实验一　　TMS320LF2407 DSP 实验开发系统及 CC 软件应用

一．实验说明

在本书的程序设计实例中，是以 DSP 微控制器 TMS320LF24x 芯片作为设计对象，以 TMS320LF2407 芯片构成的实验开发系统作为目标系统；用 XDS510 硬件仿真器通过 JTAG 接口与开发调试主机联接，来建立程序的联机调试环境。

Windows 版的 Code Composer（简称 CC）是一个功能强大的高级语言交互式调试器，它具有较多的菜单命令，对于经常用到的调试操作提供了相应的工具按钮。CC 的功能非常丰富，不可能对其所有功能全面介绍，本着从简单实用角度出发，本次实验将以一个程序例子，说明如何使用 CC’C2000 来开发简单的汇编程序、编译并使用仿真器运行这一程序。同学们也可以根据自己的应用问题，尝试创建新的项目、新的工作组和新的源文件。

二．实验目的

1．认识 TMS320LF2407 DSP 实验开发系统的硬件结构。

2．　了解 TMS320LF2407 DSP 应用程序的开发调试流程。

3．学习使用 CC’C2000 调试 TMS320LF2407 DSP 程序。

三．实验内容

1．预习附录四、五中的相关内容

2．CC’C2000 使用操作练习

四．实验操作练习

下面将引导同学们利用 CC’C2000 建立一个简单的项目，并且进行一些基本的调试，以帮助同学们尽快地熟悉集成开发环境 CC’C2000 典型的使用方法。

1．实验设备链接

在联机调试时，需要将调试主机 PC、XDP 硬件仿真器及待调试的目标系统按如下方法进行链接：

•用 JTAG 排线电缆两端链接的仿真头分别插入实验板上的 J2 插座引脚和仿真器；

•用并口电缆将 PC 主机并行口与仿真器相连；

•将稳压电源的输出调为＋5V 分别引入 P10 插座引脚的 1、2 和 3、4 中；

•将＋5VDC/1.5A 电源原边接 220V 交流电压，副边链接仿真器外接电源插孔（针对 XDSPP 仿真器）。

2．集成调试软件安装（已安装好，不需要重做）

在 Windows 2000 操作系统之下，安装 CC4.10 版本的 CC 的全过程参见附录五。

3．仿真器软件 Driver 安装（参见附录四）

3． 仿真器运行环境设置

对计算机运行环境作某些设置的目的是让 CC 按照用户的各项具体要求进行 DSP 的程序调试。具体操作方法参见附录四。

4． CC’C2000 的简单应

CC’C2000 是 CC 的核心部分，用于创建和管理项目，为开发人员提供自动化程序高、操作简便的符号化调试工作平台。在 CC’C2000 中的用户文件是组织成项目的。因此，使用 CC’C2000 的第一件事是创建新的项目，以说明用户准备运行软件的目标系统 CPU，并创建项目所包含的文件列表。

（1）创建简单的项目（在本实验指导书中，若不声明，点击或双击均指鼠标左键）

利用 CC’C2000 创建用户程序时须要创建一个项目“Project”，创建项目后可以把源代码文件和必要的库文件加入进项目中。具体步骤如下：

•建立一个保存用户文件的文件夹，例如“C:\tic2xx\myprojects\work”。

•把“C:\tic2xx\c2000\tutorial\realtime”的文件复制到新建立的文件夹中。

•在 Windows 桌面上双击“CC’C2000”图标，或在“开始”菜单中选择“CC’C2000”，显示出如图 1.1 所示的 CC’C2000 操作界面。

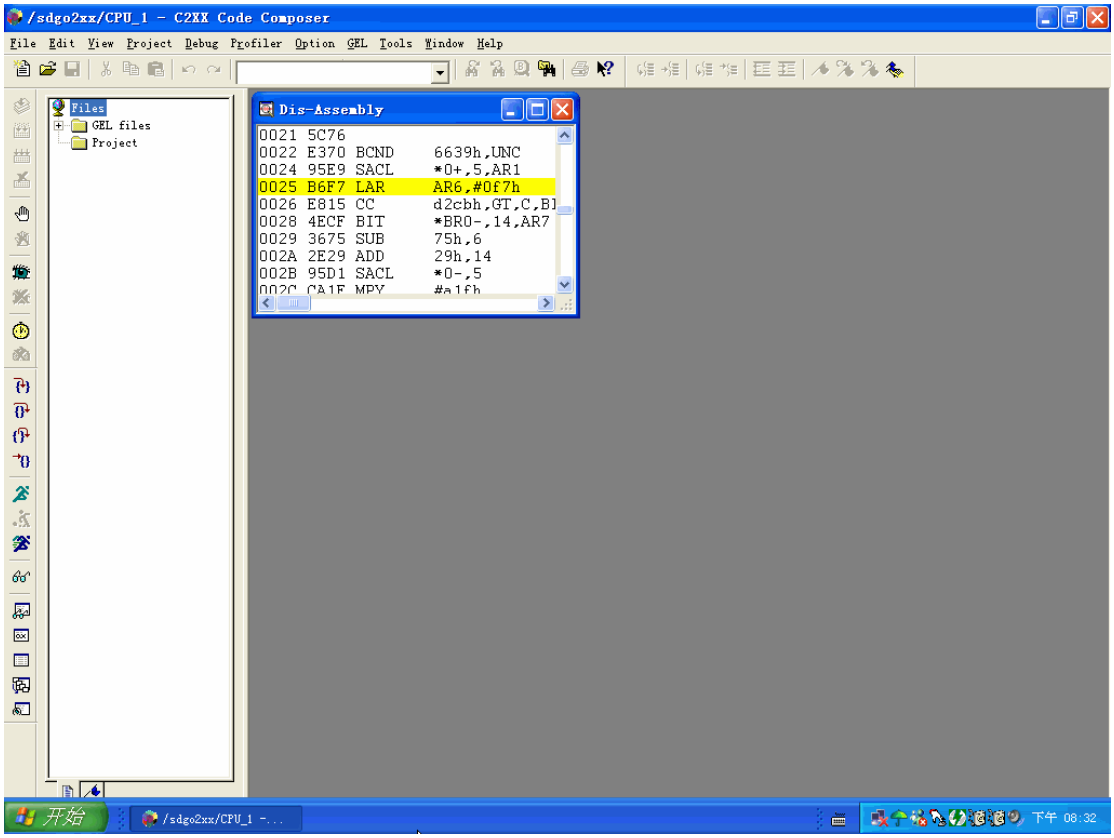


图 1.1　CC’C2000 操作界面

•在图 1.1 所示的界面中，选择菜单命令“Project＞New”（项目＞新建）时，出现一个如图 1.2 所示的“Save New Project As”（另存为项目）对话框，选择要保存项目文件的文件夹（即前面创建的工作目录 work），在“文件名”条形框中输入一个自己所认定的文件名，本试验教程中取用“sy1”作为我们新创建的项目文件名输入并保存。这时 CC’C2000 会自动创建一个名为“sy1.mak”的项目文件。

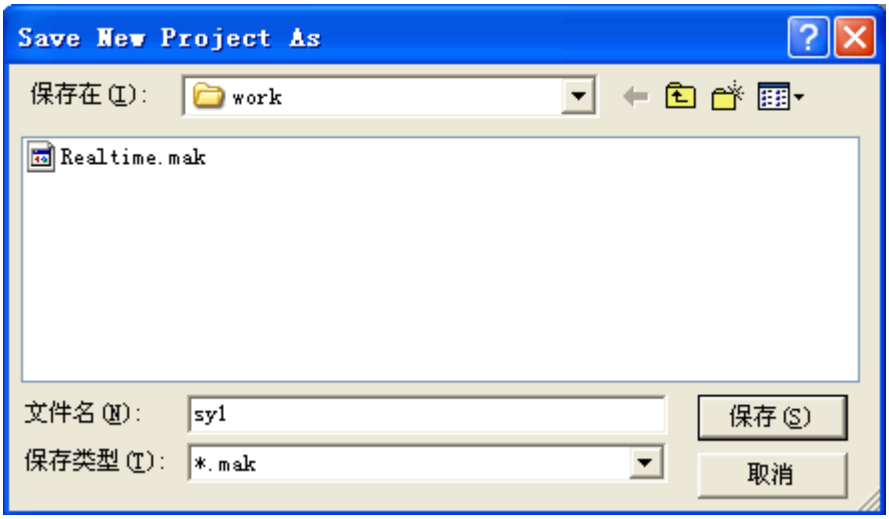


图 1.2　新项目保存对话框

（2）新建、编辑一个简单的源文件

•在图 1.1 所示的界面中，选择菜单命令“File＞New＞Source File”(文件＞新建＞源文件)时，工作区内会出现一个如图 1.3 所示的文本编辑窗口，作为创建源文件的编辑窗口，它的标题栏中显示“Untitled1”，表示源文件尚未命名。

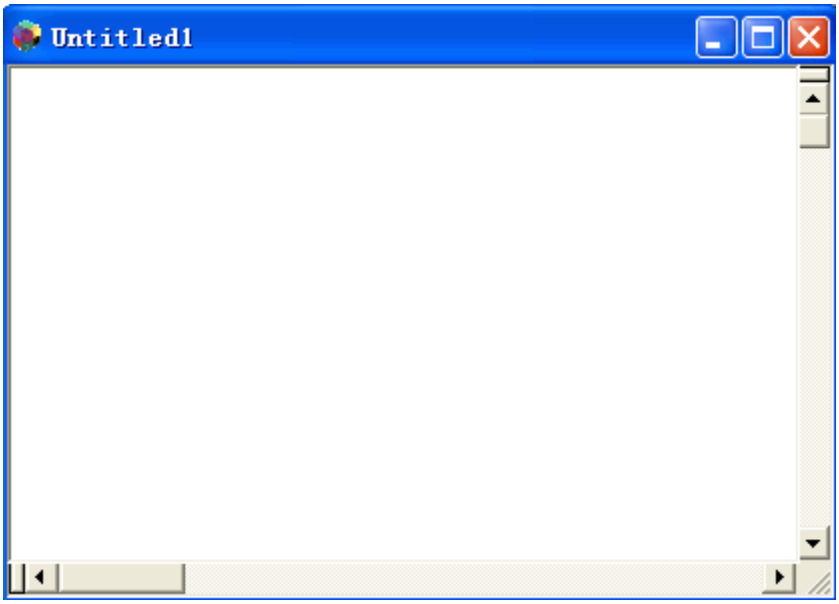


图 1.3　文本编辑窗口

•在图 1.3 中，若欲为即将输入的源文件预先命名，那就选用菜单命令“File＞Save As”（文件＞另存为），随即会出现一个如图 1.4 所示的文件“保存为”对话框。

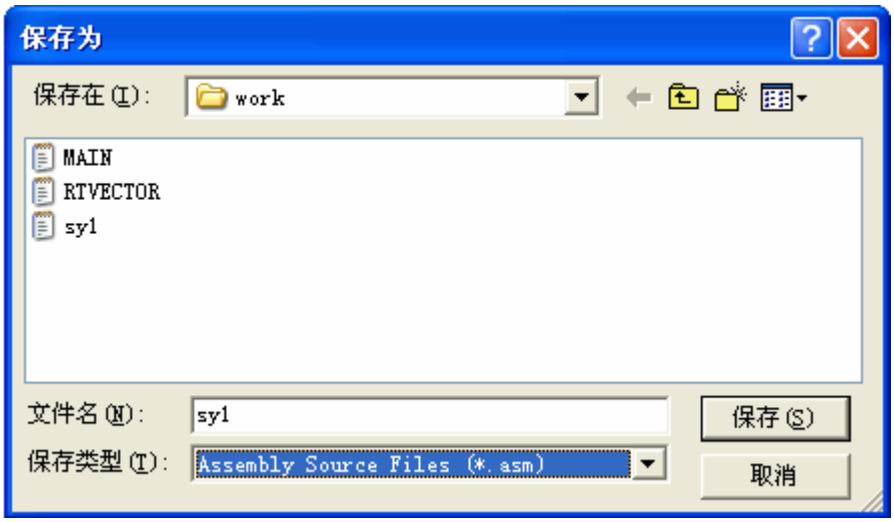


图 1.4　保存为对话框

在图 1.4 中所示的“文件名”条形框内输入文件名“sy1”，并且它的存放目录与项目文件存放目录确保一致。然后点击“保存”按钮，对话框消失，文本编辑窗口的标题栏中换成了刚定义的文件名“sy1”（参见图 1.5 所示的标题栏）并将以扩展名“.asm”保存。

然后在文本编辑窗口输入下面一段程序，作为我们编辑、汇编和调试的实例程序。CC’C2000 编辑器不支持汉字编辑工作，分号后面的注释部分可以用英语或者汉语拼音，也可以使用其他的编辑软件。由于汇编语言源文件[.asm]属于 ASII 码的纯文本文件格式的文件，当然也可以选用 Windwos 附件中的记事本或写字板等其它可以编辑纯文本文件的软件作为源文件编辑器，这样输入汉语注释就不存在任何障碍了。不过，保存时文件扩展名要用“.asm”，并且必须保存为纯文本格式，以便被 CC’C2000 调用。

CC’C2000 编辑器具有较强的编辑功能：全屏幕编辑，可就地修改，所见即所得；跨文件整块剪贴技术；彩色辨识正文等。它的键盘命令配合常规的鼠标操作可实现 C 语言和汇编语言源文件的编辑。

例 1.1

本程序将循环点亮 8 个发光二极管，通过用软件延时的方法来调整发光二极管的延时间隔。当然，在掌握了本程序后，完全可以充分发挥想象，改变一下程序内容和灯的接法（比如用光耦驱动继电器来控制霓虹灯），做出各种变化非凡的大型灯光广告牌。

程序清单：

```
                .include    "vector.h"
                .include     "F2407REGS.H"
                .def        _c_int0,LED
LED             .set        0200h
                .bss        DEYH,1
                .text
SYSINIT:                                ;系统初始化子程序
                SETC        INTM        ;关闭总中断
                CLRC        SXM
                CLRC        OVM
                CLRC        CNF         ; B0 is configured as data space
                LDP         #SCSR1>>7   ;取得 SCSR1 所在的页
                SPLK        #81FEH,SCSR1    ; CLKIN=6M,CLKOUT=24M
                SPLK        #0E8H,WDCR      ; Disable WDT
                LDP         #IMR>>7
                SPLK        #0h,IMR         ;disable all interrupt
                SPLK        #0FFFFh,IFR     ; clear all the interrupt flags of first level
                LDP         #MCRC>>7
                LACL        MCRC
                AND         #0FBFFh         ;IOPF2 口为通用的 IO 口
                SACL        MCRC
                LACL        PFDATDIR
                OR          #0400h          ;使 PF2 口为输出口
                AND         #0FFFBH
                SACL        PFDATDIR        ; 不使能 74HC273 芯片
                LACL        MCRA
                AND         #0FF00H
                SACL        MCRA            ;IOPB[0-7]为一般 I/O 口功能
                LACL        PBDATDIR
                OR          #0FF00H
                AND         #0FF00H         ;IOPB[0-7]为输出功能,并马上清 0
                SACL        PBDATDIR
                RET
; ********************************************************************************
CLOCK:                                  ; 产生 74HC273 驱动脉冲的子程序
                LDP         #PFDATDIR>>7
```

```
                LACL    PFDATDIR
                OR      #0004H          ;PF2 口输出高电平
                SACL    PFDATDIR
                RPT     #10H
                NOP                     ;延时
                LACL    PFDATDIR
                AND     #0FFFBH         ;PF2 口输出低电平
                SACL    PFDATDIR
                RET
; ***************************************************************************
DELAY:                                  ;延时子程序
                LDP     #DEYH>>7
                SPLK    0FFFFH,DEYH
                LACL    DEYH
                CLRC    C
CON:
                SUB     #1
                BCND    EXIT,NC
                NOP
                B       CON
EXIT
                RET
_c_int0
                                        ;主程序的入口
                CALL    SYSINIT         ;系统初始化
LOOP1:
                LDP     #LED>>7
                SPLK    #0001h,LED      ;给 LED 赋初值为 1
LOOP:
                LDP     #PBDATDIR>>7
                LACL    PBDATDIR        ;把 PBDATDIR 的值赋给 ACC 寄存器
                AND     #0FF00H         ;把数据位都清 0
                LDP     #LED>>7
                OR      LED
                LDP      #PBDATDIR>>7
                SACL    PBDATDIR        ;把需要显示的值赋给相应的寄存器
                CALL    CLOCK           ;产生琐存脉冲
                LDP     #LED>>7
                LACL    LED
                CLRC    C               ;清进位位,以免对移位造成影响
                ROL                     ;左移一位
                SACL    LED
                CALL    DELAY           ;延时
                LDP     #LED>>7
```

```
        BIT     LED,6           ;复制 LED 寄存器的第九位到 TC 位
        BCND    LOOP1,TC        ;如果 LED 的移位次数已到,则对 LED 重新赋值
        B       LOOP            ;循环
        NOP

PHANTOM RET
GISR1   RET
GISR2   RET
GISR3   RET
GISR4   RET
GISR5   RET
GISR6   RET
        .end
```

