ARM 嵌入式系统实验指导书

目录

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。

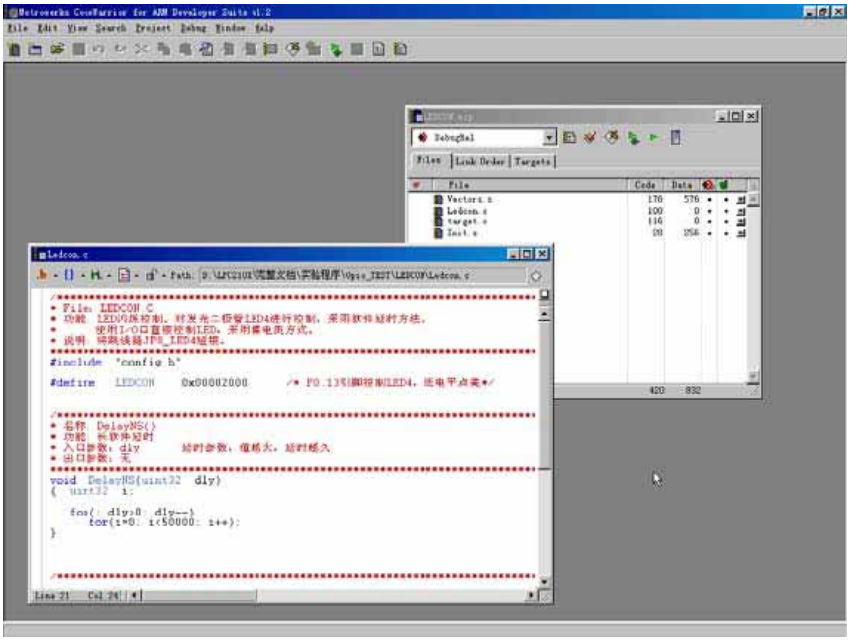

图1.1 CodeWarrior 开发环境

.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。

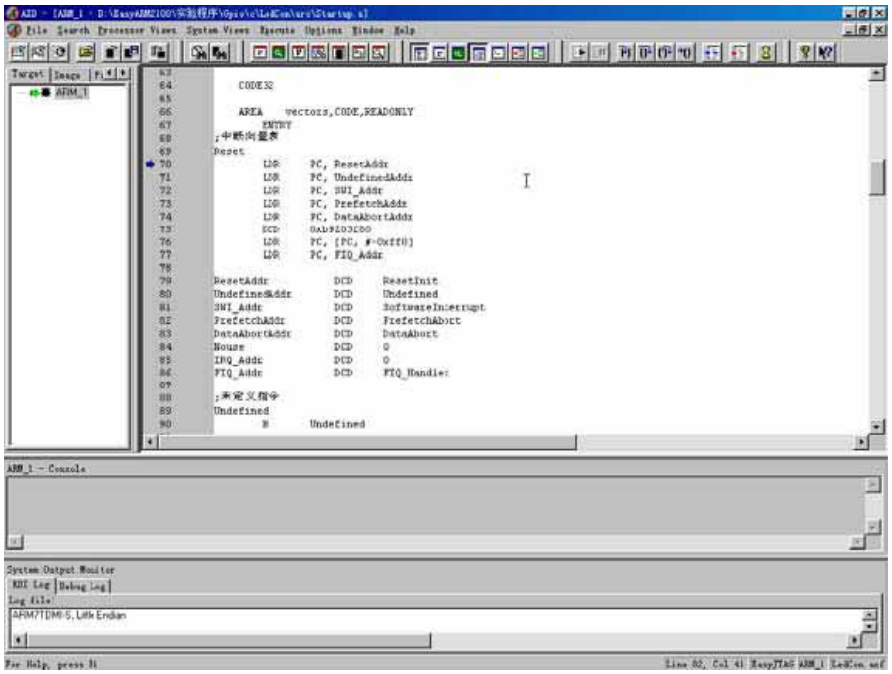

图1.2 AXD 调试器

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。

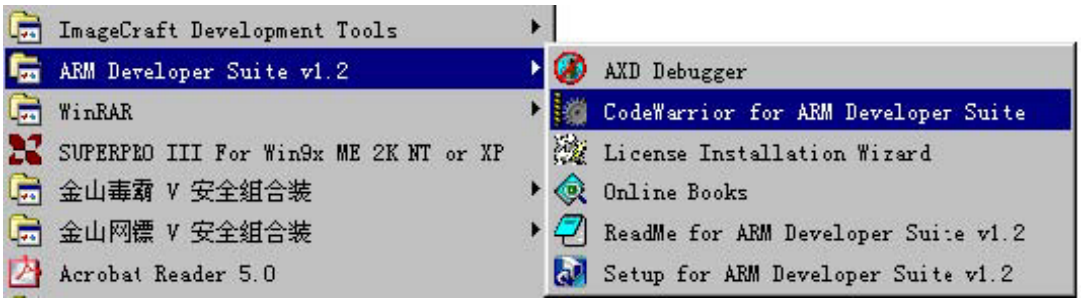

图1.3 启动ADS1.2 IDE

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

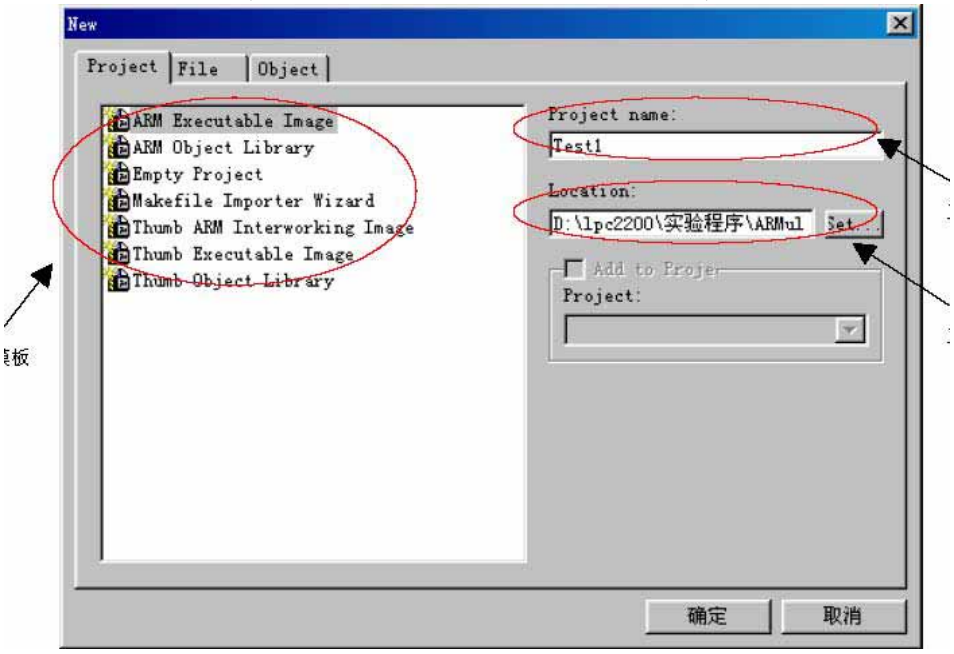

图1.4 New 对话框

