LED);
            sleep(1);
        }
    }
```

```
    }
    return 0;
}
```

11.5 WebServer 实验

1. 实验目的

通过实验掌握μCLinux 操作系统下 WebServer 的编程方法。

2. 实验设备

- 硬件：PC 机（带网卡） 一台
  SmartARM2200 教学实验开发平台 一套
- 软件：RedHat Linux 9.0 操作系统
  Windows 98/2000/XP 操作系统（可选）
  μCLinux -2.4.x 发行包和 arm-elf-gcc 交叉编译环境

3. 实验内容

在 SmartARM2200 上通过μCLinux 实现 WebServer 服务。

4. 实验预习要求

（1）查阅 Linux/Unix 相关网络编程知识，了解 WebServer 相关知识。

（2）仔细阅读本书第 1 章的内容，了解 SmartARM2200 的硬件结构，注意以太网控制电路及 JP8 跳线器的说明。

5. 实验原理

6. 实验步骤

（1）打开终端或者进入虚拟控制台，进入 arm 实验目录/home/armwork/，为本实验新建工作目录 userver。

```
# cd /home/armwork
# mkdir tcp
```

（2）使用自己熟悉的编辑器如 vi 在建立程序 server.c，编写实验代码并保存。

```
# vi server.c
```

（3）将光盘提供的 Makefile 文件拷贝到当前目录下，进行修改，适合于本实验，如果编译器路径和光盘提供的不一致，还需修改编译器路径。