源程序输完之后（如图 1.5 所示），应该用菜单命令“File＞Save”（文件＞保存）及时保存到 C:\tic2xx\myprojects\work\sy1 asm 路径及文件名下。对于较长的源程序，如果一次不能输入完毕而需要中途退出，也应该及时保存，以便在下一次能继续前次的工作。在长程序文件的输入过程中，不时地进行存储是一种好习惯，也可避免因电源意外掉电而造成前功尽弃。

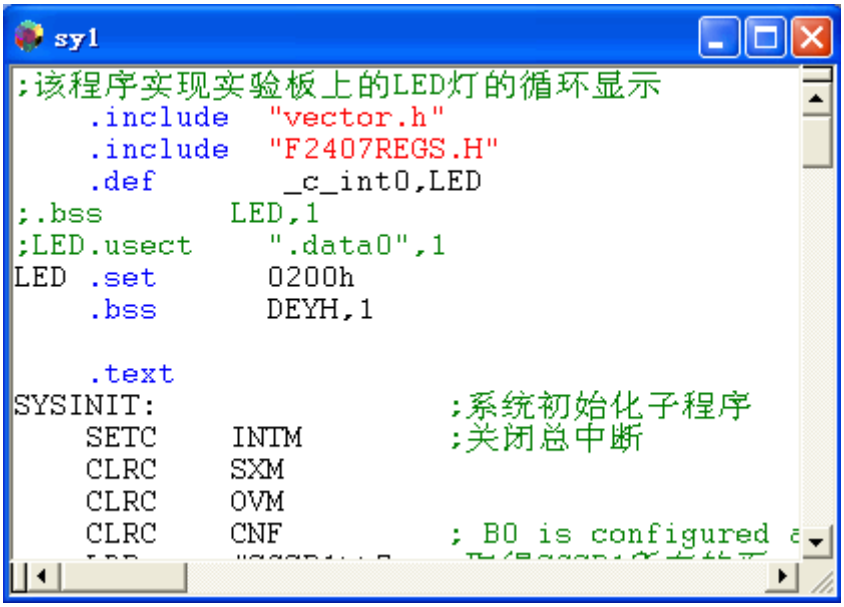


图 1.5　源程序编辑窗口

（3）将文件添加到该项目中

由于项目文件是项目的管理文件，故项目管理的信息都需存放在“sy1.mak”的项目文件中，在对用户系统进行开发时需要将所需文件包含在项目文件中，即使用工程管理方法：一次性将工程的全部源文件、头文件、链接命令文件、用户库文件送入工程管理器，统一管理“汇编 / 编译”和“链接 / 定位”，无须人工干预。

在图 1.1 所示的界面中，先选择菜单命令“Project＞Open”（项目＞打开），再根据给出的“Project Open”对话框，选择已存项目文件所在的文件夹并打开文件（如图 1.6 所示），然后再选择菜单命令“Project＞Add Files To Project”(项目＞添加文件到项目)，出现一个如图 1.7 所示的“Add Files to Project”对话框，然后将本项目要用到的.asm 汇编源文件、.cmd 链接命令文件和一个“rts2xx.lib”的库文件按类一一添加到该项目中，在默认情况下，该库

文件可以在“C:\tic2xx\c2000\cgtools\.lib”目录中找到。对于.h 头文件和在程序中用“include”引用的文件，只要在同一个目录下，工程管理器会自动将其加入。工程管理器不允许用户添入其它类型的文件。

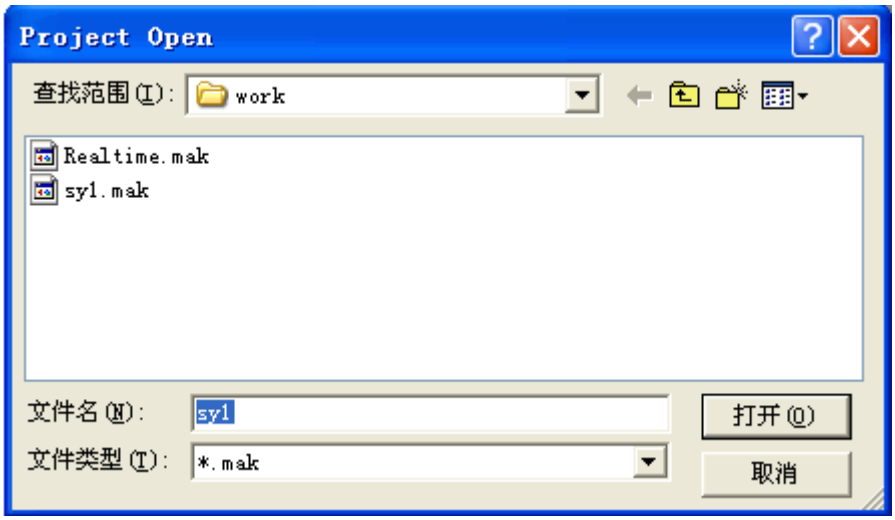


图 1.6　项目文件选择对话框

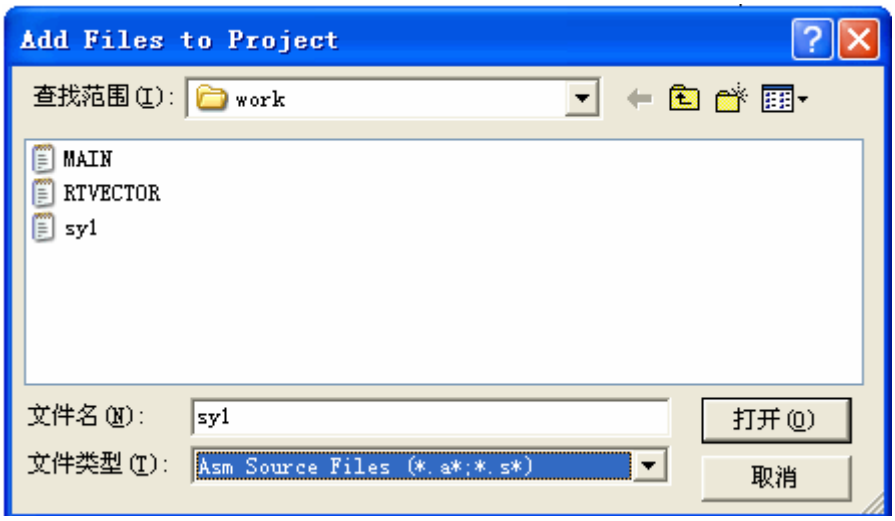


图 1.7　添加文件对话框

（4）察看项目文件

一旦编译了文件，Project 窗口将用树型结构显示项目包含的组和文件的关系，如图 1.8 所示。组用于把相关的源文件收集在一起，每一个组可以包含在一个或多个目标中，而且源文件可以出现在一个或多个组内。通过该项目管理窗口，可以十分方便地察看项目所包含的文件是否加入到相应文件夹中。

当启动 CC’C2000 后，会自动显示出项目管理窗口。如果没有显示该窗口，可以选择菜单命令“View＞Project”（观察＞项目）使之激活。用鼠标右键点击“Project”再选择“Open project”，操作方法如图 1.9 所示，根据给出（如图 1.6）的对话框，用前面相同的方法选择打开项目文件，然后可选择点击靠近源文件的“＋”符号来显示包含文件（具体操作参见附录五）。

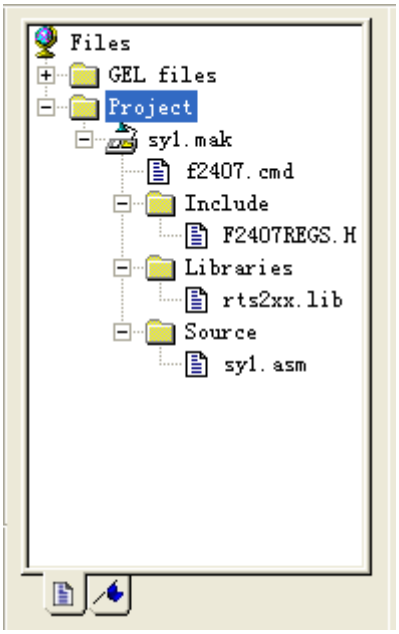


图 1.8　项目管理窗口

图 1.9　文件操作方法

（5）编译链接

启动 CC’C2000 汇编编译器对汇编语言源程序进行编译的方法有多种，在此使用的方法是：在图 1.1 所示的界面中，先选择菜单命令“Project＞Open”（项目＞打开），以便选定需要编译的项目文件，然后再选择菜单命令“Project＞Rebuild All”（项目＞汇编编译和链接所有文件），CC’C2000 将自动调用汇编编译器和链接器将项目文件“sy1.mak”管理下的源文件“sy1.asm”编译生成目标文件“sy1.obj”，这时会出现一个如图 1.10 所示的“Build”（创建）窗口，其中倒数第二行用蓝色字告诉我们：“Build Complete”（创建完成），如果有错，最后一行提示错误的个数和数型。其实，这是笔者事先有意安排的一处错误，以便同学们了解 CC’C2000 汇编编译器是如何帮助我们查找错误的（即错误和警告自动定位）。

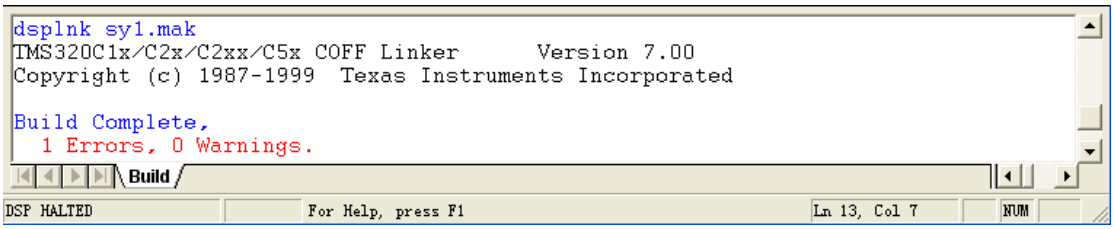


图 1.10　创建结果窗口（不成功）

根据系统提示该错误的类型和位置回到原程序中查看和修改，然后采用上述方法重新进行编译。如果编译正确，将会链接生成一个文件名为“sy1.out”的可执行文件。再次出现如图 1.11 所示的“Build”（创建）窗口，最后一行告诉我们：“没有错误。”至此，我们就建好了一个可以在 CC’C2000 环境下用 XDS510 硬件仿真器进行调试的、完整的项目。

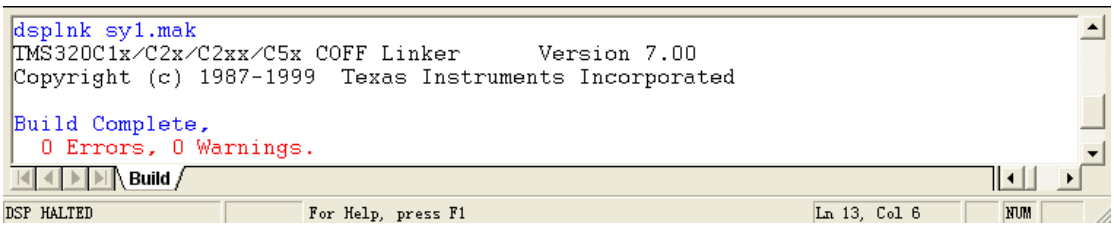


图 1.11　创建结果窗口（成功）

对于建好的项目，一般情况下，在退出 CC’C2000 调试环境时，系统会自动将“项目文件”保存在“.wks”文件中。而对于作了修改的源程序则需要重新保存，选用菜单命令“File＞Save”即可保存到“sy1.asm”文件中。

在编译链接成功之后，需要将生成的“sy1.out”可执行文件装载到实验板上之后才能对程序进行调试仿真。因此，在图 1.1 所示的界面中，选择菜单命令“File＞Load Program”（文件＞装载程序），出现一个如图 1.12 所示的“Load Program”对话框，然后将可执行的“sy1.out”文件下载到实验板上。文件下载之后，就可对文件进行在线调试。

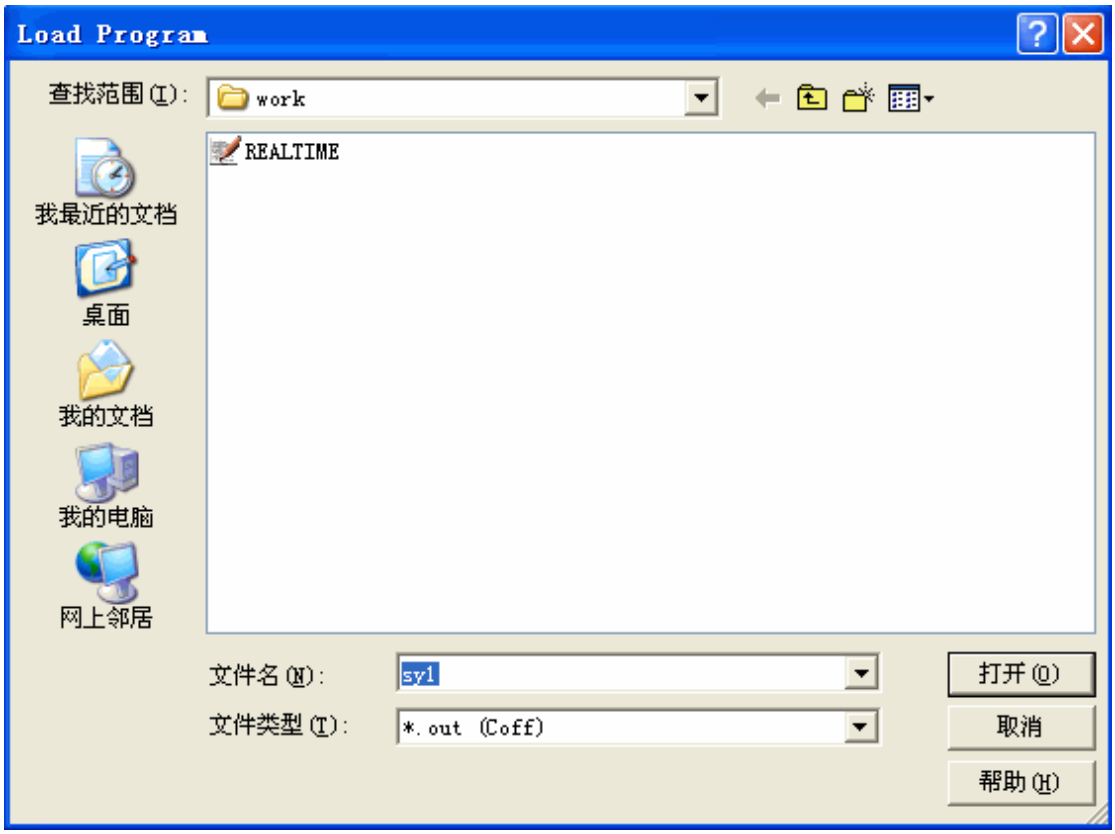


图 1.12　程序装载对话框

注意：在本次实验中所介绍的这种单文件项目，要求项目文件名、目标文件名和源文件名必须保持一致。

（6）调试程序

程序调试就是检验我们设计的程序，是否能够正常运行，是否产生正确的结果，是否存在设计漏洞，算法（可以通俗地理解为，用计算机的思想解决实际问题的方法）设计是否合理，是否能够准确地控制各种硬件资源，是否能够实现预期的功能，等等。

CC’C2000 的调试环境功能很强，可以在 C 语言级调试，也可以在汇编语言级调试，并提供了多种运行程序的方式或调试手段，比较常用的有以下几种：连续运行、设置观察窗、单步运行、动画运行和设置断点运行等。下面将通过前面给定的例子更深入地介绍各种调试手段的使用方法：

•连续运行方式：

在图 1.1 所示的界面上，将源程序观察窗作为当前窗口。首先应选择菜单命令“Debug＞Reset DSP”（调试＞DSP 复位），或者点击工具栏上的按钮，使 DSP 复位，然后选择菜单命令“Debug＞Run”（调试＞程序运行），或者按动 F_5 键，或者点击工具栏上的  图标按钮，均可令程序进入实时运行状态。当输入上述命令时，我们似乎看不出程序有任何反应，

其实程序正在连续运行，这可以通过观察工具栏上的  图标按钮是否改变颜色来判断，如果变为浅色则表明程序正在运行。

由于这段程序结构是无限循环程序结构，欲想终止程序运行可点击工具栏上的  图标按钮，或者选择菜单命令“Debug＞Halt”（调试＞停止），或者按动一下键盘上“Shitf ＋F_5”组合键。随即，工具栏的颜色复原，表明程序停止运行，并随机地停留在程序的某一行上。

在程序连续运行过程中，状态栏中的字段得不到及时更新，只有当程序停下来之后，状态栏信息才更新一次。可见，采用这种调试手段，不便于及时了解程序的运行状态，也不便于控制程序的运行过程。对于调试本实验设计的这段循环结构的程序，连续运行方式就不是一种很有效的手段。因此，需要进一步探讨和选择其它的调试手段。

•设置观察窗口：

根据被调试程序的运行需要，可以在 CC’C2000 桌面上同时选择打开最多 6 个不重叠观察窗口。通过观察各种存储器和寄存器的内容，使开发者更直观地分析程序的逻辑功能和运行状况，进而达到调试程序的目的。

实验教程中调试的这个例子比较简单，其中用到的寄存器除了 ACC、ST0、ST1 和 PC 指针等之外，当需要观察一个变量时，在观察变量窗口中输入变量名称，在程序运行过程中该变量会被不停地改写。