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。

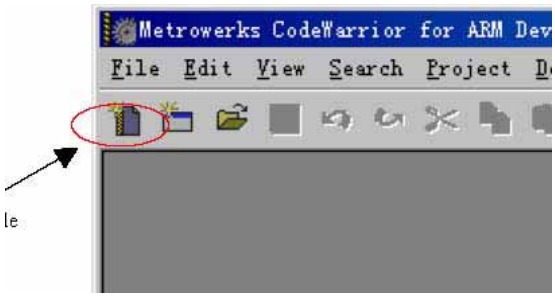

图1.5 “New Text File ”图标按钮

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。

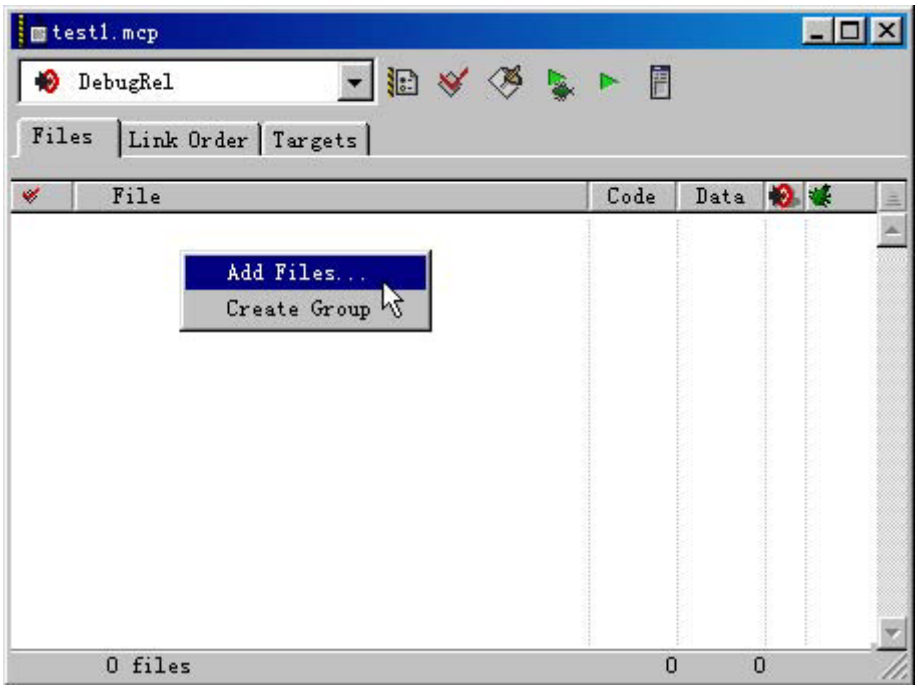

图1.6 在工程窗口中添加源文件

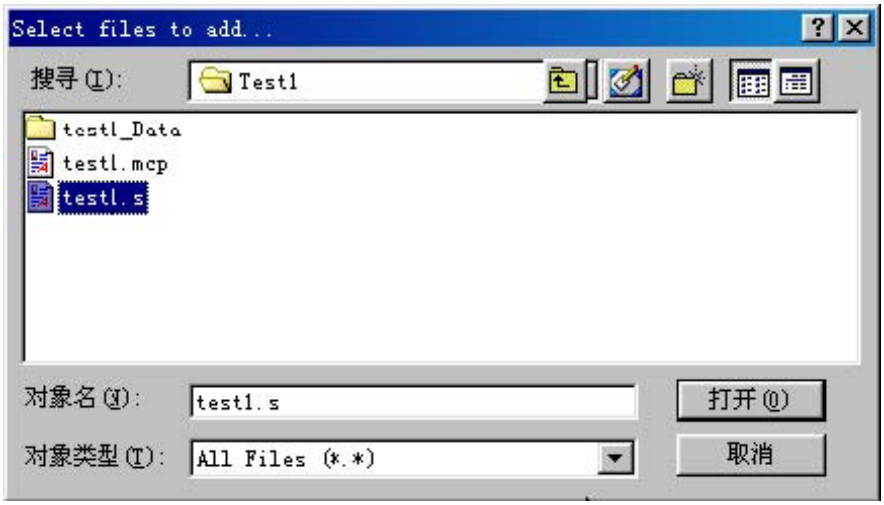

图1.7 Select files to add…对话框

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 Debug Run	编译连接**(**快捷键为**F7)**。启动**AXD** 进行调试**(**快捷键为**F5)**。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