```
EXEC = server
OBJS = server.o
SRC = server.c
```

（4）编译程序，生成可执行代码 server。

```
# make
```

（5）启动μCLinux，进行 NFS 连接，进入 userver 目录，运行端程序。

```
/> ./ server
```

（6）在电脑上打开网页浏览器，在其地址栏输入 SmartARM2200 的 IP 地址，进行连接，登录 SmaertARM2200。

7. 实验参考程序

程序清单 11.5 WebServer 实验参考程序

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <sys/socket.h>

#define PORT 80                                 // 通讯端口
#define BACKLOG    10
#define LENGTH      1024

char   httpweb[]={
      "HTTP/1.0 200 OK\r\n"
      "Date: Mon, 24 Nov 2005 10:26:00 GMT\r\n"
      "Server: microHttp/1.0 Zlgmcu Corporation\r\n"
      "Accept-Ranges: bytes\r\n"
      "Connection: Keep-Close\r\n"
      "Content-Type: text/html\r\n"
      "\r\n"};
char   web[]={
                  "<HTML>\r\n"
                  "<HEAD>\r\n"
                  "<TITLE>SmartARM2200 演示网页（周立功单片机）</TITLE>\r\n"
                  "<BODY aLink=green background=/100.bmp bgColor=#f1f1dd link=red\r\n"
                  "vLink=#321afd>\r\n"
                  "<H1>HELLO WELCOME TO SmartARM2200 WEBSERVER</H1>\r\n"
                  "<UL>\r\n"
                  "<LI> <A HREF=\"http://www.zlgmcu.com/\">周立功单片机网站 </A>\r\n"
                  "</BODY>\r\n"
                  "</HTML>\r\n"};
char   httpgif[]={
            "HTTP/1.0 200 OK\r\n"
            "Date: Mon, 24 Nov 2005 10:26:00 GMT\r\n"
            "Server: microHttp/1.0 Zlgmcu Corporation\r\n"
            "Accept-Ranges: bytes\r\n"
            "Connection: Keep-Close\r\n"
            "Content-Type: image/bmp\r\n"
            "\r\n"};
```

```
char     bmp[442]={
0x42,0x4d,0xb6,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x76,0x00,0x00,0x00,0x28,0x00,
0x00,0x00,0x1a,0x00,0x00,0x00,0x14,0x00,0x00,0x00,0x01,0x00,0x04,0x00,0x00,0x00,
0x00,0x00,0x40,0x01,0x00,0x00,0xc4,0x0e,0x00,0x00,0xc4,0x0e,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x00,0x00,0x80,
0x00,0x00,0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x00,0x80,0x00,0x80,0x00,0x80,0x80,
0x00,0x00,0xc0,0xc0,0xc0,0x00,0x80,0x80,0x80,0x00,0x00,0x00,0xff,0x00,0x00,0xff,
0x00,0x00,0x00,0xff,0xff,0x00,0xff,0x00,0x00,0x00,0xff,0x00,0xff,0x00,0xff,0xff,
0x00,0x00,0xff,0xff,0xff,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf6,0x66,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x6f,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0x66,0x66,0x6f,0x6f,0xff,0x66,0x66,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xf6,0xff,0xff,0x6f,0xf6,0xff,0xf6,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0x6f,0xff,0x6f,0xf6,0xff,0xf6,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xf6,0xff,0xf6,0xf6,0xff,0xff,
0x6f,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0x6f,0xf6,0xff,0x6f,0xff,
0x6f,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xf6,0x66,0x66,0xf6,0xff,0xf6,0x66,
0x6f,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x6f,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x6f,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x6f,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00
};

int main ()
{
    int sockfd;                         // 暂存套接字
    int nsockfd;                        // 暂存新套接字
    int i,num;
    int flag = 1;
    int sin_size;
    //char sdbuf[LENGTH];
    char revbuf[LENGTH];
    struct sockaddr_in addr_local;      // 本地套接口地址
    struct sockaddr_in addr_remote;     // 远程套接口地址

    /* 获取 Socket 文件描述符 */
```

```
if( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1 )
{
    printf ("ERROR: Cannot obtain Socket Despcritor\n");
    return (0);
}
else
{
    printf ("OK: Obtain Socket Despcritor sucessfully\n");
}

/* 设置本地套接口信息 */
addr_local.sin_family = AF_INET;                    // 填入协议族
addr_local.sin_port = htons(PORT);                  // 填入通讯端口
addr_local.sin_addr.s_addr  = INADDR_ANY;           // 自动填充本地地址
bzero(&(addr_local.sin_zero), 8);                   // 清除结构体其他成员

/* 把套接口绑定到本地机的某一个端口 */
if( bind(sockfd, (struct sockaddr*)&addr_local, sizeof(struct sockaddr)) == -1 )
{
    printf ("ERROR: Cannot bind Port %d\n",PORT);
    return (0);
}
else
{
    printf("OK: Bind the Port %d sucessfully\n",PORT);
}

/*  侦听远程连接 */
if(listen(sockfd,BACKLOG) == -1)
{
    printf ("ERROR: Cannot listen Port %d\n", PORT);
    return (0);
}
else
{
    printf ("OK: Listening the Port %d sucessfully\n", PORT);
}

while(1)
{
    sin_size = sizeof(struct sockaddr_in);

    /*  等待连接，并获取新的 Socket 文件描述符 */
    if ((nsockfd = accept(sockfd, (struct sockaddr *)&addr_remote, &sin_size)) == -1)
```

```
        {
            printf ("ERROR: Obtain new Socket Despcritor error\n");
            continue;
        }
        else
        {
            printf ("OK: Server has got connect from %s\n", inet_ntoa(addr_remote.sin_addr));
        }

        num = recv(nsockfd, revbuf, LENGTH, 0);

        /* 子进程 */
        if(!vfork())
        {
            printf("OK: Http web is servering.\n");

            if(revbuf[5]==' ')
            {
                send(nsockfd, httpweb, 169, 0);
                send(nsockfd, web, 395, 0);
            }
            else if(revbuf[5]=='1')
            {
                send(nsockfd, httpgif, 169, 0);
                send(nsockfd, bmp, 442, 0);
            }
        }

        close(nsockfd);
        while(waitpid(-1, NULL, WNOHANG) > 0);
    }
}
```

8. 思考

分析 WebServer 模型的通讯过程。