我们可以根据这种情况来设置观察窗。除了桌面上已有的一个源程序窗口之外，可以再增设：反汇编观察窗口、存储器窗口、CPU 寄存器窗口、观察窗口。

反汇编观察窗口：用来显示汇编程序和程序存储器的内容。在这些程序中，用一反显高亮条来表示当前指针，可用鼠标点击汇编语句的方法来设置断点，再点击一次即可取消断点。详细用法参见附录五。

存储器窗口：可直接观察存储器的内容。详细用法参见附录五。

CPU 寄存器窗口：其中包含“CPU Register”和“Status Register”两个选项，点击相应的选项可以观察调试过程中 CPU 寄存器和各个状态位的变化情况。详细用法参见附录五。

观察窗口：观察调试过程中的变量、C 表达式、地址和寄存器的值。在图 1.1 所示的界面上，将源程序观察窗作为当前窗口，选择菜单命令“View＞Watch Window”（视图＞观察窗口），将有一个空白窗口出现在 CC’C2000 窗口的下部，如图 1.13 所示。该窗口用来显示所选择的变量、寄存器和存储器的内容。

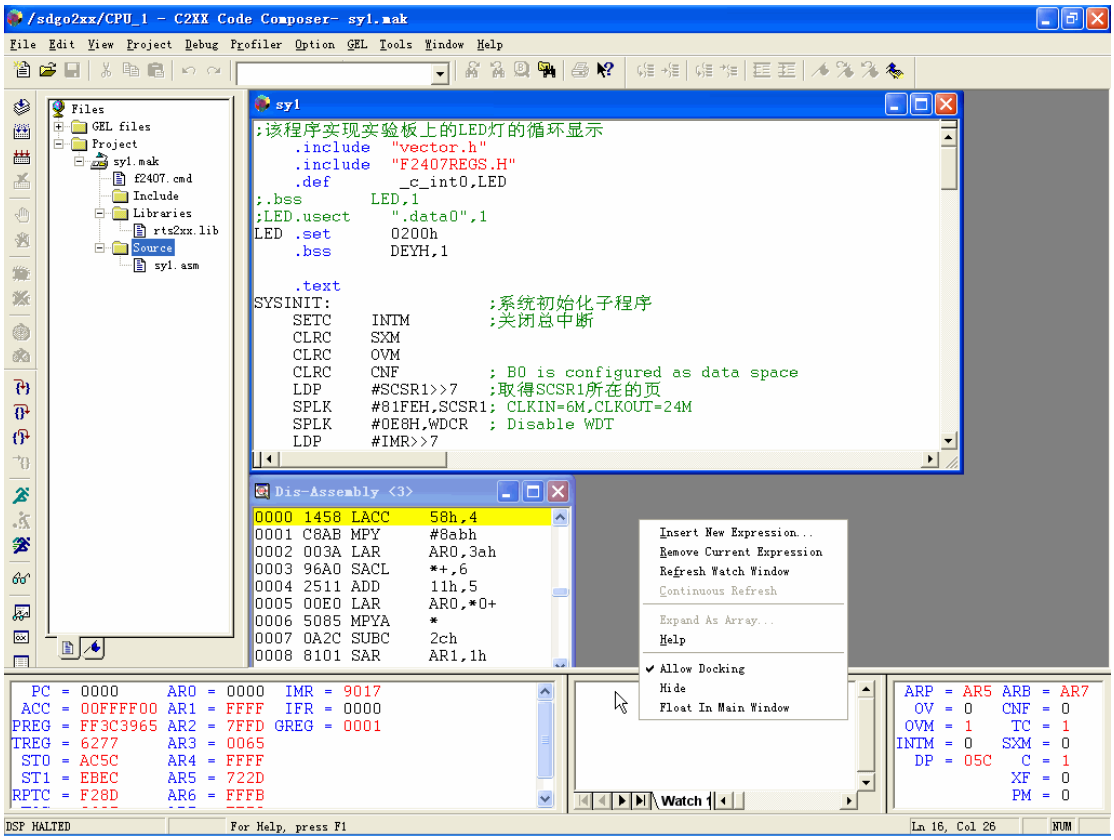


图 1.13　CC’C2000 窗口下的观察窗口

为了监视跟踪变量，用户可以将变量设置为查看点。通常用如下的方法在观察窗口中加入一个新的变量：在空白观察窗口中单击鼠标右键，弹出语境菜单，选择其中的“Insert New Expression”，如图 1.13 所示。进入所选择的图 1.14 “Watch Add Expression”（添加变量对话框），在文本栏中键入打算观察的变量名，然后点击“OK”按钮，即可在观察窗口中增加一个待观察的变量，如果有必要，用同样的方法还可以在观察窗口中添加多个变量。在该观察窗中可以查看到以蓝色显示的变量的物理地址（十进制）、符号名和数值。

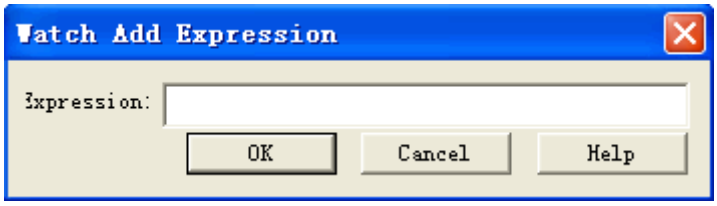


图 1.14　添加变量对话框

通常用如下的方法在观察窗口中删除一个表达式：用鼠标点击需要删除的表达式，单击鼠标右键，在弹出的语境菜单中选择“Remove Current Expression”即可。

如果添加变量时在“添加变量对话框”的文本栏中只输入变量名，则在观察窗口中只显示出该变量的地址；如果需要显示该变量的值，则需要在变量名前加“*”号。

变量的地址、值其显示默认格式为十进制，如果想改变显示格式，则在输入变量时在其后紧跟一个逗号和一个格式指示字母。常用的格式指示字母和其代表格式的对应关系如表 1.1 所列。

表 1.1　常用格式指示字母表

字母符号	代表的格式
D	十进制
E	指数浮点
F	十进制浮点

X	十六进制
O	八进制
U	无符号整型
C	ASCII 字符（字节）

•单步运行方式：

单步运行是一种控制程序运行过程的有效方法，而且能够及时观察到程序的运行状态。每输入一次控制命令，程序就会被执行一条指令，并且立刻更新 CC'C2000 桌面上各个观察窗中的显示信息以及状态栏中的显示信息，也就是及时显示该指令的执行结果。具体操作方法如下：

在图 1.1 所示的界面上，将源程序观察窗作为当前窗口，用前面所述方法使 DSP 复位。然后选择菜单命令 Debug＞Step Into（调试＞单步运行方式），或者按一下键盘上的 F_8 键，或者点击工具栏中的  图标按钮，均可令程序进入单步运行状态。一次次地点击  按钮的同时，可以看到观察窗中出现变红的寄存器或变量，并且其中的值会不断地及时更新。

可见，采用这种调试手段，不仅可以及时地了解程序的运行状态，而且还能很好地控制程序的运行过程。程序的单步运行方式与实时连续运行方式相比，两者具有很强的互补性。

•动画运行方式

动画运行方式既像连续运行方式那样自动控制程序的运行过程，又像单步运行方式那样在每条指令执行过后刷新屏幕显示信息。程序不停，刷新不止，显示效果类似于播放动画片。

在图 1.1 所示的界面上，也应将源程序观察窗作为当前窗口，并先让 DSP 复位，然后选择菜单命令 Debug＞Animate（调试＞动画运行方式），或者按一下键盘上的 F_{12} 键，或者点击工具栏中的  图标按钮，均可令程序进入动画运行状态。同时应注意观察寄存器变量的变化规律，是否符合设计要求。若想让程序停止，点击  按钮即可。

•设置断点运行方式：

在调试长程序的过程中，可以控制连续运行那些简单的或者已调通的程序片段，而控制单步运行那些复杂的或者待调试的程序片段。这样就可以将连续运行方式和单步运行方式的优势结合起来交替使用。例如，有时候希望执行一个程序片段之后暂停下来，观察各寄存器变量的值，以便分析中间结果。这些均可通过在程序中对所选定的行设置断点来实现。可以说，设置断点是控制程序运行过程的另一种有效方法。

一种最简便的设置断点的方法：在图 1.1 所示的界面上确保当前窗口为源程序观察窗。用光标标记准备设置断点的源程序语句行后，点击  图标按钮或者双击鼠标，此时，选定行会出现红色背景，表示设置完成；点击  图标按钮，取消所设置的断点。

对已设置断点的程序进行调试时，可以采用连续运行方式。即先让 DSP 复位，再点击  按钮使程序连续运行，直到遇上第 1 个断点即停止；对运行结果观察完成之后，再次点击  按钮使程序继续运行，直到遇上第 2 个断点又会暂停，等等一直调试下去。

实际上，常常需要将前面介绍的 5 种调试手段混合使用，以达到最佳的调试效果。实现同一功能有多种方法，本次实验仅向同学们展示了 CCS（以 CC4.10 版本为例）的基本用法，

有许多的功能还需要同学们根据自己的习惯爱好去慢慢挖掘。

实验二　　常用指令操作实验

一．实验说明

在设计 DSP 应用系统时，指令组成了 DSP 系统的应用软件。由于种种原因，所编写的软件中总会存在一些问题，因此必须对软件进行调试，通过调试修改软件中不合理的地方。

在软件调试中，一个重要问题是观察指令执行的结果。例如，在 DSP 应用系统中输出一个信号波形，但发现输出的波形与预期的不同，这时就需要检查原因。由于 DSP 指令执行的速度相当快，所以就需要掌握调试中的指令响应观察技术，也就是通常使用的单步执行方法。

二．实验目的

1．　通过在 TMS320LF2407 DSP 实验开发系统中实际使用指令，初步学会如何使用实验中的指令。

2．学习指令的功能和基本使用操作方法。

3．学习 DSP 应用系统中的系统调试方法，学会单步调试的基本方法。

4．学会如何观察指令的执行结果。

三．实验内容

1．熟悉常用汇编指令。

2．熟悉单步执行的调试方法。

3．熟悉在调试环境下观察指令的执行结果和相应寄存器内容。

四．实验步骤

1．立即数载入指令实验操作

立即数载入指令包括：

（1）　累加器载入立即数。

（2）　向辅助寄存器载入数。

（3）　DP 载入立即数。DP 是状态寄存器的 ST0（0~8）位，作为数据空间直接寻址时地址的高 9 位（A15~A7）。

本次练习中，需要完成如下工作：

（1）　向累加器装载立即数 1234h。

（2）　向累加器装载立即数 ABCDh 并左移 4 位。

（3）　向辅助寄存器 AR3-5 装载立即数。

（4）　向 DP 的装载立即数。

可以看到，本次练习不仅是向某个寄存器装载数据，而且还包括向某个寄存器内容指向的某些数据存储单元装载数据。

实验时，可以先编写好上述指令，作为一个程序段进行编译。然后打开 CPU 寄存器观察窗口，利用单步执行命令单步执行程序。每完成一次单步执行操作，可以观察到相应寄存器内容的变化。

单步操作前可以通过 CPU 寄存器观察窗口对各相关寄存器进行修改，以便更清楚地观察到指令执行的结果。

2．直接寻址方式下的数据存取

对数据空间的直接寻址采用以下两种方式：

程序如下：

3．间接寻址方式下的数据存取

在本操作中，练习间接寻址中的操作数存储器操作。这是利用辅助寄存器 ARx

（x=0~7）对数据存储空间进行访问的方法。ARx 的内容就是数据空间的地址，ARx 加星号（*）前缀表示的是 ARx 中的地址所指向的存储器单元，而且保存在 ARx 中的地址在对存储单元访问前 / 后可以进行修改。具体有 7 种修改方式：

```
*              ；访问后 ARx 中地址不变
* +            ；访问后 ARx 中地址加 1
* -            ；访问后 ARx 中地址减 1
* 0 +          ；访问后 ARx 中地址加上 AR0 中的值
* 0 -          ；访问后 ARx 中地址减去 AR0 中的值
* BRO +        ；访问后 ARx 中地址加上 AR0 中的值，并反向进位
* BRO -        ；访问后 ARx 中地址减去 AR0 中的值，并反向进位
```

程序编制说明如下：

（1） TS 是 T 寄存器中存放的移位值；

（2） 语句 AR3 = # 2000h 和语句* AR3 = # 0AC01h 完成了向地址为 2000h 的数据单元存放数据 AC01h；

（3） 语句 T = # 8 和 B = * AR3-<<TS-将 2000h 中的数据左移 12 位载入累加器 B 中，而且 AR3 中的地址值减 1，成为 1FFFh；

（4） 语句*， AR3 + 0 = # 1111h 到语句*， AR3 (# 16) = # 5555h。

程序清单如下：

4．加减运算

使用加减运行指令时，应当注意有关寄存器的影响，主要有如下两个：

（1） ST1 中符号扩展模式位 SXM－－的设置对加减运算的影响：

SXM = 0，符号不扩展

SXM = 1，符号扩展

（2） ALU 运算模式位 C16 的设置及其对加减运算的影响：

C16 = 0，双精度（32 位）运算

C16 = 1，16 位运算

程序清单如下：

5．逻辑运算

程序清单如下：

6．移位运算

程序清单如下：

7．乘法运算

本指令操作主要是练习如何完成乘法操作，其中包括乘、加操作。

程序清单如下：

实验三　　“追灯”式电路控制

一．实验说明

输入 / 输出端口（简称 I/O 口）是 DSP 芯片内部电路与外部世界交换信息的通道。输入端口负责从外界接收检测信号、键盘信号等各种开关量信号；输出端口负责向外界输送由内部电路产生的处理结果、显示信息、控制命令、驱动信号等。本实验利用 TMS320LF2407 芯片做基本的 I/O 控制接口。

使用循环指令和多种送数延时循环程序，设计 DSP 的“追灯”控制器，是最容易和最简单的事例。如果使用查表的方式来做控制，所谓查表的方式是将一些特定的数据，在此是 LED 展示的变化组合数据事先存在数组中，而在程序中逐一由数组中取出个别的样本数据

送往 74HC273 锁存以驱动 LED 发光二极管（SW-DIP8 须置 ON），便可完成“追灯”式电路展示的效果。由于展示的样本数据可以随时组合，因此以查表法来做“追灯”式电路控制的变化较多，展示效果较佳。

实验箱中的 TMS320LF2407 芯片的输出端口用 IOPB0~IOPB7 引脚，TMS320LF2407 的 IOPF2 引脚接 74HC273 的 CLK 引脚作为 74HC273 的控制信号；TMS320LF2407 的 RESET（复位）引脚接 74HC273 的 CLR，复位时，74HC273 输出引脚为低电平。

二．实验目的

1． 熟悉基本的 TMS320LF240X 系列的汇编语言。

2． 了解实验开发系统的基本 I/O 硬件电路的控制方法。

3． 进一步熟悉设计并调试程序的基本方法。

三．实验内容

1．设计并调试用于 TMS320LF2407 的“追灯”控制器，要求“灯”的花样和显示次数均由软件任意设定。

2．程序要求具有多种不同的灯亮花样，能实现发光二极管左移、右移即所谓“追灯”功能。

四．实验硬件电路

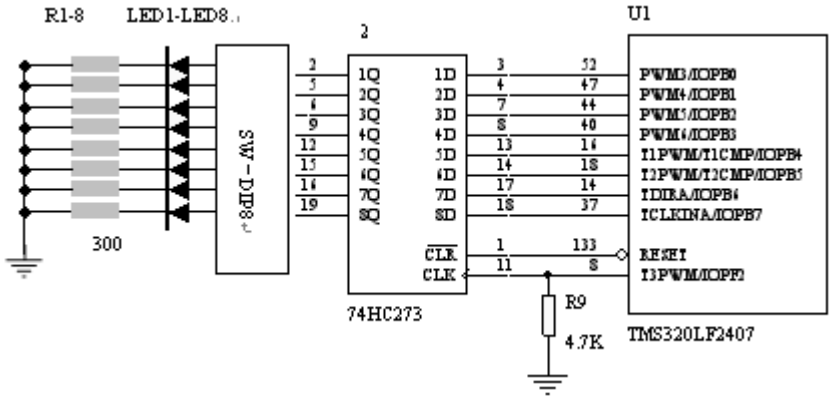


图 3.1　　TMS320LF 与 40LED 接口电路

五．实验参考程序清单

C 程序

I/0 端口作为输出（追灯）

源程序代码：

（1）　所需的复位和中断向量定义文件“vectors.asm”

```
.title      "vectors.asm"
.ref        _c_int0，_nothing
.sect       ".vectors"
reset:          b       _c_int0
int1:           b       _nothing
int2:           b       _nothing
int3:           b       _nothing
int4:           b       _nothing
int5:           b       _nothing
int6:           b       _nothing
```