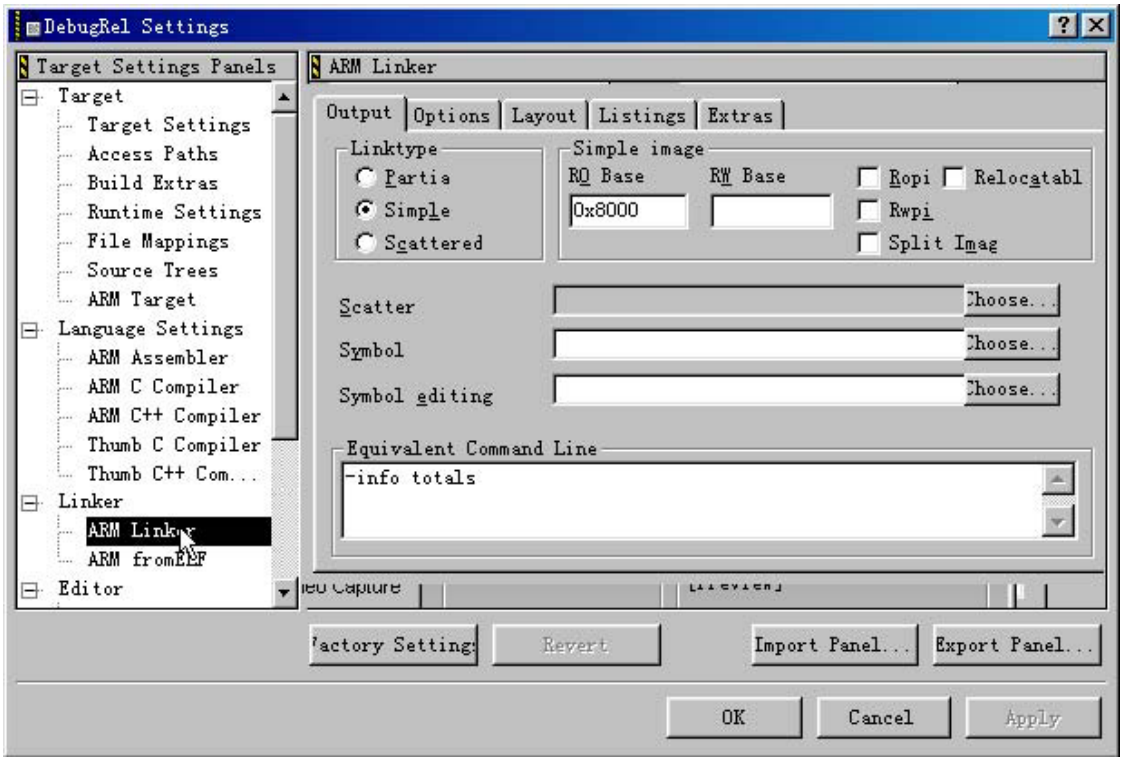

图1.9 DebugRel Settings 窗口

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。

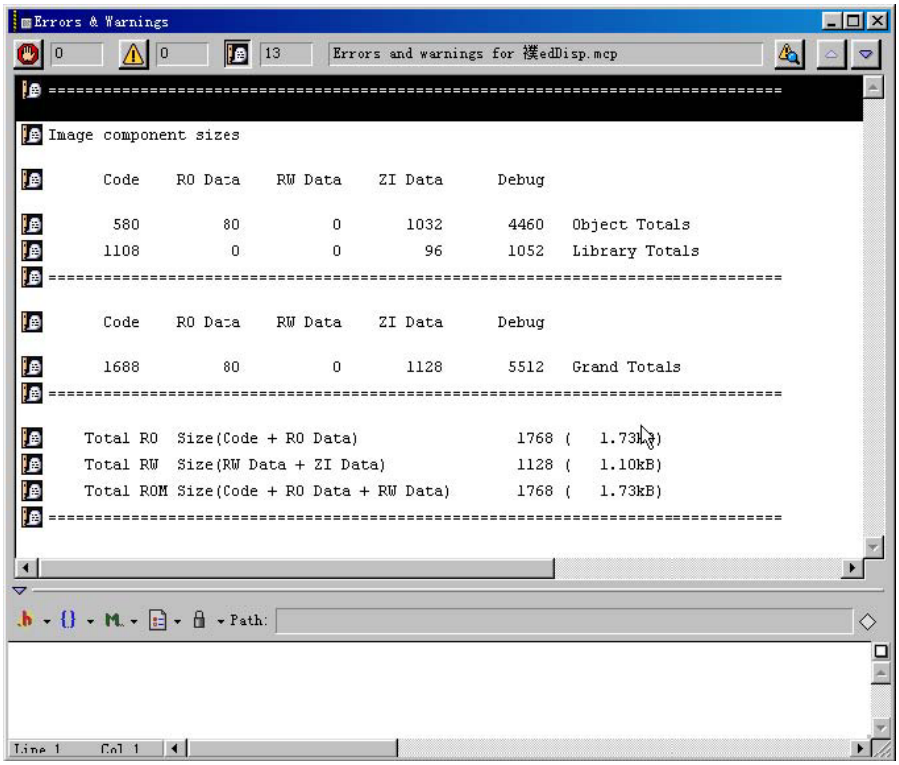

图1.10 编译连接输出窗口

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“√”。

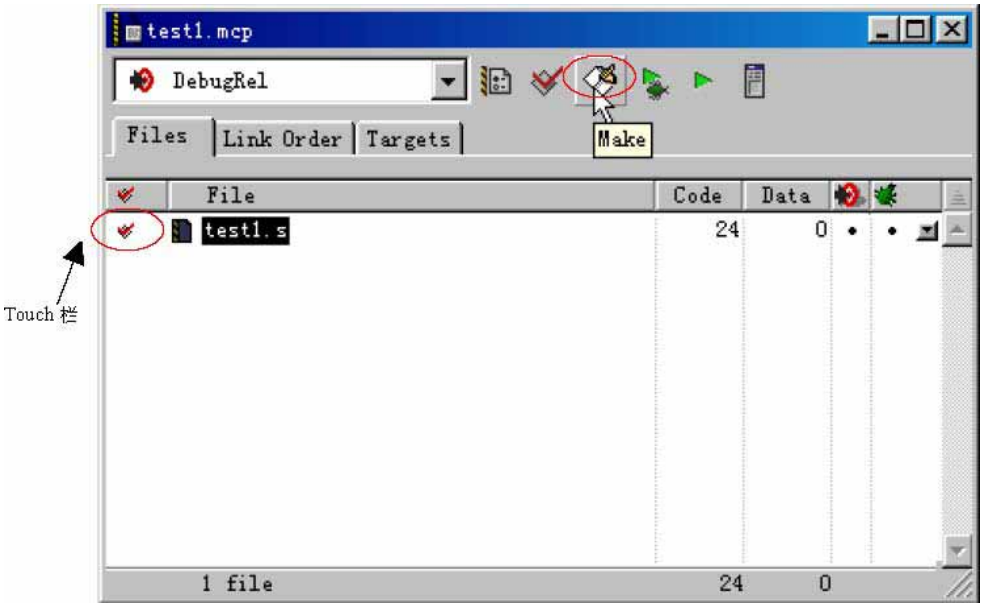

图1.11 工程窗口中Make 操作

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

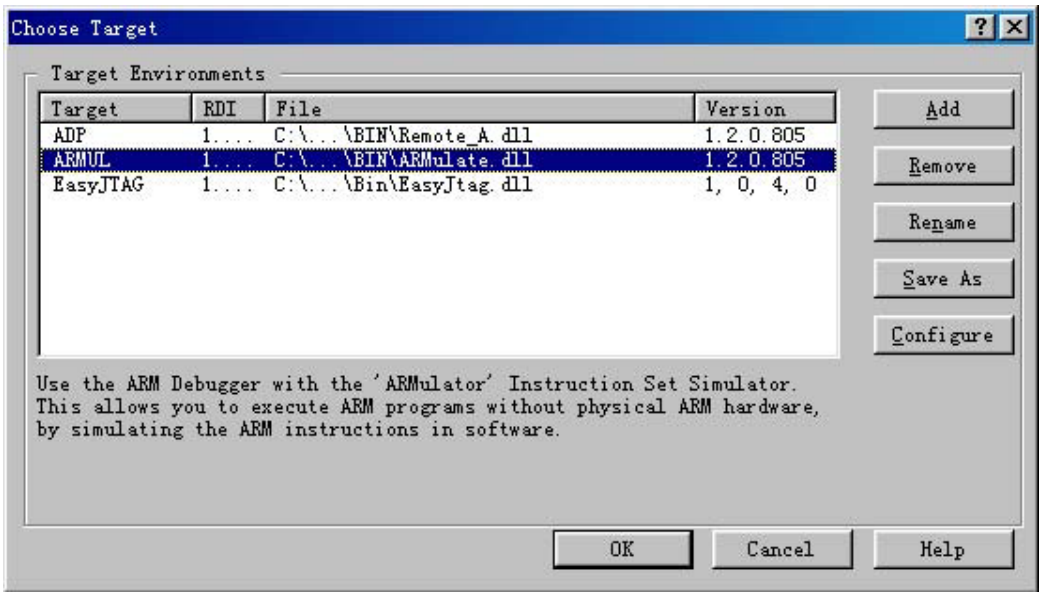

图1.12 Choose Target 窗口

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。

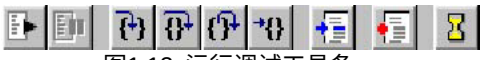

图1.13 运行调试工具条

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

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。

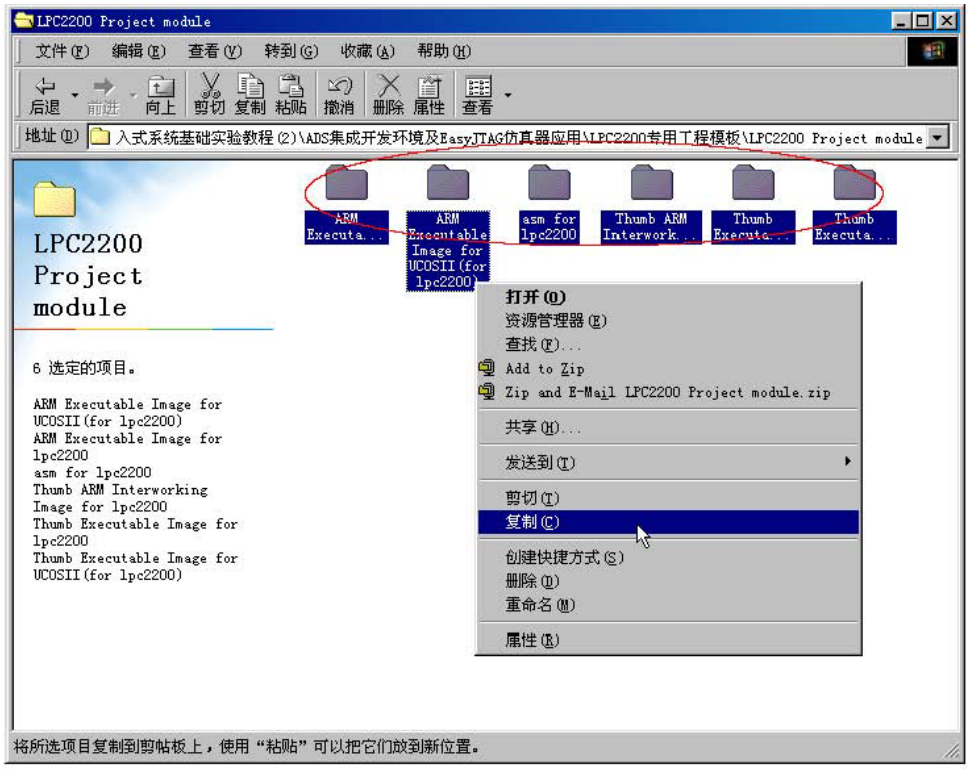

图 1.16 选择拷贝的文件和目录

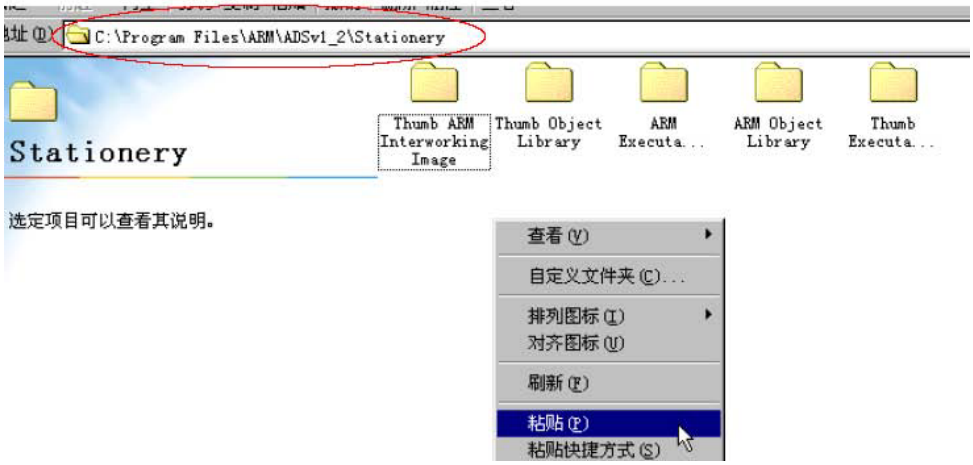

图1.17 复制文件目录

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。

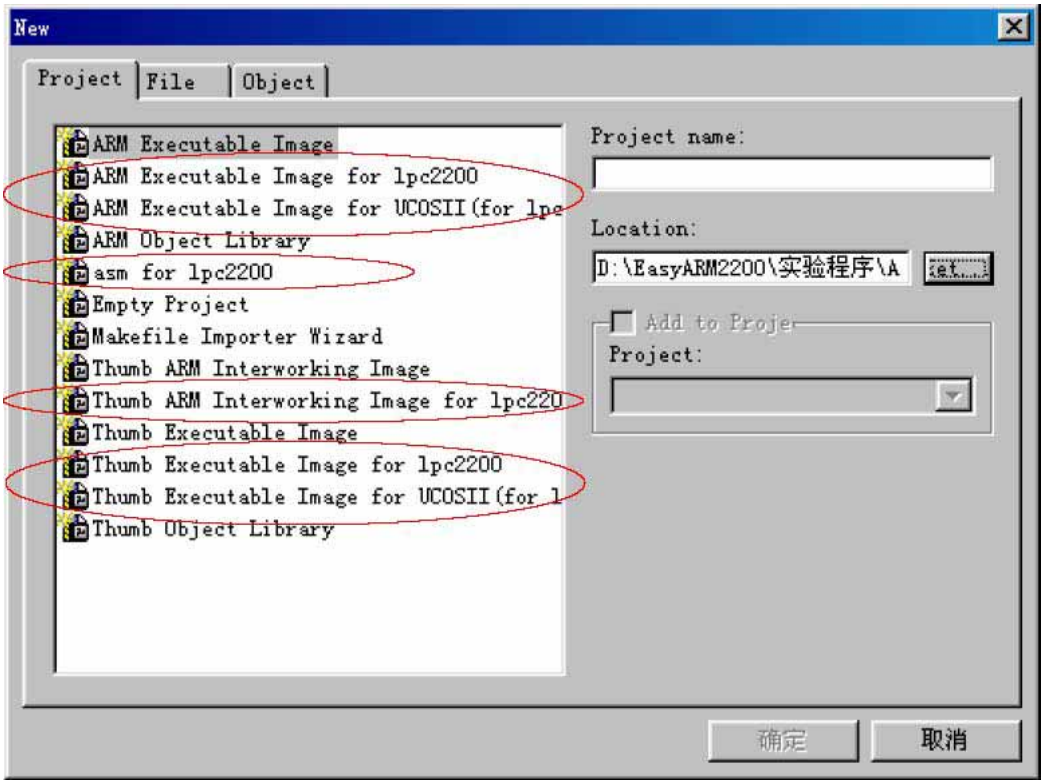

图1.18 增加的工程模板

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