（2）　主程序

```
// 该程序用于实验演示模板上的 8 个 LED 的循环显示
#include   "register.h"
initial()                                   // 初始化子程序
{
      asm(" setc  SXM");                    // 抑制符号位扩展
      asm(" clrc  OVM");                    // 累加器中结果正常溢出
      asm(" clrc  CNF");                    // B0 被配置为数据存储空间
      asm(" setc  INTM");                   // 禁止所有中断
      *SCSR1=0x81FE;                        // CLKIN=6M，CLKOUT=4*CLKIN=24M
      *WDCR=0x0E8;                          // 不使能看门狗，因为 SCSR2 中的 WDOVERRIDE
                                            // 即 WD 保护位复位后的缺省值为 1，故可以用
                                            // 软件禁止看门狗
      *IMR=0x0000;                          // 禁止所有中断
      *IFR=0x0FFFF;                         // 清除全部中断标志，"写 1 清 0"
      *MCRA=*MCRA&0x0FF;                    // IOPB 端口配置为一般的 I/O 功能， TMS320LF240x 的
                                            // 端口均为 8 位，MCRA 为 16 位因此控制了 IOPA，
                                            // 和 IOPB 的设置
      *PBDATDIR=*PBDATDIR|0x0FF00;          // IOPB 端口设置为输出方式
      *MCRC=*MCRC&0X0FBFF;                  // 把 IOPF2 端口配置为一般 I/O 端口
      *PBDATDIR=*PBDATDIR&0x0FF00;          // 熄灭全部的 LED 灯
      *PFDATDIR=*PFDATDIR|0x0404;           // IOPF2  设置为输出方式，且 IOPF2=1
      *PFDATDIR=*PFDATDIR&0x0FFFB;          // IOPF2=0
      // 以上的操作产生一个脉冲，使 LED 全部熄灭
}
// 主程序
main()
{
      int led;                              // 定义一个局部变量
      int i，k;                             // 定义其它一些临时变量
      initial( );                           // 系统初始化
      while(1)
      {
      for(led=0x0080，i=0；i<8；led=led>>1，i++)
            {
            *PBDATDIR=*PBDATDIR&0x0FF00；// 首先屏蔽 IOPB 的各位
            *PBDATDIR=*PBDATDIR|led；      // 把需要显示的值赋给 IOPB 端口
            *PFDATDIR=*PFDATDIR|0x0404；   // IOPF2  设置为输出方式，且 IOPF2=1
            *PFDATDIR=*PFDATDIR&0x0FFFB；// IOPF2=0，这两句语句给一个脉冲，
                                           // 使 LED 上显示 IOPB 端口的值
            for(k=0；k<0x0ffff；k++)
            k=k;                           // 为了保证显示时间，给一定时间的延时
            }
      }
}
```

```
// 直接返回中断服务程序
void interrupt nothing( )
{
      return;
}
```

汇编程序

```
;I/O 端口作为输出（追灯）
;（1）      主程序
IOSFT_REG   .usect     ".data0",1          ;要显示的数据寄存器
            .include   "F2407REGS.H"       ;引用头部文件
;           .include   "vector.h"
            .def       _c_int0
            .text
_c_int0:                                   ;相当于主程序的入口
            CALL       SYSINIT             ;调系统初始化程序
            LDP        #DP_PF2             ;指向 7080h～7100h 区
            LACL       MCRA
            AND        #000FFH             ;IOPB 口配置为一般 I/O 功能
            SACL       MCRA
            LACL       MCRC
            AND        #0FBFFH             ;IOPF2  配置为一般 I/O 功能
            SACL       MCRC
            LACL       PBDATDIR
            OR         #0FF00H             ;IOPB 口设置为输出方式
            SACL       PBDATDIR

            LDP        #5H                 ;指向 0280h～0300h 区
            SPLK       #01H,IOSFT_REG      ;给显示的数据赋初值
LOOP:       LDP        #DP_PF2
            LACL       PFDATDIR
            OR         #0404H              ;IOPF2  设置为输出方式,且 IOPF2=1
            SACL       PFDATDIR            ;开 74HC273 片选信号
            LDP        #5H
            LACL       IOSFT_REG
            LDP        #DP_PF2
            OR         #0FF00H
            SACL       PBDATDIR            ;送要显示的数据到 IOPB 口

            LACL       PFDATDIR
            AND        #0FFFBH             ;IOPF2=0   (应该使能显示)
            SACL       PFDATDIR            ;关 74HC273 片选信号
```

```
                CALL    DELAY               ;调延时程序
                LDP     #5H
                LACL    IOSFT_REG
                SFL                         ;左移一位
                SACL    IOSFT_REG
                BIT     IOSFT_REG,BIT8      ;判是否循环完一次,即已点亮第 8 个发光二极管
                BCND    LOOP1,TC
                B       WAIT
LOOP1:          LDP     #5H
                SPLK    #01H,IOSFT_REG      ;如循环完一次则显示数据赋初值
WAIT:           NOP
                B       LOOP
;（2）      系统初始化程序
SYSINIT:
                SETC    INTM
                CLRC    SXM
                CLRC    OVM
                CLRC    CNF                 ;B0  被配置为数据存储空间
                LDP     #0E0H
                SPLK    #81FEH,SCSR1        ;CLKIN=6 M,CLKOUT=24 M
                SPLK    #0E8H,WDCR          ;不使能 WDT
                LDP     #0
                SPLK    #0000H,IMR          ;不使能
                SPLK    #0FFFFH,IFR         ;清全部中断标志
                RET
;（3）      软件延时程序
DELAY:
                MAR     *,AR4
                LAR     AR4,#0FFFEH
                LAR     AR0,#00H
DELAY1:         SBRK    #1
                NOP
                CMPR    00
                BCND    DELAY1,NTC
                RET
```

实验四　　按键计数器

一．实验说明

在控制电路中，通常需要以按键来控制程序执行流程或是输入数据。在图 4.1 中，4 个按键 K_1~K_4 分别对应 TMS320LF2407 芯片的引脚 IOPF3~IOPF4 作为 I/O 端口的输入，8 只发光二极管 LED1~I LED8 通过 SW-DIP8 拨码开关和 74HC273 锁存器芯片分别对应 TMS320LF2407 芯片的引脚 IOPF0~IOPF7 作为 I/O 端口的输出。本实验仅使用一条 I/O 引脚，

借助软件查询方法点亮 8 只发光二极管。

本实验箱中的 K1 键对应的 TMS320LF2407 输入 I/O 引脚为 IOPF3，当按下 K1 键，则将所对应的端口 F 数据和方向控制寄存器（PFDATDIR）的第 3 位为（IOPF3 引脚）0，同时点亮发光二极管。

二．实验目的

1．掌握按键的工作原理、按键和 TMS320LF2407 芯片的接口技术以及按键输入程序的设计和调试方法。

2．掌握输出端发光二极管的工作原理、显示的信息与程序的设计和调试方法。

三．实验内容

1．设计并调试用于 TMS320LF2407 芯片的计数程序，要求由按键 K1 作输入并对其进行计数，计数的结果由 LED7~LED0 发光二极管以二进制方式显示。

2．对程序稍作改动，用 K4 按键完成上述功能。

四．实验硬件电路

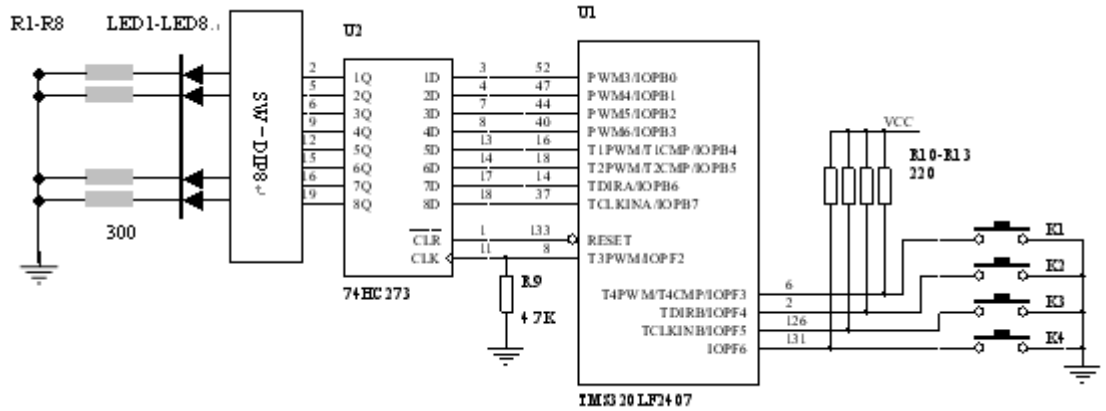


图 4.1 TMS320LF2407 与键盘、LED 接口电路

五．实验参考程序清单

通过编程，设计一个按键计数器，要求刚接通电源时，8 只发光二极管都不亮，表示计数器的初始值为 0，即二进制数的 00000000B；当按下 K1 键时，计数器的值加 1，发光二极管 LED0 点亮，表示二进制数的 00000001B，然后松开按键；再次按下 K1 键时，计数器的值又加 1，发光二极管 LED1 点亮，表示二进制数的 00000010B，然后再松开按键；依次类推。直到按动了 255 次按键时，发光二极管 LED7~LED0 会全部点亮，其后的一次 K1 键按下时将使计数器回 0，就这样循环往复。

C 语言程序

（1） 所需的复位和中断向量定义文件“vectors.asm”同于前。在这以后的所有例程中，如果没有特别说明，vectors.asm 都是相同的。

(2) 主程序

源程序代码：

```
#include    "register.h"
int m=0x0001;
initial()
{
      asm("       setc   SXM");
      asm("       clrc   OVM");
```

```
        asm("         clrc   CNF");

        *SCSR1=0x81FE;
        *WDCR=0x0E8;
        *IMR=0x0000;
        *IFR=0x0FFFF;
        *MCRA=*MCRA&0x0FF;
        *PFDATDIR=*PFDATDIR|0x0400;
        *PBDATDIR=*PBDATDIR|0x0FF00;
        *PFDATDIR=*PFDATDIR|0x0404;
        *PFDATDIR=*PFDATDIR&0xFFFB;

}


void inline disable()
{
        asm("         setc INTM");
}

int keyscan()
{
        int k,j;
        k=*PFDATDIR&0x0008;
        if(k==0x0008)
        k=0;
        else
        k=1;
        if(k==1)
        {
        for(j=30000;j>0;j--)
        j=j;
        k=*PFDATDIR&0x0008;
        if(k==0x0008)
        k=0;
        else
        k=1;
        }
        return(k);
}

int keyserve()
{
        int k;
```

```
        k=*PFDATDIR&0x0008;
        if(k==0x0000)
        *PBDATDIR=(*PBDATDIR&0xFF00)+m++;
        else
        *PBDATDIR=*PBDATDIR;
        *PFDATDIR=*PFDATDIR|0x0404;
        *PFDATDIR=*PFDATDIR&0xFFFB;
}

main()
{
        disable();
        initial();
        while(1)
        {
        int i;
        i=0;
        i=keyscan();
        if(i==1)
        keyserve();
        }
}

void interrupt nothing()
{
        return;
}
```

汇编程序

```
;键盘与发光二极管配合使用程序
st0_temp        .usect      ".b20",1            ;60
st1_temp        .usect      ".b20",1            ;61
context         .usect      ".b20",7            ;62-68
STACK           .usect      ".stack",40

IOSFT_REG       .usect      ".data0",1          ;显示数据移位寄存器
IO_COUNT        .usect      ".data0",1          ;延时计数寄存器
IO_DATA         .usect      ".data0",1          ;I/O 临时数据缓冲区
LEDXS           .usect      ".data0",1          ;LED 显示的数据
LEDFLAG         .usect      ".data0",1          ;LED 显示标志寄存器
K1FLAG          .usect      ".data0",1          ;K1  标志寄存器
KEYDATA         .usect      ".data0",1          ;读得键盘值存放寄存器
DP_USER         .set        5
                .include    "F2407REGS.H"       ;引用头部文件
```

```
                .def        _c_int0
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（1）建立中断向量表
                .sect       ".vectors"      ;定义主向量段
RSVECT          B           _c_int0         ;PM 0       Reset Vector        1
INT1            B           PHANTOM         ;PM 2       Int level 1   4
INT2            B           GISR2           ;PM 4       Int level 2   5
INT3            B           PHANTOM         ;PM 6       Int level 3   6
INT4            B           PHANTOM         ;PM 8       Int level 4   7
INT5            B           PHANTOM         ;PM A       Int level 5   8
INT6            B           PHANTOM         ;PM C       Int level 6   9
RESERVED        B           PHANTOM         ;PM E       (Analysis Int)      10
SW_INT8         B           PHANTOM         ;PM 10      User S/W int        —
SW_INT9         B           PHANTOM         ; PM 12    User S/W int         -
SW_INT10        B           PHANTOM         ; PM 14    User S/W int         -
SW_INT11        B           PHANTOM         ; PM 16    User S/W int         -
SW_INT12        B           PHANTOM         ; PM 18    User S/W int         -
SW_INT13        B           PHANTOM         ; PM 1A    User S/W int         -
SW_INT14        B           PHANTOM         ; PM 1C    User S/W int         -
SW_INT15        B           PHANTOM         ; PM 1E    User S/W int         -
SW_INT16        B           PHANTOM         ; PM 20    User S/W int         -
TRAP            B           PHANTOM         ; PM 22    Trap vector  -
NMI             B           PHANTOM         ; PM 24    Non maskable Int3
EMU_TRAP        B           PHANTOM         ; PM 26    Emulator Trap2
SW_INT20        B           PHANTOM         ; PM 28    User S/W int         -
SW_INT21        B           PHANTOM         ; PM 2A    User S/W int         -
SW_INT22        B           PHANTOM         ; PM 2C    User S/W int         -
SW_INT23        B           PHANTOM         ; PM 2E    User S/W int         -
SW_INT24        B           PHANTOM         ; PM 30    User S/W int         -
SW_INT25        B           PHANTOM         ; PM 32    User S/W int         -
SW_INT26        B           PHANTOM         ; PM 34    User S/W int         -
SW_INT27        B           PHANTOM         ; PM 36    User S/W int         -
SW_INT28        B           PHANTOM         ; PM 38    User S/W int         -
SW_INT29        B           PHANTOM         ; PM 3A    User S/W int         -
SW_INT30        B           PHANTOM         ; PM 3C    User S/W int         -
SW_INT31        B           PHANTOM         ;PM 3E    User S/W int—
;中断子向量入口定义 pvecs
                .sect       ".pvecs"        ;定义子向量段
PVECTORS        B           PHANTOM         ;保留向量地址偏移量-0000h
                B           PHANTOM         ;保留向量地址偏移量-0001h
                B           PHANTOM         ; 保留向量地址偏移量
                B           PHANTOM         ; 保留向量地址偏移量
                B           PHANTOM         ; 保留向量地址偏移量
                B           PHANTOM         ; 保留向量地址偏移量
```

```
B       PHANTOM         ; 保留向量地址偏移量-05
B       PHANTOM         ; SCI_RX_ISR; 保留向量地址偏移量
B       PHANTOM         ; SCI_TX_ISR  ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-0A
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-10
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-15
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-1A
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-20
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-25
B       PHANTOM         ; 保留向量地址偏移量-0026h
B       T1GP_ISR        ; 保留向量地址偏移量-0027h   T1PINT 中断
B       PHANTOM         ; 保留向量地址偏移量-0028h
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-2A
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; pvector addr offset 0x02f - T3PINT
B       PHANTOM         ; 保留向量地址偏移量-30
```

```
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量-35
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量-3A
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量
            B        PHANTOM        ; 保留向量地址偏移量-3F
            B        PHANTOM        ; CANMBX_ISR; 保留向量地址偏移量
            B        PHANTOM        ;保留向量地址偏移量-0041h
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（2）主程序
        .text
_c_int0
            CALL     SYSINIT        ;调系统初始化程序
            CALL     KEYLEDINIT     ;调键盘和 LED 初始化程序
            LDP      #DP_USER       ;指向 0280h～0300h 区
            SPLK     #01H,IOSFT_REG ;寄存器和标志初始化
            SPLK     #00H,IO_COUNT
            SPLK     #001H,IO_DATA
            SPLK     #001H,LEDXS
            SPLK     #00H,LEDFLAG   ;LEDFLAG.0=1 表示  K1,K2,K3  按下
                                    ;LEDFLAG.0=0 表示  K4  按下
            SPLK     #01H,K1FLAG
            CLRC     INTM           ;开总中断
LOOP:       CALL     KEY            ;调键盘程序,即扫描键盘
            LDP      #DP_USER
            BIT      LEDFLAG,15
            BCND     LEDBD1,TC
            LACL     IO_COUNT
            SUB      #03E8H
            BCND     WAIT,LEQ       ;判 10 s 延时到否
LEDBD1:     SPLK     #00H,IO_COUNT
            LDP      #DP_PF2
            LACL     PFDATDIR
            OR       #0404H         ;IOPF2=1
            SACL     PFDATDIR       ;开 74HC273 片选信号
```

```
                LDP     #DP_USER
                LACL    LEDXS
                OR      #0FF00H         ;IOPB 口为输出方式
                LDP     #DP_PF2
                SACL    PBDATDIR        ;送要显示的数据到 IOPB 口
                LACL    PFDATDIR
                AND     #0FFFBH         ;IOPF2=0
                SACL    PFDATDIR        ;关 74HC273 片选信号
                LDP     #DP_USER
                BIT     LEDFLAG,15
                BCND    LEDBD2,TC
                LACL    LEDXS
                SFL
                SACL    LEDXS
                LACL    IOSFT_REG
                SFL
                SACL    IOSFT_REG
                BIT     IOSFT_REG,BIT8  ;判是否循环完一次
                BCND    LOOP1,TC
                B       WAIT
LOOP1:          LDP     #DP_USER
                SPLK    #01H,IOSFT_REG  ;赋初值
LEDBD2:         LACL    IO_DATA
                SACL    LEDXS
WAIT:           NOP
                B       LOOP
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（3）系统初始化程序
SYSINIT:
                SETC    INTM
                CLRC    SXM
                CLRC    OVM
                CLRC    CNF             ;B0 区被配置为数据空间
                LDP     #0E0H           ;指向 7000h～7080h 区
                SPLK    #81FEH,SCSR1     ;时钟 4 倍频,CLKIN=6 M,CLKOUT=24 M
                SPLK    #0E8H,WDCR      ;不使能 WDT
                LDP     #0
                SPLK    #02H,IMR        ;使能中断优先级 INT2
                SPLK    #0FFFFh,IFR     ;清中断标志
                LDP     #DP_EVA         ;指向 7400h～7480h 区
                SPLK    #80H,EVAIMRA    ;使能 T1PINT 中断
                SPLK    #0FFFFh,EVAIFRA ;清 EVA 中断标志
                SPLK    #0,GPTCONA
                SPLK    #0EA6H,T1PR     ;使定时器每 10 ms 产生一次中断
```

```
                SPLK    #0,T1CNT
                SPLK    #0164CH,T1CON     ;设置通用定时器 1
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（4）键盘和发光二极管初始化程序
KEYLEDINIT:
                LDP     #DP_PF2
                LACL    MCRC
                AND     #083FFH           ;IOPF2,IOPF[3～6] 配置为一般的 I/O 口
                SACL    MCRC
                LACL    MCRA
                AND     #000FFH           ;IOPB[0～7] 配置为一般的 I/O 口
                SACL    MCRA
                LACL    PFDATDIR
                OR      #0400H            ;IOPF2 为输出方式
                AND     #08787H           ;IOPF[3～6] 为输入方式
                SACL    PFDATDIR
                LACL    PBDATDIR
                OR      #0FF00H           ;IOPB[0～7] 为输出方式
                SACL    PBDATDIR
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（5）键盘程序
KEY:
                CALL    READKEY           ;调读键程序
                LACL    KEYDATA
                BCND    KEYRET,EQ         ;ACC=0?
                CALL    KEYDELAY          ;延时消抖动
                CALL    READKEY           ;再一次读键值
                LACL    KEYDATA
                BCND    KEYRET,EQ
                LDP     #DP_USER          ;判断按键情况
                BIT     KEYDATA,15
                BCND    KEYRET,NTC
                CALL    K1                ; "+"键按下
                B       KEYRET
KEYRET:         LACL    KEYDATA
                AND     #0FFF0H           ;清读取的键值寄存器
                SACL    KEYDATA
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（6）读键子程序
READKEY:
                LDP     #DP_PF2
```

```
                LACL    PFDATDIR        ;取出键值 PFDATDIR.3～PFDATDIR.6
                RPT     #2
                SFR                     ;右移 3 位
                OR      #0FFF0H         ;屏蔽高 4 位（用到 4 个键）
                CMPL
                LDP     #DP_USER
                SACL    KEYDATA         ;存放键值
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（7）用软件延时 30mS 消抖动
KEYDELAY:
                LACC    #6000
KEYD1:          SUB     #1
                RPT     #80
                NOP
                BCND    KEYD1,NEQ
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（8）键子程序
K1:                                     ;"+"键子程序
READK1:         CALL    KEYDELAY
                CALL    READKEY
                LDP     #DP_USER        ;判断按键  K1  是否松开
                BIT     KEYDATA,15
                BCND    READK1,TC
                SPLK    #01,LEDFLAG     ;关闭 LED 左移标志,即 LED 对同一个数不刷新
                LACL    IO_DATA
                ADD     #1
                SACL    IO_DATA
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（9）中断程序
GISR2:                                  ;优先级 INT2 中断人口
; 保护现场
                LDP     #0              ; 保存机器上下文
                SST     #0, st0_temp    ; 使用自动寻址 DP-0
                SST     #1, st1_temp    ; 保存状态寄存器到 B2 DARAM.
                SACL    context         ; 保存 ACC 的低 16 位
                SACH    context+1       ; 保存 ACC 的高 16 位
                SAR     AR1,context+2
                SAR     AR2,context+3
                SAR     AR3,context+4
                SAR     AR4,context+5
                SAR     AR5,context+6
```

```
                LDP     #0E0H
                LACC    PIVR,1          ;读取外设中断向量寄存器（PIVR）,并左移一位
                ADD     #PVECTORS       ;加上外设中断入口地址
                BACC                    ;跳到相应的中断服务子程序
T1GP_ISR:                               ;通用定时器 1 中断入口
                LDP     #DP_USER
                LACL    IO_COUNT
                ADD     #1
                SACL    IO_COUNT
;  恢复现场
                LDP     #DP_EVA
                SPLK    #0FFFFH,EVAIFRA
                LDP     #0
                LAR     AR5,context+6
                LAR     AR4,context+5
                LAR     AR3,context+4
                LAR     AR2,context+3
                LAR     AR1,context+2
                LACC    context+1,16
                ADDS    context
                LST     #1, st1_temp
                LST     #0, st0_temp
                CLRC    INTM            ;开总中断,因为一进中断就自动关闭总中断
                RET
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;（10）假中断程序
PHANTOM
KICK_DOG                                ;复位看门狗
                RET
                END
```

实验五　键盘和 LED 发光二极管显示电路

一．实验说明

在 DSP 的应用系统中，为实现人－－机对话，显示和键盘是两个必不可少的功能配置。在本实验程序设计中，用键盘的响应作 I/O 端口的输入，用 I/O 端口的输出点亮发光二极管，形成的键盘、LED 电路如图 4.1 所示。

二．实验目的

1． 学会配合使用键盘与发光二极管的编程方法。

2． 进一步认识 TMS320LF2407 芯片的 I/O 端口的功能及控制方法。

三．实验内容

1．用 TMS320LF2407 芯片设计并调试一个能使 8 只发光二极管按 8 种花样显示。

2．给按键定义各种不同功能，以实现对发光二极管循环顺序和点亮个数的控制。

四．实验参考程序清单

通过编程，使 8 只发光二极管实现如下显示功能：按下 K1 键，可点亮 8 种不同组合的发光二极管花样模式显示，按 8 次为一个循环周期；当按下一次 K2 键，将使设定的发光二极管显示的个数加 1；当按下一次 K3 键，将使设定的发光二极管的位置往右移 1 位；当按下 K4 键或不按键时，返回到循环点亮发光二极管模式。

用通用定时器 1 产生 10ms 作为时基，用中断编程方法实现延时，再用软件计数得到 10s 延时。

C 程序

I/O 端口作为输入和输出使用

源程序代码：

（2） 所需的复位和中断向量定义文件“vectors.asm”同于前。在这以后的所有例程中，如果没有特别说明，vectors.asm 都是相同的。

（3） 主程序

```
// 该程序用于键盘的识别，按键情况通过 LED 表示.
#include   "register.h"
// 初始化子程序
initial()
{
      asm(" setc  SXM");                    // 抑制符号位扩展
      asm(" clrc  OVM");                    // 累加器中结果正常溢出
      asm(" clrc  CNF");                    // B0 被配置为数据存储空间
      *SCSR1=0x81FE;                        // CLKIN=6M，CLKOUT=4*CLKIN=24M
      *WDCR=0x0E8;                          // 不使能看门狗，因为 SCSR2 中的 WDOVERRIDE
                                            // 即 WD 保护位复位后的缺省值为 1，故可以用
                                            // 软件禁止看门狗
      *IMR=0x0000;                          // 禁止所有中断
      *IFR=0x0FFFF;                         // 清除全部中断标志，"写 1 清 0"
      *MCRA=*MCRA&0x0FF;                    // IOPB 端口配置为一般的 I/O 功能
      *PBDATDIR=*PBDATDIR|0x0FF00;          // IOPB 端口设置为输出方式
      *MCRC=*MCRC&0x03FF;                   // IOPF2 端口和 IOPF3~6 配置为一般的 I/O 功能
      *PFDATDIR=*PFDATDIR|0x0400;           // IOPF2 端口为输出端口，IOPF3-IOPF6 为输入端口
      *PBDATDIR=*PBDATDIR&0x0FF00;          // 熄灭全部的 LED 灯
      *PFDATDIR=*PFDATDIR|0x0404;           // IOPF2  设置为输出方式，且 IOPF2=1
      *PFDATDIR=*PFDATDIR&0x0FFFB;          // IOPF2=0
// 以上的操作产生一个脉冲，使 LED 全部熄灭
}
//  屏蔽中断子程序
void   inline   disable()
{
      asm(" setc INTM");
}
int   keyscan( )
{
      int k，j;                             // 定义局部变量
```

```
    k=*PFDATDIR&0x0078；                    // 读入键盘状态并屏蔽掉相应的位
    if(k == 0x0078)     k=0；
    else   k=1；                            // 有键按下，则 k=1
    if(k == 1)                              // 若无键按下，则直接返回
    {
        for(j=30000；j>0；j--)   j=j；       // 若有键按下，则延时消抖动
        k=*PFDATDIR&0x0078；                // 读入键盘状态并屏蔽掉相应的位
        if(k == 0x0078)     k=0；
        else   k=1；                        // 有还有键按下，则 k=1
    }
    return(k)；                             // 返回 K 值
}
int   keyserve( )                           // 键服务子程序
{
    int    k；                              // 定义局部变量
    k=*PFDATDIR&0x0078；                    // 读入键盘状态并屏蔽掉相应的位
    switch(k)
    {
        case 0x0070：*PBDATDIR=(*PBDATDIR&0xFF00)|0X0001；break；
    // 若按下 K1 键，则显示"1"
        case 0x0068：*PBDATDIR=(*PBDATDIR&0xFF00)|0X0002；break；
    // 若按下 K2 键，则显示"2"
        case 0x0058：*PBDATDIR=(*PBDATDIR&0xFF00)|0X0003；break；
    // 若按下 K3 键，则显示"3"
        case 0x0038：*PBDATDIR=(*PBDATDIR&0xFF00)|0X0004；break；
    // 若按下 K4 键，则显示"4"
        default：*PBDATDIR=*PBDATDIR；
    }
    *PFDATDIR=*PFDATDIR|0x0404；            // IOPF2  设置为输出方式，且 IOPF2=1
    *PFDATDIR=*PFDATDIR&0x0FFFB；           // IOPF2=0
// 以上给一个脉冲，使 B 端口的值显示出来
}
main( )
{
    disable( )；                            // 屏蔽所有中断
    initial( )；                            // 系统初始化
    while(1)
    {
        int i；                             // 定义局部变量
        i=0；
        i=keyscan()；                       // 键盘扫描，若有键按下，则返回值为"1"，否则返回值为"0"
        if(i==1)     keyserve()；           // 如果有键按下，则进行键服务程序
    }
}
```

```
// 直接返回中断服务子程序
void interrupt nothing()
{
    return;
}
```