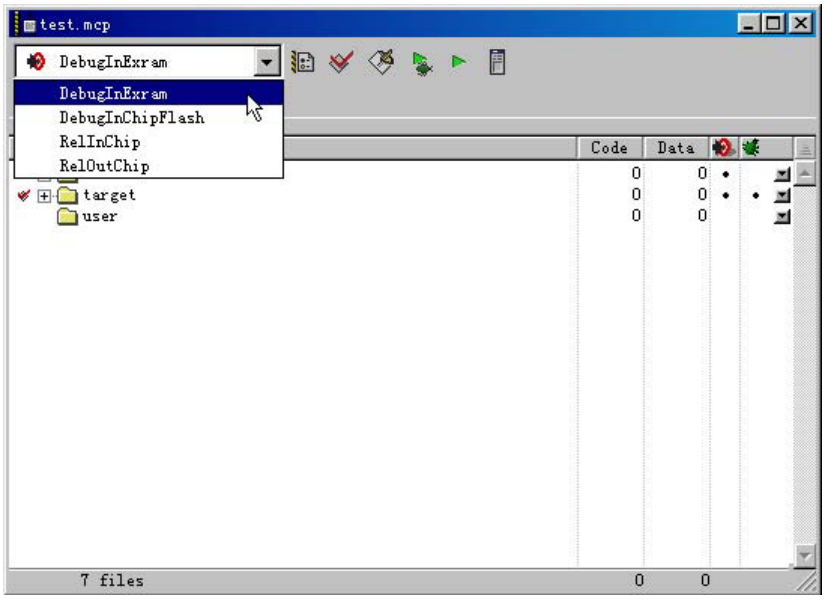

图1.19 用LPC2200 专用工程模板建立的工程

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m em_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TargetRe setInit()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 Sui te 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(0)...”，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“Connectmode...”， 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。

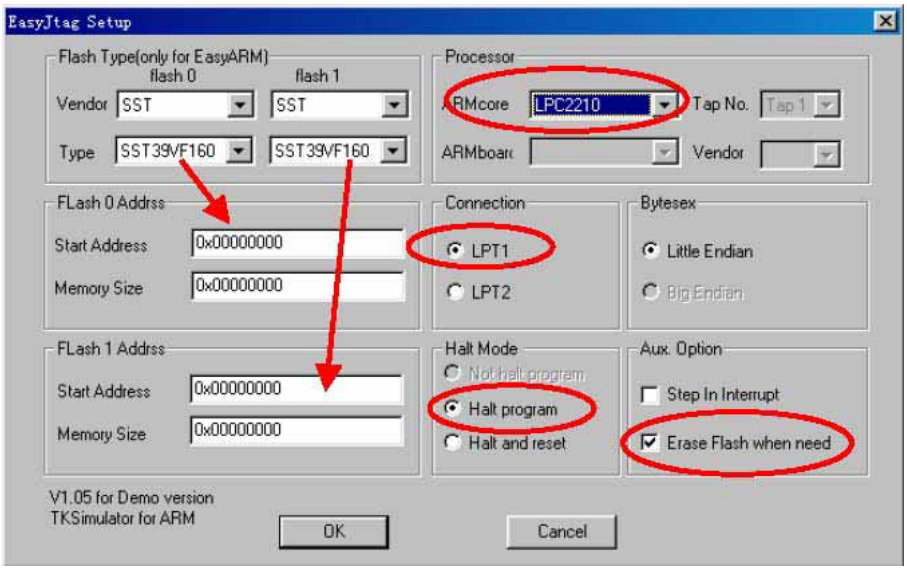

图1.21 “EasyJTAG Setup ”设置窗口

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。

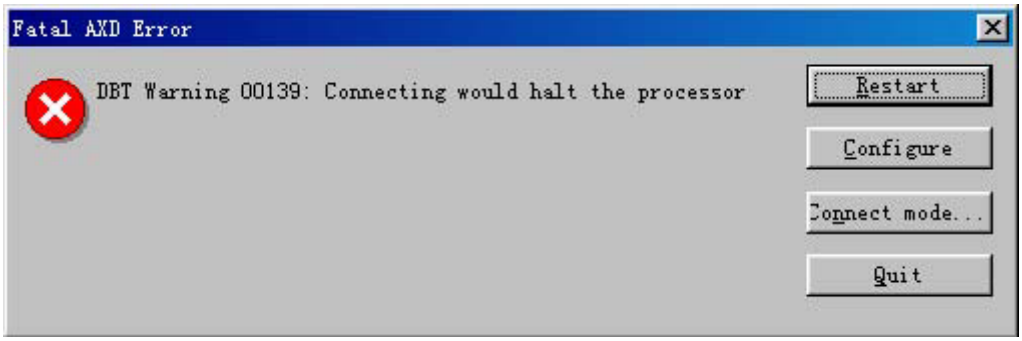

图1.23 Fatal AXD 错误提示

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。

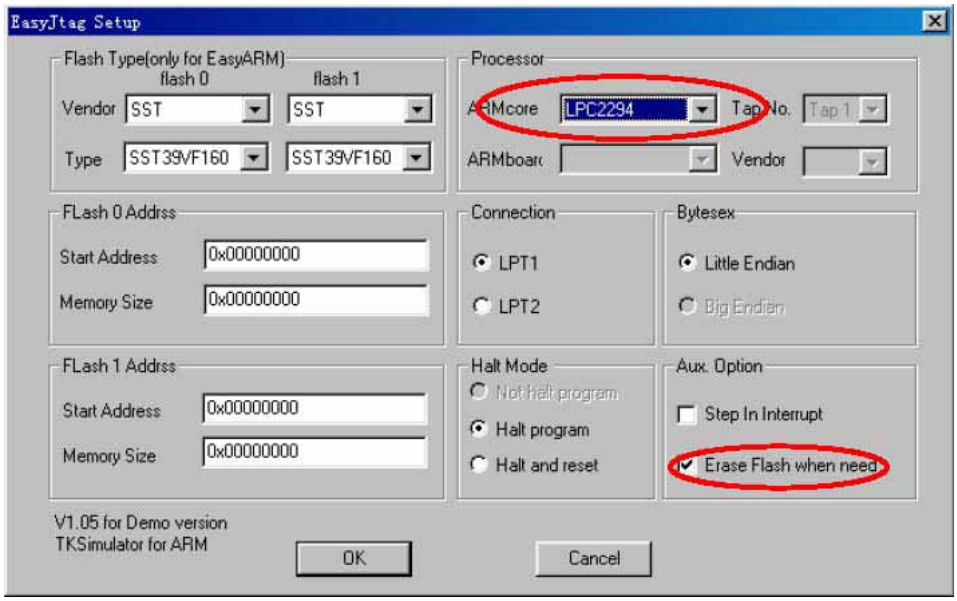

图1.24 下载片内FLASH 的EasyJTAG 设置

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）。

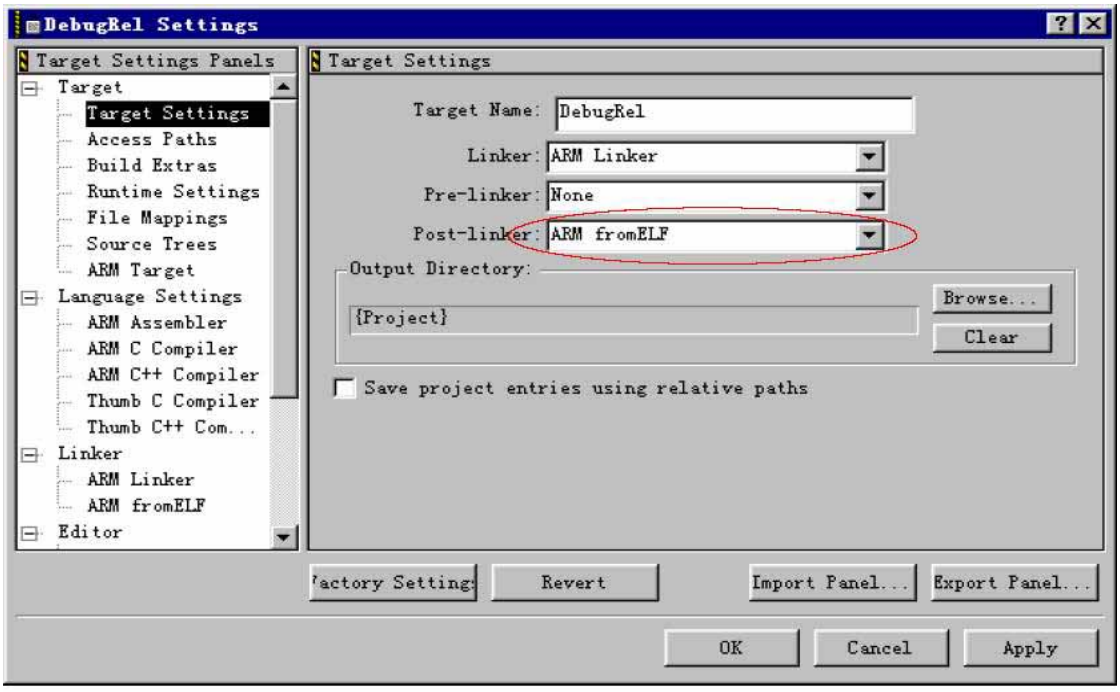

图 1.25　设置 Post-linker

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。

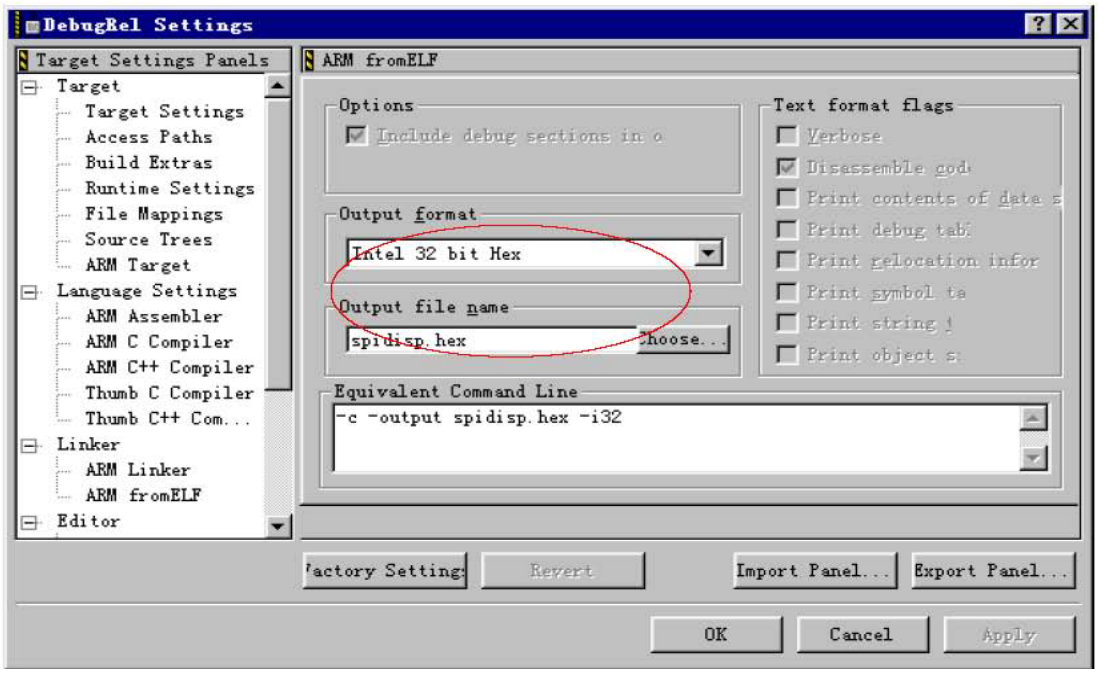

图1.26 生成文件设置

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。

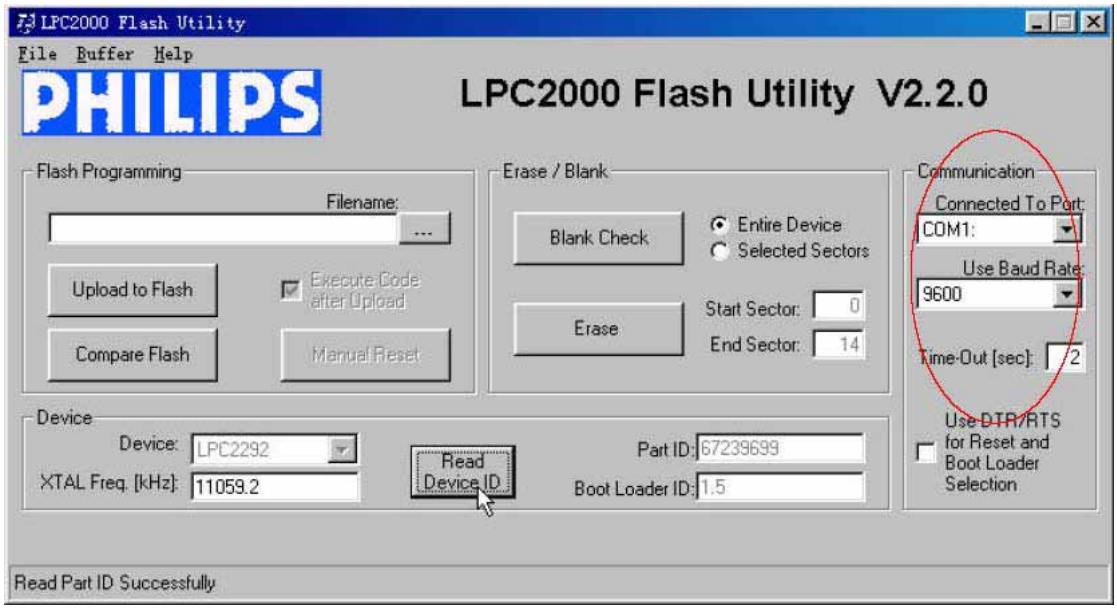

图1.27 LPC2000 Flash Utility 软件设置

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。

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；

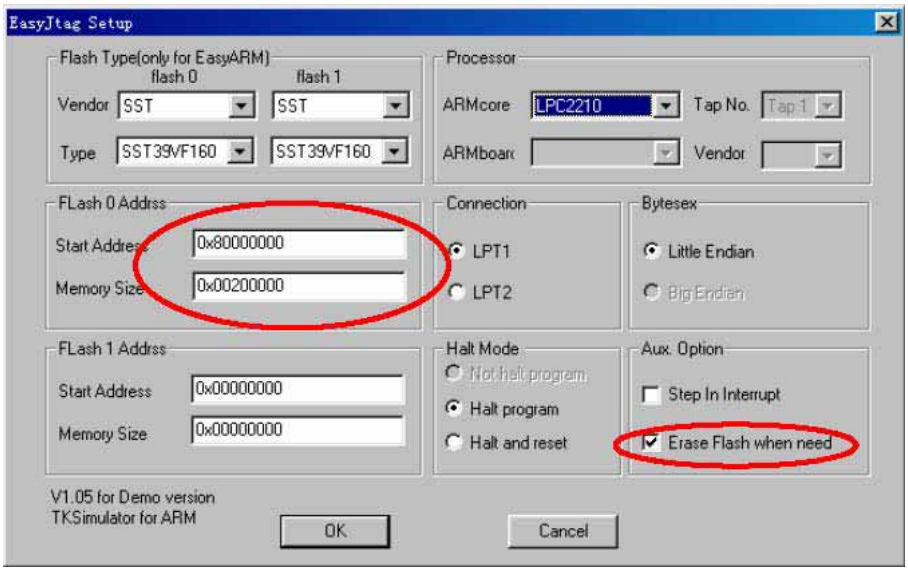

图1.29 下载片外FLASH 的EasyJTAG 设置

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

```
/*****************************************************************************
4       * 文件名：main.c
5       * 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。
6       * 使用向量中断方式，EINT3 下降沿有效。
7       * 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．（1）产生IRQ 中断后，CPSR 寄存器的I 位是0 还是1？
2．（2）如何正确理解VIC 的中断优先级？
3．（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。

1．（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。

2．5． 实验步骤

3．（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。

4．（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>。

5．（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。

6．（4）选用DebugInExram 生成目标，然后编译连接工程。

7．（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。

8．（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。

9．（7）选择【Project】->【Debug 】，启动AXD 进行JTAG 仿真调试。

10．（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
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