汇编程序

```
st0_temp        .usect      ".b20",1            ;60
st1_temp        .usect      ".b20",1            ;61
context         .usect      ".b20",7            ;62-68
STACK    .      usect       ".stack",40

IOSFT_REG       .usect      ".data0",1          ;280      ; IOPB SHIFT REG
IO_COUNT        .usect      ".data0",1          ;281      ; IO COUNT REG
IO_DATA         .usect      ".data0",1          ;282      ; I/O 临时数据缓冲区
LEDXS           .usect      ".data0",1          ;283      ; LED 显示的数据
LEDFLAG         .usect      ".data0",1          ;284      ; LED 显示标志
K1FLAG          .usect      ".data0",1          ;285      ; K1  标志寄存器
KEYDATA.        usect       ".data0",1          ;286
KEYTIMER        .usect      ".data0",1          ;287
DP_USER         .set        5

                .include    "F2407REGS.H"
                .def        _c_int0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .sect       ".vectors"          ;定义主向量段
RSVECT          B           _c_int0             ; PM 0    Reset Vector          1
INT1            B           PHANTOM             ;GISR1    ; PM 2    Int level 1  4
INT2            B           GISR2               ; PM 4    Int level 2      5
INT3            B           PHANTOM             ; PM 6    Int level 3      6
INT4            B           PHANTOM             ; PM 8    Int level 4      7
INT5            B           PHANTOM             ;GISR5; PM A Int level 5   8
INT6            B           PHANTOM             ; PM C    Int level 6      9
RESERVED        B           PHANTOM             ; PM E    (Analysis Int)   10
SW_INT8         B           PHANTOM             ; PM 10   User S/W int     -
SW_INT9         B           PHANTOM             ; PM 12   User S/W int     -
SW_INT10        B           PHANTOM             ; PM 14   User S/W int     -
SW_INT11        B           PHANTOM             ; PM 16   User S/W int     -
SW_INT12        B           PHANTOM             ; PM 18   User S/W int     -
SW_INT13        B           PHANTOM             ; PM 1A   User S/W int     -
```

```
SW_INT14    B       PHANTOM         ; PM 1C   User S/W int     -
SW_INT15    B       PHANTOM         ; PM 1E   User S/W int     -
SW_INT16    B       PHANTOM         ; PM 20   User S/W int     -
TRAP        B       PHANTOM         ; PM 22   Trap vector-
NMI         B       PHANTOM         ; PM 24   Non maskable Int3
EMU_TRAP    B       PHANTOM         ; PM 26   Emulator Trap    2
SW_INT20    B       PHANTOM         ; PM 28   User S/W int     -
SW_INT21    B       PHANTOM         ; PM 2A   User S/W int     -
SW_INT22    B       PHANTOM         ; PM 2C   User S/W int     -
SW_INT23    B       PHANTOM         ; PM 2E   User S/W int     -
SW_INT24    B       PHANTOM         ; PM 30   User S/W int     -
SW_INT25    B       PHANTOM         ; PM 32   User S/W int     -
SW_INT26    B       PHANTOM         ; PM 34   User S/W int     -
SW_INT27    B       PHANTOM         ; PM 36   User S/W int     -
SW_INT28    B       PHANTOM         ; PM 38   User S/W int     -
SW_INT29    B       PHANTOM         ; PM 3A   User S/W int     -
SW_INT30    B       PHANTOM         ; PM 3C   User S/W int     -
SW_INT31    B       PHANTOM         ; PM 3E   User S/W int     -

;=============================================================
;中断子向量入口定义 pvecs
;=============================================================
            .sect   ".pvecs"        ;定义子向量段
PVECTORS    B       PHANTOM         ; 保留向量地址偏移量-00
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量-05
            B       PHANTOM         ;SCI_RX_ISR   ; 保留向量地址偏移量
            B       PHANTOM         ;SCI_TX_ISR   ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量-0a
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量-10
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
```

```
B       PHANTOM         ; 保留向量地址偏移量-15
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-1a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-20
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-25
B       PHANTOM         ; 保留向量地址偏移量
B       T1GP_ISR        ; 保留向量地址偏移量-0027 T1PINT
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-2a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; pvector addr offset 0x02f - T3PINT
B       PHANTOM         ; 保留向量地址偏移量-30
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-35
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-3a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-3f
B       PHANTOM         ;CANMBX_ISR ; 保留向量地址偏移量
```

```
                B       PHANTOM         ; 保留向量地址偏移量-41

;程序段
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; 引脚定义：键盘 K1=IOPF3，K2=IOPF4， K3=IOPF5, K4=IOPF6, LED 的片选信号:IOP
; 此程序为 4 个键盘与 8 个发光二极管组合程序，再无键按下或 K4 按下时用定时器 1 的方式，循环点亮 8 个发光二极管
; K1 按下则为功能模式，按键的次数不同显示的发光二极管也不同
; K2 按下则为“＋”键模式，按一次加一，点亮不同的发光二极管
; K3 按下则为移位模式，按一次往右移一位，即 LED7=LED6....LED1=LED0, LED0=LED7
; 注意 LED 显示时要关闭 LCD 的块选 E1 和 E2
                .text
_c_int0
                CALL    INIT                ;;;;
                CALL    KEYLEDINIT

                LDP     #DP_USER
                SPLK    #01H,IOSFT_REG      ;显示数据移位寄存器
                SPLK    #00H,IO_COUNT       ;延时计数寄存器
                SPLK    #001H,IO_DATA       ;I/O 临时数据缓冲区
                SPLK    #001H,LEDXS         ;LED 显示的数据
                SPLK    #00H,LEDFLAG        ; LED 显示标志寄存器 LEDFLAG.0=1 表示
                                            ; K1,K2,K3 按下
                                            ; LEDFLAG.0=0 表示 K4 按下
                SPLK    #01H,K1FLAG         ;K1 标志寄存器
                CALL    INIT_TIME1

LOOP:
                CALL    KEY
                LDP     #DP_USER
                BIT     LEDFLAG,15
                BCND    LEDBD1,TC
                LACL    IO_COUNT
                SUB     #030H
                BCND    WAIT,LEQ

LEDBD1:
                SPLK    #00H,IO_COUNT

                LDP     #DP_PF2
                LACL    PFDATDIR
                OR      #0004H              ; IOPF2 输出 1
                SACL    PFDATDIR            ;开 74HC273 片选信号
```

```
        LDP     #DP_USER
        LACL    LEDXS
        OR      #0FF00H         ; IOPB  是输出
        LDP     #DP_PF2
        SACL    PBDATDIR        ;  送要显示的数据

        LDP     #DP_PF2
        LACL    PFDATDIR
        AND     #0FFFBH         ;IOPF2  输出 0
        SACL    PFDATDIR        ;关 74HC273 片选信号

        LDP     #DP_USER
        BIT     LEDFLAG,15
        BCND    LEDBD2,TC
        LACL    LEDXS
        SFL                     ;要显示的数左移一位
        SACL    LEDXS
        LACL    IOSFT_REG
        SFL
        SACL    IOSFT_REG
        BIT     IOSFT_REG,BIT8  ;  判是否循环完一次
        BCND    LOOP1,TC
        B       WAIT

LOOP1:
        LDP     #DP_USER
        SPLK    #01H,IOSFT_REG
LEDBD2:
        LACL    IO_DATA
        SACL    LEDXS
WAIT:

        NOP
        B       LOOP


INIT:                           ;系统初始化
        SETC    INTM            ;关全局中断
        CLRC    SXM             ;抑止符号扩展
        CLRC    OVM
        CLRC    CNF             ; B0  配置为数据空间

        LDP     #0E0H
        SPLK    #81FEH,SCSR1    ; CLKIN=6M,CLKOUT=24M
```

```
        SPLK    #0E8h,WDCR      ; 关闭看门狗 WDT
        LDP     #0
        SPLK    #0002h,IMR      ; 使能 interrupt 2
        SPLK    #0FFFFh,IFR     ; 清中断标志
        RET

INIT_TIME1:                     ;定时器 1 初始化子程序
        LDP     #DP_EVA
        SPLK    #80H,EVAIMRA    ;开定时器 1 周期中断
        SPLK    #0FFFFh,EVAIFRA;清 EVA 所有的中断标志(写 1 清除)
        SPLK    #0,GPTCONA      ;不启动 AD,禁止定时器 1 比较输出
        SPLK    #0BBH,T1PR      ;使定时器每 1ms 产生一次中断
        SPLK    #00H,T1CNT      ;清定时器 1 计数单元
        SPLK    #0174CH,T1CON   ;TMODE=10(连续增模式),
                                ;TPS=111 X/128,,CLOCK=Internal, 启动定时器 1
        CLRC    INTM
        RET

KEYLEDINIT:                     ;键盘 LED 初始化子程序
        LDP     #DP_PF2
        LACL    MCRC
        AND     #083FFH         ; IOPF2--IOPF6 配置为一般的 I/O 口
        SACL    MCRC

        LACL    MCRA
        AND     #000FFH         ; IOPB[0-7] 配置为一般的 I/O 口
        SACL    MCRA

        LACL    PFDATDIR
        OR      #0400H          ; IOPF2 为输出方式
        AND     #087FBH         ;IOPF[3-6] 为输入方式
        SACL    PFDATDIR

        LACL    #0FF00H         ;IOPB[0-7] 为输出方式
        SACL    PBDATDIR

        RET
;*********************************************************************
KEY:
        CALL    READKEY         ;读键盘值
        LACL    KEYDATA
        BCND    KEYRET,EQ       ; ACC=0?
        CALL    KEYDELAY        ; KEYDATA 不等于 0,表示有键按下,延时消抖动
        CALL    READKEY         ;键盘软件延时子程序
```

```
                BCND    KEYRET,EQ
                LDP     #DP_USER        ;判断按键情况(考虑只有一个键按下)
                BIT     KEYDATA,15
                BCND    KEY1,NTC
                CALL    K1              ; KEY1 功能键
                B       KEYRET
KEY1:
                BIT     KEYDATA,14
                BCND    KEY2,NTC
                CALL    K2              ;KEY2"+"键
                B       KEYRET
KEY2:
                BIT     KEYDATA,13
                BCND    KEY3,NTC
                CALL    K3              ;KEY3 "移位"键
                B       KEYRET
KEY3:
                BIT     KEYDATA,12      ;KEY4  确认键
                BCND    KEYRET,NTC
                CALL    K4
KEYRET:
                LACL    KEYDATA         ;清 KEYDATA 的低 4 位
                AND     #0fff0h
                SACL    KEYDATA
                RET
;*****************读键盘子程序*******************************************
READKEY:                                ；判键盘状态子程序
                LDP     #DP_PF2         ; DP-->7080h-70FFh
                LACL    PFDATDIR        ；取出键值 PFDATDIR.3--PFDATDIR.6
                RPT     #2
                SFR                     ;右移 3 位
                OR      #0fff0h         ；屏蔽高 4 位（用到 4 个键）
                CMPL                    ;有键盘按下时对应位为低,按位取反之后为高
                LDP     #DP_USER
                SACL    KEYDATA         ;键值保存在 KEYDATA 中
                RET
KEYDELAY:
                LACC    #6000           ；键盘延时子程序  86ms(20000) 30mS(6000)
KEYD1:
                SUB     #1              ；延时 30mS 消抖动
                RPT     #080H
                NOP
                BCND    KEYD1, NEQ
                LDP     #5
```

```
                RET
; ********************  功能键子程序  ******************************
K1:
                NOP
READK1:
                CALL    READKEY
                CALL    KEYDELAY
                LDP     #DP_USER            ；判断按键 K1 是否松开
                BIT     KEYDATA,15
                BCND    READK1,TC
                LDP     #DP_USER            ；可扩展成 16 个或更多分支
                SPLK    #01H,LEDFLAG
                BIT     K1FLAG,15
                BCND    GN_KEY1,TC
                BIT     K1FLAG,14
                BCND    GN_KEY2,TC
                BIT     K1FLAG,13
                BCND    GN_KEY3,TC
                BIT     K1FLAG,12
                BCND    GN_KEY4,TC
                BIT     K1FLAG,11
                BCND    GN_KEY5,TC
                BIT     K1FLAG,10
                BCND    GN_KEY6,TC
                BIT     K1FLAG,9
                BCND    GN_KEY7,TC
                BIT     K1FLAG,8
                BCND    GN_KEY8,TC
                B       K1_RET
GN_KEY1:
                SPLK    #01H,IO_DATA        ;K1 按第 1 次时显示 1H
                B       K1_LOOP1
GN_KEY2:
                SPLK    #03H,IO_DATA        ;K1 按第 2 次时显示 3H
                B       K1_LOOP1
GN_KEY3:
                SPLK    #05H,IO_DATA        ;K1 按第 3 次时显示 5H
                B       K1_LOOP1
GN_KEY4:
                SPLK    #007H,IO_DATA       ;K1 按第 4 次时显示 7H
                B       K1_LOOP1
GN_KEY5:
                SPLK    #11H,IO_DATA        ;K1 按第 5 次时显示 11H
                B       K1_LOOP1
```

```
GN_KEY6:
        SPLK    #33H,IO_DATA        ;K1 按第 6 次时显示 33H
        B       K1_LOOP1
GN_KEY7:
        SPLK    #55H,IO_DATA        ;K1 按第 7 次时显示 55H
        B       K1_LOOP1
GN_KEY8:
        SPLK    #77H,IO_DATA        ;K1 按第 8 次时显示 77H
        B       K1_LOOP1

K1_LOOP1:
        LACL    K1FLAG
        SFL                         ;K1FLAG 左移一位
        SACL    K1FLAG
        BIT     K1FLAG,7
        BCND    K1_LOOP2,TC
        B       K1_RET

K1_LOOP2:
        SPLK    #01H,K1FLAG
K1_RET:
        NOP
        RET

;======= "+"键子程序 ==========================
K2:
        NOP
READK2:
        CALL    READKEY
        CALL    KEYDELAY
        LDP     #DP_USER            ; 判断按键 K2 是否松开
        BIT     KEYDATA,14
        BCND    READK2,TC

        LDP     #DP_USER
        SPLK    #01,LEDFLAG         ; 关闭 LED 左移标志,即 LED 对同一个数不刷新
        LACL    IO_DATA
        ADD #1
        SACL    IO_DATA
        RET

;======= "移位"键子程序 ==========================
K3:
        NOP
```

```
READK3:
        CALL    READKEY
        CALL    KEYDELAY
        LDP     #DP_USER            ; 判断按键 K3 是否松开
        BIT     KEYDATA,13
        BCND    READK3,TC

        LDP     #DP_USER
        SPLK    #01H,LEDFLAG
        LACL    IO_DATA
        SFR
        SACL    IO_DATA
        BCND    K3_LOOP1,NC         ; C=1 则 IO_DATA+80H
        LACL    #80H
        ADD     IO_DATA

K3_LOOP1:
        SACL    IO_DATA

        RET

;======= 确认键子程序 ============================
K4:
        NOP
READK4:
        CALL    READKEY
        CALL    KEYDELAY
        LDP     #DP_USER            ; 判断按键 K4 是否松开
        BIT     KEYDATA,12
        BCND    READK4,TC

        LDP     #DP_USER
        SPLK    #00H,     LEDFLAG;  恢复 LED 左移标志
        SPLK    #01H,     IO_DATA

        RET
;*************************定时器中断子程序*********************************
GISR2:  LDP     #0                  ; 保存机器上下文
        SST     #0, st0_temp        ; 使用自动寻址，DP-0
        SST     #1, st1_temp        ; 保存状态寄存器到 B2 DARAM.
        SACL    context             ; 保存 ACC 的低 16 位
        SACH    context+1           ; 保存 ACC 的高 16 位
        SAR     AR1,context+2
        SAR     AR2,context+3
```

```
        SAR     AR3,context+4
        SAR     AR4,context+5
        SAR     AR5,context+6
        LDP     #0E0h           ; 为 PIVR 控制寄存器改变 DP
        LACC    PIVR,1          ; 读 EVIVRB, 左移 1 位
        ADD     #PVECTORS       ; 加偏移向量到基向量
        BACC

T1GP_ISR:
        LDP     #DP_USER
        LACL    IO_COUNT
        ADD     #1
        SACL    IO_COUNT

GISR2_RET:
        LDP     #DP_EVA
        SPLK    #0FFFFH,EVAIFRA
        LDP     #0
        LAR     AR5,context+6
        LAR     AR4,context+5
        LAR     AR3,context+4
        LAR     AR2,context+3
        LAR     AR1,context+2
        LACC    context+1,16
        ADDS    context
        LST     #1, st1_temp
        LST     #0, st0_temp
        CLRC    INTM
        RET

;*****************************************************************
PHANTOM
        KICK_DOG                ; Resets WD counter
        RET
        END
```

实验六　模 / 数转换器 ADC 应用

一．实验说明

随着应用对象和设计目标的千差万别，对于 ADC 的性能指标提出了各种各样的不同要求，从而形成了市场上大量涌现和种类繁多的以独立形态出现的 ADC。虽然 ADC 这样的模拟接口类器件常常是以独立形态出现的，但是，拥有这类器件生产技术的的制造厂家，越来越多地把这项功能集成到单片机、DSP 芯片内部，以便适应芯片朝着普及化、专用化、片上系统化（SOC）、内部模块种类和数量的可整合性和可裁剪性以及纯单片应用的发展潮流。

在 TMS320LF2407 芯片内配置了带 10 位具有 16 个模拟通道的 ADC 模块。本次实验主要是为了验证 TMS320LF2407 片内 ADC 模块功能，并用 TMS320LF2407 芯片来实现一个单通道的 10 位 A/D 转换，即任选 ADCIN 通道之一作为模拟信号输入端，并将此转换结果以二进制形式经 IOPB 口与 74HC273 锁存器而链接的 8 只发光二极管输出显示。可以看到，转换结果会随着模拟量的变化而变化，从而可以让我们了解片内 ADC 模块的工作情况。

二．实验目的

1． 了解 TMS320LF2407 芯片的 ADC 模块的特性及自动排序功能。

2． 掌握对 ADCIN 端口的引脚功能设置，学习设计和调试 ADC 的应用程序，要求转换结果由中断服务子程序读出。

三．实验内容

1． 选择 A/D 转换通道。首先选择 A/D 转换引脚功能，将相应的 A/D 引脚设置为输入，其次选择 A/D 转换模拟通道。

2． 设计并调试一个能通过 8 只 LED 发光二极管实现二进制显示转换数据的程序。

3． 要求从实验箱 P4 输入 0～＋3.3V 直流电压，**P3** 输入－3.3V～＋3.3V 交流电压

四． 实验硬件电路图

加图见 P4,P3

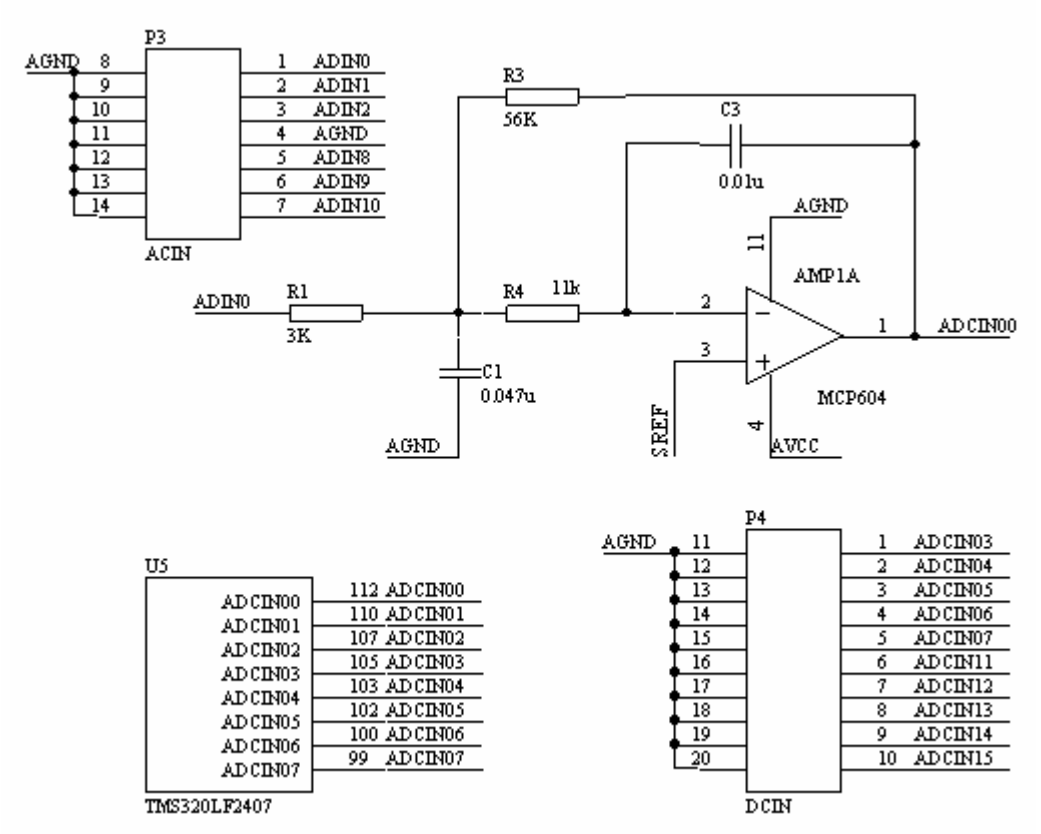


图 ADC 硬件电路

五．实验参考程序清单

C 语言程序

```
#include "register.h"
int ADRESULT[16];                    // 定义一个数组用于保存 AD 转换的结果
volatile unsigned int *j;            // 定义一个指针变量 j
int i=0x00,cesi;
```

```
void inline disable()
{
    asm("    setc INTM");
}

void inline enable()
{
    asm("    clrc INTM");
}

initial()
{
    asm("    setc SXM");              // 符号位扩展有效
    asm("    clrc OVM");              // 累加器中结果正常溢出
    asm("    clrc CNF");              // B0 被配置为数据存储空间
    *SCSR1=0x81EF;                    // CLKIN=6M，CLKOUT=4*CLKIN=24M
    *WDCR=0x0E8;                      // 不使能看门狗，因为 SCSR2 中的 WDOVERRIDE
                                      // 即 WD 保护位复位后的缺省值为 1，故可以用
                                      // 软件禁止看门狗

    *IMR=0x0001;                      // 允许 INT1 中断
    *IFR=0x0FFFF;                     // 清除全部中断标志，"写 1 清 0"
}
// AD 初始化子程序
void ADINIT()
{
    *T4CNT=0x0000;                    // T4 计数器清 0
    *T4CON=0x170C;                    // T4 为连续增计数模式，128 分频，且选用内部时钟源
    *T4PER=0x75;                      // 设置 T4 的周期寄存器
    *GPTCONB=0x400;                   // T4 周期中断标志触发 AD 转换
    *EVBIFRB=0x0FFFF;                 // 清除 EVB 中断标志，写"1"清 0
    *ADCTRL1=0x10;                    // 采样时间窗口预定标位 ACQ PS3-ACQ PS0 为 0，
                                      // 转换时间预定标位 CPS 为 0，AD 为启动/停止模式，排
                                      // 序器为级连工作方式，且禁止特殊的两种工作模式
    *ADCTRL2=0x8404;                  // 可以用 EVB 的一个事件信号触发 AD 转换，
                                      // 且用中断模式 1
    *MAXCONV=0x0F;                    // 16 通道
    *CHSELSEQ1=0x3210;
    *CHSELSEQ2=0x7654;
    *CHSELSEQ3=0x0BA98;
    *CHSELSEQ4=0x0FEDC;               // 转换通道是 0-15
}
// 启动 AD 转换子程序(通过启动定时器 4 的方式间接启动)
```

```
void ADSOC()
{
    *T4CON=*T4CON|0x40;                 // 启动定时器 4
}
// 若是其它中断则直接返回子程序
void interrupt nothing()
{
    return;
}
// AD 中断服务子程序
void interrupt adint()
{
    asm("       clrc INTM");            // 抑制符号位扩展
    j=RESULT0;                          // 取得 RESULT0 的地址
    for(i=0;i<=15;i++,j++)
    {
    ADRESULT[i]=*j>>6;                  // 把 AD 转换的结果左移 6 位后存入规定的数组
    cesi=ADRESULT[i];                   // 检验每个 A/D 转换的结果
    }
    *ADCTRL2=*ADCTRL2|0x4200;           // 复位 SEQ1，且清除 INT FLAG SEQ1 标志写"1"清 0
    enable();                           // 开总中断，因为一进入中断总中断就自动关闭了
}

main()
{
    disable();                          // 禁止总中断
    initial();                          // 系统初始化
    ADINIT();                           // AD 初始化子程序
    enable();                           // 开总中断
    ADSOC();                            // 启动 AD 转换
    while(1)
    {
        if(i==0x10)
        break;                          // 如果已发生中断，则停止等待（发生中断后，i=0x10）
    }                                   // 等待中断发生
    *T4CON=*T4CON&0x0FFBF;              // 停止定时器 4，即间接停止 A/D 转换
    while(1)
    {
    ;
    }                                   // 死循环，在实际的工程应用中在此可以利用 A/D 转换的
                                        // 结果用于一些运算
}
```