```
/*****************************************************************************
1.  * 文件名：main.c
2.  * 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，
3.  * 然后将运行时间值向UART0 发送。
4.  * 说明：将跳线器JP12 全部短接
5.  * 通讯波特率115200，8 位数据位，1 位停止位，无奇偶校验。
6.  * 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 };


/*****************************************************************************
1.  * 名称：DelayNS()
2.  * 功能：长软件延时
3.  * 入口参数：dly 延时参数，值越大，延时越久
4.  * 出口参数：无
    *****************************************************************************/
    void DelayNS(uint32    dly)
    { uint32 i;

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


}


/*****************************************************************************
1.  * 名称：UART0_Ini()
2.  * 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
3.  * 入口参数：无
4.  * 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];
             DelayNS(10);
             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

5． 实验步骤

向量中断控制器的说明。

（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

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;
        IO2DIR = LEDCON;
        rcv_new = 0;

        uart0_set.datab = 8;

        uart0_set.stopb = 1;

        uart0_set.parity = 0;

        UART0_Init(115200, uart0_set);

        U0FCR = 0x81;

        U0IER = 0x01;

    /*  设置中断允许  */
        VICIntSelect = 0x00000000;
        VICVectCntl0 = 0x26;
        VICVectAddr0 = (int)IRQ_UART0;
```

```
    VICIntEnable = 0x00000040;

    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;
        VICVectCntl0 = 0x29;
        VICVectAddr0 = (int)IRQ_I2C;
        VICIntEnable = 0x0200;
}
/*****************************************************************************
        *  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。

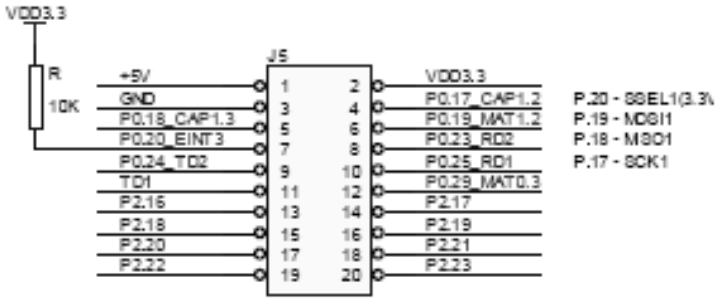

图2.2 SmartARM2210 连接器J5

（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。

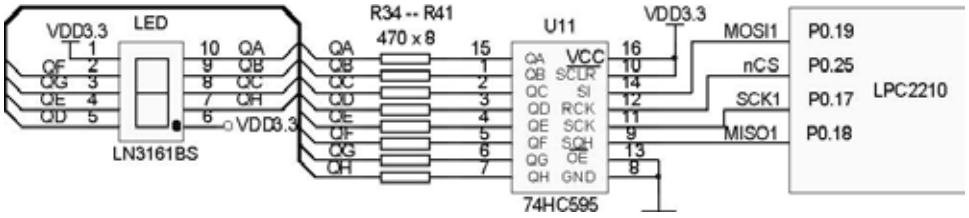

图2.3 LED 显示SPI 接口板

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。

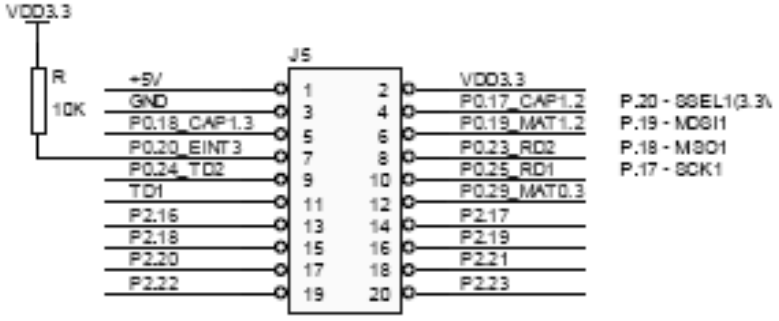

图 3.1 SmartARM2210 跳线座J5

（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。

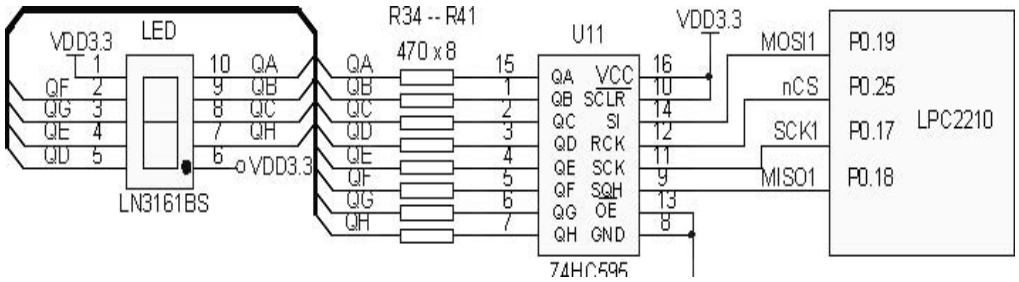

图 3.2 LED 显示SPI 接口板

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。

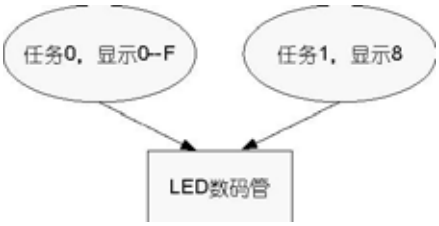

图 3.3 SPI 控制原理

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。

（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。

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);
```

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 I²C 总线的EEPROM 应用

1． 实验目的

掌握I²C 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)，I²C 总线驱动中间件

3． 实验内容

使用I²C 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》I²C 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I²C 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 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。

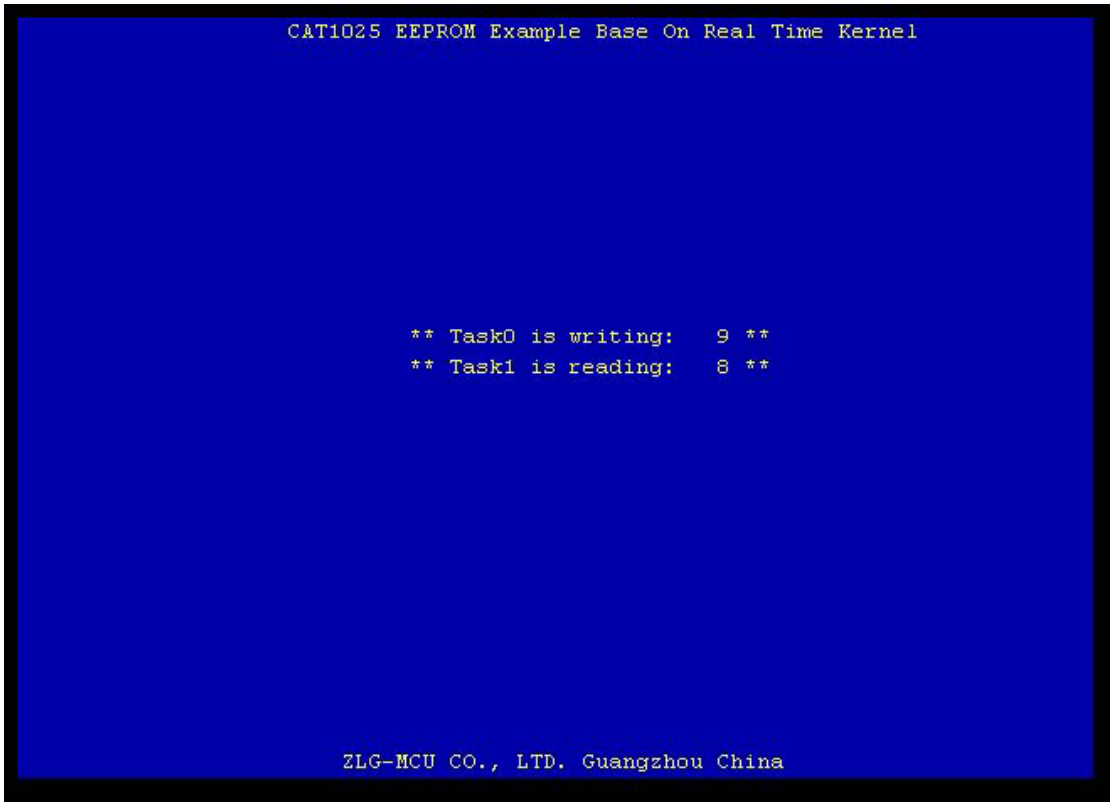

图 3.4 CAT1025 读写提示

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I²C 总线操作，I²C 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 I²C 总线的ZLG7290 应用

1． 实验目的

掌握I²C 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)，I²C 总线驱动中间件

3． 实验内容

使用I²C 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》I²C 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ZLG7290 的数据手册。

（4）了解I²C 中间件的函数调用方法。

5． 实验原理

利用I²C 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I^2C 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 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 3.10。

程序清单 3.10 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 3.11 。

程序清单 3.11 I^2C 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。

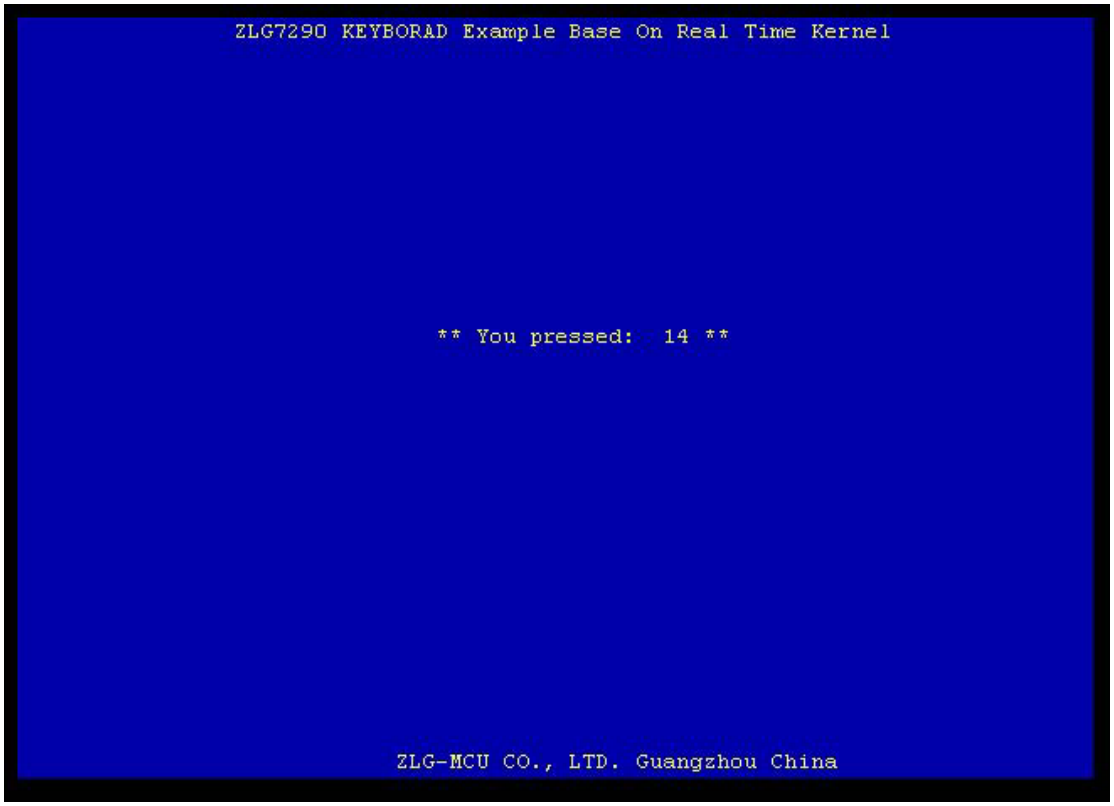

图 3.5 ZLG7290 键值提示

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 拔号操作(建立连接)		
函数参数	Number： 拔号串，即拔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		

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 拔号(#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 拔号，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 拔号字符串
        * 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);
}
```