汇编程序

```
st0_temp        .usect      ".b20",1            ;60
st1_temp        .usect      ".b20",1            ;61
context         .usect      ".b20",7            ;62-68
STACK           .usect      ".stack",40

ADCOUNT         usect       ".data0",1          ;288        ;;;;;
ADRESULT .      usect       ".data0",1          ;289        ;;;;;

                .include    "F2407REGS.H"       ;引用头部文件
                .def         _c_int0
;（1）建立中断向量表
                .sect       ".vectors"          ;定义主向量段
RSVECT          B           _c_int0             ;PM 0       复位向量            1
INT1            B           GISR1               ;PM 2       中断优先级 1        4
INT2            B           PHANTOM             ;PM 4       中断优先级 2        5
INT3            B           PHANTOM             ;PM 6       中断优先级 3        6
INT4            B           PHANTOM             ;PM 8       中断优先级 4        7
INT5            B           PHANTOM             ;PM A       中断优先级 5        8
INT6            B           PHANTOM             ;PM C       中断优先级 6        9
RESERVED        B           PHANTOM             ;PM E       模拟量输入中断(保留)  10
SW_INT8         B           PHANTOM             ;PM 10      用户定义软件中断      —
SW_INT9         B           PHANTOM             ; PM 12     User S/W int    -
SW_INT10        B           PHANTOM             ; PM 14     User S/W int    -
SW_INT11        B           PHANTOM             ; PM 16     User S/W int    -
SW_INT12        B           PHANTOM             ; PM 18     User S/W int    -
SW_INT13        B           PHANTOM             ; PM 1A     User S/W int    -
SW_INT14        B           PHANTOM             ; PM 1C     User S/W int    -
SW_INT15        B           PHANTOM             ; PM 1E     User S/W int    -
SW_INT16        B           PHANTOM             ; PM 20     User S/W int    -
TRAP            B           PHANTOM             ; PM 22     Trap vector-
NMI             B           PHANTOM             ; PM 24     Non maskable Int3
EMU_TRAP        B           PHANTOM             ; PM 26     Emulator Trap   2
SW_INT20        B           PHANTOM             ; PM 28     User S/W int    -
SW_INT21        B           PHANTOM             ; PM 2A     User S/W int    -
SW_INT22        B           PHANTOM             ; PM 2C     User S/W int    -
SW_INT23        B           PHANTOM             ; PM 2E     User S/W int    -
SW_INT24        B           PHANTOM             ; PM 30     User S/W int    -
SW_INT25        B           PHANTOM             ; PM 32     User S/W int    -
SW_INT26        B           PHANTOM             ; PM 34     User S/W int    -
SW_INT27        B           PHANTOM             ; PM 36     User S/W int    -
SW_INT28        B           PHANTOM             ; PM 38     User S/W int    -
SW_INT29        B           PHANTOM             ; PM 3A     User S/W int    -
SW_INT30        B           PHANTOM             ; PM 3C     User S/W int    -
SW_INT31        B           PHANTOM             ;PM 3E      用户定义软件中断      —
```

```
;中断子向量入口定义 pvecs
                .sect       ".pvecs"            ;定义子向量段
PVECTORS        B           PHANTOM             ;保留向量地址偏移量   0000h
                B           PHANTOM             ;保留向量地址偏移量   0001h
                B           PHANTOM             ;保留向量地址偏移量   0002h
                B           PHANTOM             ;保留向量地址偏移量   0003h
                B           ADCINT_ISR          ;保留向量地址偏移量   0004h   ADC 中断
                B           PHANTOM             ;保留向量地址偏移量   0005h
                B           PHANTOM             ;SCI_RX_ISR; 保留向量地址偏移量
                B           PHANTOM             ;SCI_TX_ISR   ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-0a
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-10
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-15
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-1a
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-20
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-25
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量
                B           PHANTOM             ; 保留向量地址偏移量-2a
```

```
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; pvector addr offset 0x02f - T3PINT
                B       PHANTOM         ; 保留向量地址偏移量-30
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量-35
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量-3a
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量
                B       PHANTOM         ; 保留向量地址偏移量-3f
                B       PHANTOM         ;CANMBX_ISR ; 保留向量地址偏移量
                B       PHANTOM         ;保留向量地址偏移量   0041h
;（2）主程序:
                .text
_c_int0
                SETC    INTM
                CLRC    SXM
                CLRC    OVM
                CLRC    CNF
                LDP     #0E0H
                SPLK    #81FEH,SCSR1    ;CLKIN=6 M,CLKOUT=24 M
                SPLK    #0E8h,WDCR      ;关看门狗
                LDP     #0
                SPLK    #0001h,IMR      ;允许 INT1 中断
                SPLK    #0FFFFh,IFR     ;清所有中断标志
                CALL    ADINIT          ;初始化 ADC 程序
                CLRC    INTM            ;开总中断
                CALL    AD              ;启动模数转换程序
WAIT:           NOP
                B       WAIT
;（3）ADC 初始化程序
ADINIT:
                LDP     #234            ;设置通用定时器 4
```

```
        SPLK    #0000H,T4CNT
        SPLK    #170CH,T4CON
        SPLK    #075H,T4PR
        SPLK    #0400H,GPTCONB
        SPLK    #0FFFFH,EVBIFRB
        SPLK    #0000H,EVBIMRB
        LDP     #DP_PF2
        SPLK    #0010H,ADCCTRL1 ;设置 ADC 控制寄存器
        SPLK    #8404H,ADCCTRL2
        SPLK    #000FH,MAXCONV ;16 通道
        SPLK    #3210H,CHSELSEQ1
        SPLK    #7654H,CHSELSEQ2
        SPLK    #0BA98H,CHSELSEQ3
        SPLK    #0FEDCH,CHSELSEQ4
        LDP     #DP_SARAM2      ;指向 0A00h～0A80h
        SPLK    #ADRESULT,ADCOUNT
        RET
;（4）启动模数转换程序
AD:
        LDP     #234
        LACL    T4CON
        OR      #40H            ;启动定时器 4
        SACL    T4CON
        RET
;（5）中断程序
GISR1:                          ;优先级 INT1 中断子程序入口
        ;保护现场
        LDP     #0              ; 保存机器上下文
        SST     #0, st0_temp    ; 使用自动寻址 DP-0
        SST     #1, st1_temp    ; 保存状态寄存器到 B2 DARAM.
        SACL    context         ; 保存 ACC 的低 16 位
        SACH    context+1       ; 保存 ACC 的高 16 位
        SAR     AR1,context+2
        SAR     AR2,context+3
        SAR     AR3,context+4
        SAR     AR4,context+5
        SAR     AR5,context+6
        LDP     #0E0H
        LACC    PIVR,1          ;读取外设中断向量寄存器（PIVR）,并左移一位
        ADD     #PVECTORS       ;加上外设中断入口地址
        BACC                    ;跳到相应的中断服务子程序
ADCINT_ISR:
        CLRC    SXM
        LDP     #DP_SARAM2
```

```
                LAR     AR1,ADCOUNT
                LAR     AR0,#15
                LAR     AR2,#RESULT0
                MAR     *,AR2
ADC1:           LACC    *+,10,AR1
                SACH    *
                ADRK    #32
                MAR     *,AR0
                BANZ    ADC1,*-,AR2
                LDP     #DP_PF2
                LACL    ADCCTRL2
                OR      #4000H          ;复位  SEQ1
                AND     #0FFDFH         ;清  INT FLAG SEQ1
                SACL    ADCCTRL2
                LDP     #DP_SARAM2
                LACL    ADCOUNT
                SUB     #ADRESULT+31
                BCND    ADC2,GEQ
                ADD     #ADRESULT+32 ;<32
                SACL    ADCOUNT
                B       GISR1_RET
ADC2:           SPLK    #ADRESULT,ADCOUNT   ;=32
                LACL    T4CON
                AND     #0FFBFH         ;停止定时器 4,即停止 AD 转换
                SACL    T4CON
                B       GISR1_RET
GISR1_RET:                              ;中断返回出口
                LDP     #DP_EVA
                SPLK    #0FFFFH,EVAIFRA
                LDP     #0
                LAR     AR5,context+6
                LAR     AR4,context+5
                LAR     AR3,context+4
                LAR     AR2,context+3
                LAR     AR1,context+2
                LACC    context+1,16
                ADDS    context
                LST     #1, st1_temp
                LST     #0, st0_temp
;恢复现场
                CLRC    INTM            ;开总中断,因为一进中断就自动关闭了总中断
                RET
;（6）假中断程序
```

```
PHANTOM
        KICK_DOG                ;复位看门狗
        RET
        END
```

实验七　　串行外围接口 SPI 的应用

一．实验说明

SPI 是由美国摩托罗拉公司最先推出的一种同步串行传输规范，DSP 外设芯片具有串行扩展接口 SPI。SPI 接口主要用来和带串行接口的外围器件进行通讯的一种串行标准接口。由于具备 SPI 接口的外围器件，具有引脚少、封装简便、造价低廉等突出优点，在各种应用上得到了迅速而广泛的普及。

SPI 接口可以同时发送和接收 8 位数据，它共使用了 4 条引脚：

•SPI 从器件输入 / 主器件输出引脚（简称 SPISIMO）：作用是在一个方向上传送数据：先送高位（MSB），后送低位（LSB）。

•SPI 从器件输出 / 主器件输入引脚（简称 SPISOMI）：作用也是在一个方向上传送数据：先送高位，后送低位。如果从器件没有被选中，则主器件的 SOMI 线处于高阻状态。

•SPI 串行时钟引脚（简称 SPICLK）： 用于主、从器件之间在 MISO 和 MOSI 线上传送数据时进行同步。在主器件中作为输出线，在从器件中作为输入线。在 8 个时钟周期之内，主、从器件之间完成一个字节信息的交换。SPISCK 定时信号由主器件负责产生和输出。

•SPI 从器件发送使能引脚（简称 SPISTE）：对于工作于从器件模式的芯片，SPISTE 输入线用作选通信号输入端，该引脚必须在传送数据之前被设置为低电平，并且在整个数据传送过程中维持为稳定的低电平；对于工作于主器件模式的芯片，SPISTE 输入线必须接高电平。

在 DSP 片内现有的硬件资源如果不能满足应用中产品需求时，可以利用同步串行接口来扩展各种通用外设芯片。本实验选用带 SPI 接口的 DAC 芯片 MAX5121，用于目标系统的扩展，其应用原理：由 TMS320LF2407 送出的数字数据经过串行输出接口 SPI 将数值信号锁存在 DAC 控制芯片上，再经过 DAC 做数字到模拟信号的转换，最后输出设计所要求的正弦波、三角波等各种波形信号。

二．实验目的

1．展示 SPI 接口同时完成向对方发送一个字节数据和从对方接收一个字节数据两项任务的功能。

2．提供一种与带串行接口的外围器件接口方法，为其它方便、有效的 DSP 应用系统扩展方法打下基础。

三．实验内容

1． 使用 MAX5121 芯片功能产生模拟信号并设计调试相应的软件。

2． 将数字信号经 DAC 芯片转换的正弦波、三角波等波形输出用示波器观察、分析。

四．实验硬件电路

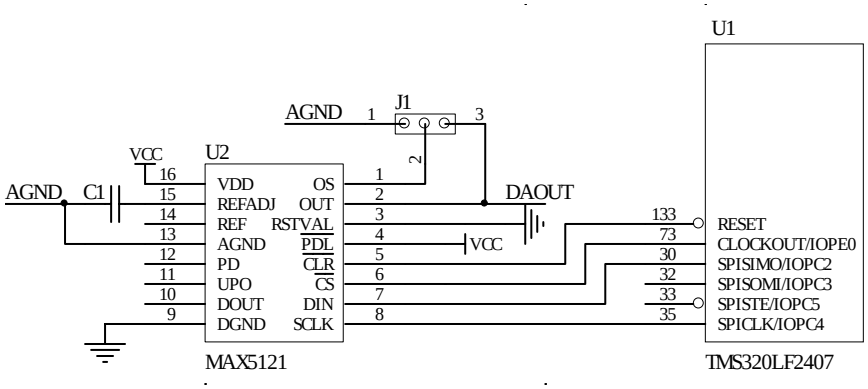


图 7.1　TMS320LF2407 与 MAX5121 接口电路

MAX5121 的 SPI 接口指令有 9 条，采用 16 位操作码，其操作码由 3 位控制位、12 位数据及 1 位子位组成。其指令集如表 7.1 所示。

表 7.1 MAX5121 的 SPI 接口指令集

16 位串行字					功能说明
C2	C1	C0	D11……D0	S0	
0	0	0	XXXXXXXXXXXX	0	空操作
0	0	1	12 位 DAC 数据	0	移入输入寄存器，DAC 寄存器不变并关闭 DAC
0	1	0	12 位 DAC 数据	0	同时移入输入和 DAC 寄存器并关闭 DAC
0	1	1	XXXXXXXXXXXX	0	把输入寄存器的值移入到 DAC 寄存器
1	0	1	XXXXXXXXXXXX	0	关闭 DAC 条件是 PDL＝1）
1	0	0	XXXXXXXXXXXX	0	UPO 引脚上输出低电平
1	1	0	XXXXXXXXXXXX	0	UPO 引脚上输出高电平
1	1	1	1XXXXXXXXXXX	0	方式 1：DOUT 在 SCLK 的上升沿发送数据
1	1	1	00XXXXXXXXXX	0	方式 0：DOUT 在 SCLK 的下降沿发送数据（缺省）

五．实验参考程序清单

C 语言程序

```
#include "register.h"
int GPR3;
int flag1;
int flag;

int initial()
{
    asm("    setc INTM");
    WSGR=0x00;
    asm("    clrc SXM");
    asm("    clrc OVM");
    asm("    clrc CNF");
    *SCSR1=0x81FE;
    *WDCR=0x0E8;
```

```
}

int SPIINITIAL()
{
        *SPICCR=0x004F;
        *SPICTL=0x0006;
        *SPIBRR=0x0007;
        *MCRB=0x003C;
        *MCRC=*MCRC&0x0FFFE;
        *SPICCR=*SPICCR|0x0080;
}

int SPITRANS()
{
        *PEDATDIR=(*PEDATDIR|0x0100)&0x0FFFE;
        *SPITXBUF=GPR3;
        while(1)
        {
        flag=*SPISTS&0x40;
        if(flag==0x40)
        break;
        }
        *SPIRXBUF=*SPIRXBUF;
        *PEDATDIR=*PEDATDIR|0x01;
}

main()
{
        initial();
        SPIINITIAL();
        GPR3=0x4000;
        flag1=0x00;
        while(1)
        {
        if(flag1==0x00)
        GPR3=GPR3+2;
        else
        GPR3=GPR3-2;
        if(GPR3==0x5FFE)
        flag1=0x01;
        if(GPR3==0x4000)
        flag1=0x00;
        SPITRANS();
        }
```

```
}

void interrupt nothing()
{
      return;
}
```

汇编程序

```
;用户寄存器定义
SPI_DATA  .usect     ".data0",1          ;临时数据寄存器
SPI_FLAG  .usect     ".data0",1          ;SPI  标志位
SPI_CON   .usect     ".data0",1          ;MAX5121 的控制字
DP_USER .set    5                        ;用户寄存器存放页指针
;MAX5121  的控制字宏定义
DACOUT  .set         4000h               ;C2C1C0=010
UPINREG .set         2000h               ;C2C1C0=001
UPDACREG     .set         6000h               ;C2C1C0=011
SHUTDAC      .set         0A00h               ;C2C1C0=101
;（1）  主程序
                .include   "F2407REGS.H"
                .def       _c_int0
          .text
_c_int0
                CALL   SYSINIT              ;调系统初始化程序
                CALL   SPI_INIT          ;调 SPI 初始化程序
LOOP:           CALL   SPI_SEND          ;调输出三角波程序
                NOP
WAIT:           LDP       #DP_USER
                SPLK      #00H, SPI_FLAG     ;清标志
                SPLK      #00H, SPI_DATA     ;重置初值
                B         LOOP
;（2）  系统初始化程序
SYSINIT:
                SETC      INTM
          CLRC      SXM
                CLRC   OVM
                CLRC   CNF
          LDP       #0E0H
          SPLK      #81FEH,SCSR1  ;4 倍频 CLKIN=6 M,CLKOUT=24 M
          SPLK      #0E8h,WDCR    ;关看门狗
          LDP       #0
                SPLK      #0001h,IMR      ;使能中断 1
                SPLK      #0FFFFh,IFR     ;清中断标志
                RET
```

```
;（3） SPI 初始化程序
SPI_INIT:
        LDP     #DP_PF2
        LACL    MCRB
        OR      #00014H         ;配置 SPISIMO 和 SPICLK 引脚为特殊功能方式
        SACL    MCRB
        LACL    MCRC
        AND     #0FFFEH         ;配置 IOPE0 为一般的 I/O 口功能
        SACL    MCRC            ;CS=IOPE0
        LDP     #DP_PF1
        SPLK    #004FH,SPICCR   ;配置 SPI 寄存器允许初始化,16 位数据输出
        SPLK    #0006H,SPICTL   ;主机方式,时钟方式为无延时的下降沿
        SPLK    #0002H,SPIBRR   ;SPI 波特率为 6 MHz
        SPLK    #00CFh,SPICCR   ;初始化结束,并关闭初始化使能位
        LDP     #DP_USER
        SPLK    #00H,SPI_DATA   ;置发送数据初值
        SPLK    #00H,SPI_FLAG   ;SPI_FLAG=0000H,执行三角波上
                                ;SPI_FLAG=0001H,执行三角波下降
        SPLK    #DACOUT,SPI_CON;送 MAX5121 的控制字
        RET
;（4） 输出三角波程序
SPI_SEND:
SPI_TX:     LDP     #DP_PF2
            LACL    PEDATDIR
            OR      #0100H          ;IOPE0 脚为输出方式
            AND #0FFFEH         ;CS= IOPE0=0
            SACL    PEDATDIR
            NOP
            NOP
            LDP     #DP_USER
            LACC    SPI_DATA
            AND #01FFEH         ;S0=0
            OR      SPI_CON
            SACL    SPI_DATA    ;规格化发送的数据
            LDP     #DP_PF1
            SACL    SPITXBUF        ;数据写入到 SPI 发送缓冲区
XMIT_RDY:
            BIT     SPISTS,BIT6     ;等待数据
            BCND    XMIT_RDY,NTC  ;发送完
            NOP
            NOP
            NOP
            LDP     #DP_PF2
            LACL    PEDATDIR
```

```
        OR      #0101H          ;CS=IOPE0=1
        SACL    PEDATDIR        ;锁存数据
        LDP     #DP_USER
        BIT     SPI_FLAG,BIT0
        BCND    SPI_FALL,TC
        LDP     #DP_USER        ;三角波上升段程序
        LACC    SPI_DATA
        AND #01FFEH
        ADD #02H            ;递增
        SACL    SPI_DATA
        SUB     #01FFEH
        BCND    SPI_FALL,EQ
        B       SPI_TX
SPI_FALL:
        LDP   #DP_USER          ;三角波下降段程序
        SPLK    #01, SPI_FLAG
        LACC    SPI_DATA
        AND #01FFEH
        SUB     #02H            ;递减
        SACL    SPI_DATA
        BCND    SPI_RET,EQ
        B       SPI_TX
SPI_RET:  RET
```

实验八　串行通信接口模块（SCI）

一．实验说明

计算机与外界所进行的信息交换经常被人们称为数据通信（简称通信）。通信的基本方式又可分为并行通信和串行通信两种。串行通信的实现，在制式、种类、形式、规范、标准、编码、检错、纠错、帧结构、组网方式、调制方式、主要用途等许多方面，存在着多种类型、变化、选择和解决方案，其中之一有美国电子工业协会推荐标准 RS-232，它是用来实现与串行通信功能相关的技术和规范。

RS-232 接口采用 25 针的链接器 DB-25 或 9 针链接器 DB-9，其每一条插针的信号功能都是标准的，对于各种信号的电平规定也是标准的，因而便于各种数字设备之间的兼容和互相链接。其基本的信号定义如表 8.1 所列。

表 8.1 RS-232 接口的信号定义

DB-25 脚位	DB-9 脚位	信号名称	方向	含义
2	3	TXD	输出	数据发送端
3	2	RXD	输入	数据接收端
	7	RTS	输出	请求发送（计算机要求发送数据）
4	8	CTS	输入	清除发送（MODEM 准备接收数据）
5	6	DSR	输入	数据设备准备就绪
6	5	SG	—	信号地
7	1	DCD	输入	数据载波检测
8	4	DTR	输出	数据终端准备就绪（计算机）

20 22 1	9 —	RI —	输入 —	响铃指示 保护地

大多数的电脑设备都具有 RS-232 接口，尽管它的性能指标并非很好，但其在广泛的市场支持下依然常胜不衰。就使用而言，RS-232 也确实有其优势：仅需 3 根线（TXD、RXD 和 SG）便可在两个数字设备之间全双工的传送数据。

TMS320LF2407 内部集成 SCI 模块。SCI 模块采用的是一种在标准规范基础上简化了的、无握手信号的、二线式的串行通信方式，使得占用芯片引脚资源的数量降低到最低限度。

TMS320LF2407 的 SCI 具有异步和同步通信能力，其异步通信能力主要用于与其它计算机系统或 DSP 系统进行远程通信，而同步通信能力则主要用于本 DSP 电路系统之内的片外器件串行扩展。

SCI 可以定义三种工作方式：全双工异步方式、半双工同步主控方式和半双工同步从动方式。在 DSP 应用项目的开发过程中，如何利用更少的 DSP 引脚实现更多信息吞吐？如何利用更简练的电路和更廉价的器件实现更丰富功能？始终是我们孜孜以求的努力目标。为此，本实验利用 PC 机的超级终端程序，经过异步串行通讯端口 COM 与 TMS320LF2407 芯片的串行通信模块接口，进行双向异步通信的实验。本例的应用方案将会对同学们有一定的启发和实用价值。

二．实验目的

1. 了解 DSP 与 PC 机之间串行异步通信接口的应用。
2. 了解串行通信的设置，以便正确实现与 PC 机（上位机）的通信。
3. 通过实验了解串行数据的接收及传送。

三．实验内容

1. 理解硬件电路和 RS－232 标准。
2. 令 SCI 模块工作于异步接收 / 发送两种状态。
3. 将通信波特率设定在 9600 波特、8 位数据、1 位停止位、不设奇偶校验位。
4. 将 PC 机发送的数据返回到 PC 机。

四．实验硬件电路

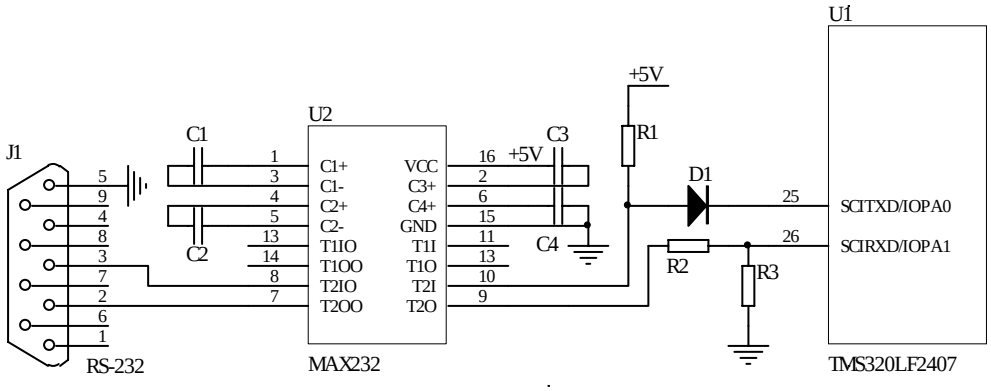


图 8.1　TMS320LF2407 与 MAX232 接口电路

五．实验参考程序清单

C 语言程序

```
#include "register.h"
int x,y,k=0;
static int receive[4],j=0;
//const int var[4]={0x12,0x23,0x34,0x45};
int var[4]={0x12,0x23,0x34,0x45};
void inline disable()
```

```
{
        asm("       setc INTM");
        asm("       setc SXM");
}

void inline enable()
{
        asm("       clrc INTM");
}

main()
{
        disable();
        *IFR=0xFFFF;
        *SCSR1=0x81FE;
        *WDCR=0xE8;
        *SCICCR=0x7;
        *SCICTL1=0x13;
        *SCICTL2=0x3;
        *SCIHBAUD=0x1 ;
        *SCILBAUD=0x38;
        *SCICTL1=0x33;
        *SCIPRI=0x60;
        *MCRA=0x3;
        *PADATDIR=0x100;
        *IMR=0x10;
        enable();
        *SCITXBUF=0x55;
        while(1)
        ;
}

void UartSent()
{
        static int i=0,m;
        if(i>3)
        return;
        *SCITXBUF=var[i++];
        *IFR=0x0010;
        enable();
}

void UartRec()
{
```

```
        receive[j++]=*SCIRXBUF;
        if(j>9)
        j=0;
        *IFR=0x0010;
        enable();


}

void interrupt uarttr()
{
        switch(*PVIR)
        {
        case 6:UartRec();
        case 7:UartSent();
        }
}

void interrupt nothing()
{
        return;
}
```

汇编程序

```
SCI_FLAG    .usect    ".data0",1          ;SCI 标志寄存器
TXD_PTR     .usect    ".data0",8          ;发送的数据存放区
RXD_PTR     .usect    ".data0",8          ;接收到的数据存放区
N           .set      8

            .include  "F2407REGS.H"       ;引用头部文件
            .def      _c_int0
;（1）建立中断向量表
            .sect     ".vectors"          ;定义主向量段
RSVECT      B         _c_int0             ;PM 0    复位向量          1
INT1        B         GISR1               ;PM 2    中断优先级 1      4
INT2        B         PHANTOM             ;PM 4    中断优先级 2      5
INT3        B         PHANTOM             ;PM 6    中断优先级 3      6
INT4        B         PHANTOM             ;PM 8    中断优先级 4      7
INT5        B         PHANTOM             ;PM A    中断优先级 5      8
INT6        B         PHANTOM             ;PM C    中断优先级 6      9
RESERVED    B         PHANTOM             ;PM E     (保留位)         10
SW_INT8     B         PHANTOM             ;PM 10   用户定义软件中断        —
SW_INT9     B         PHANTOM             ; PM 12  User S/W int     -
SW_INT10    B         PHANTOM             ; PM 14  User S/W int     -
```

```
SW_INT11    B       PHANTOM         ; PM 16    User S/W int     -
SW_INT12    B       PHANTOM         ; PM 18    User S/W int     -
SW_INT13    B       PHANTOM         ; PM 1A    User S/W int     -
SW_INT14    B       PHANTOM         ; PM 1C    User S/W int     -
SW_INT15    B       PHANTOM         ; PM 1E    User S/W int     -
SW_INT16    B       PHANTOM         ; PM 20    User S/W int     -
TRAP        B       PHANTOM         ; PM 22    Trap vector-
NMI         B       PHANTOM         ; PM 24    Non maskable Int3
EMU_TRAP    B       PHANTOM         ; PM 26    Emulator Trap    2
SW_INT20    B       PHANTOM         ; PM 28    User S/W int     -
SW_INT21    B       PHANTOM         ; PM 2A    User S/W int     -
SW_INT22    B       PHANTOM         ; PM 2C    User S/W int     -
SW_INT23    B       PHANTOM         ; PM 2E    User S/W int     -
SW_INT24    B       PHANTOM         ; PM 30    User S/W int     -
SW_INT25    B       PHANTOM         ; PM 32    User S/W int     -
SW_INT26    B       PHANTOM         ; PM 34    User S/W int     -
SW_INT27    B       PHANTOM         ; PM 36    User S/W int     -
SW_INT28    B       PHANTOM         ; PM 38    User S/W int     -
SW_INT29    B       PHANTOM         ; PM 3A    User S/W int     -
SW_INT30    B       PHANTOM         ; PM 3C    User S/W int     -
SW_INT31    B       PHANTOM         ;PM 3E     用户定义软件中断    —
;中断子向量入口定义 pvecs
            .sect   ".pvecs"        ;定义子向量段
PVECTORS    B       PHANTOM         ;保留向量地址偏移量 0000h
            B       PHANTOM         ;保留向量地址偏移量 0001h
            B       PHANTOM         ;保留向量地址偏移量 0002h
            B       PHANTOM         ;保留向量地址偏移量 0003h
            B       PHANTOM         ;保留向量地址偏移量 0004h ADC 中断
            B       PHANTOM         ;保留向量地址偏移量 0005h
            B       SCI_RX_ISR      ;保留向量地址偏移量 0006h SCI 接收中断
            B       PHANTOM         ;保留向量地址偏移量 0007h
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量-0a
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量-10
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
            B       PHANTOM         ; 保留向量地址偏移量
```

```
B       PHANTOM         ; 保留向量地址偏移量-15
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-1a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-20
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-25
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-0027 T1PINT
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-2a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 向量地址偏移量 0x02f - T3PINT
B       PHANTOM         ; 保留向量地址偏移量-30
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-35
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-3a
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量
B       PHANTOM         ; 保留向量地址偏移量-3f
B       PHANTOM         ;CANMBX_ISR; 保留向量地址偏移量
B       PHANTOM         ;保留向量地址偏移量 0041h
```

```
;（2）主程序:
        .text
_c_int0
        SETC    INTM
        CLRC    SXM
        CLRC    OVM
        CLRC    CNF
        LDP     #0E0H
        SPLK    #81FEH,SCSR1        ;CLKIN=6 M,CLKOUT=24 M
        SPLK    #0E8h,WDCR          ;关看门狗
        CALL    SCI_INIT            ;调串口初始化程序
        LDP     #5H
        SPLK    #00H,SCI_FLAG       ;清发送软件标志位
TXD_LOOP:
        LDP     #5H
        BIT     SCI_FLAG,BIT0

        BCND    TXD_DATA,TC         ; SCI_FLAG.0=1 则发送数据
        B       WAIT
TXD_DATA:                           ;发送 N 个数据程序
        MAR     *,AR0
        LAR     AR0,#TXD_PTR
        ADRK    #N+1                ;发送 N 个数据
        MAR     *,AR4
        LACC    *+                  ;读数据
        CMPR    00
        BCND    TXD_DATA_END,TC;判数据发送完？
        MAR     *,AR2
        SACL    *,AR4               ;数据送 SCITXBUF 寄存器
XMIT_RDY:
        LDP     #DP_PF1
        BIT     SCICTL2,BIT7
        BCND    XMIT_RDY,NTC        ;判发送器是否空？
        B       TXD_DATA
TXD_DATA_END:
        MAR     *, AR4
        LAR     AR4,#TXD_PTR        ;恢复发送的数据指针
        LDP     #5H
        SPLK    #00H,SCI_FLAG       ;清发送软件标志位
WAIT:   NOP
        B       TXD_LOOP
;（3）串行通信初始化程序:
SCI_INIT:
        LDP     #0E1H
```

```
                LACL    MCRA
                OR      #03H
                SACL    MCRA            ;配置串行口引脚为特殊功能:TXD、RXD
                LDP     #DP_PF1
                SPLK    #000FH,SCICCR   ;地址位唤醒模式,8 位数据,
;1 位停止位,无奇偶校验
                SPLK    #0007H,SCICTL1  ;接收、发送、内部时钟使能 ,SLEEP=1
                SPLK    #0002H,SCICTL2  ;接收中断使能
                SPLK    #0001H,SCIHBAUD
                SPLK    #0038H,SCILBAUD ;波特率为 9600
                SPLK    #0027H,SCICTL1  ;串口初始化完成
                SPLK    #00H,SCIPRI
                LAR     AR1,#SCIRXBUF   ;接收缓冲寄存器地址
                LAR     AR2,#SCITXBUF   ;发送缓冲寄存器地址
                LAR     AR3,#RXD_PTR    ;接收的数据指针
                LAR     AR4,#TXD_PTR    ;发送的数据指针

                LDP     #0
                SPLK    #0001h,IMR      ;允许 INT1 中断
                SPLK    #0FFFFh,IFR     ;清所有中断标志
                CLRC    INTM            ;开总中断
                RET
;（4）中断程序
GISR1:                                  ;优先级 INT1 中断人口
          ; 保护现场
                LDP     #0              ; 保存机器上下文
                SST     #0, st0_temp    ; 使用自动寻址 DP-0
                SST     #1, st1_temp    ; 保存状态寄存器到 B2 DARAM.
                SACL    context         ; 保存 ACC 的低 16 位
                SACH    context+1       ; 保存 ACC 的高 16 位
                SAR     AR1,context+2
                SAR     AR2,context+3
                SAR     AR3,context+4
                SAR     AR4,context+5
                SAR     AR5,context+6

                LDP     #0E0H
                LACC    PIVR,1          ;读取外设中断向量寄存器（PIVR）,并左移一位
                ADD     #PVECTORS       ;加上外设中断人口地址
                BACC                    ;跳到相应的中断服务子程序
SCI_RX_ISR:                             ;接收中断服务程序
                LDP     #DP_PF1
                MAR     *,AR1
                BIT     SCICTL1,BIT2
```

```
                BCND    RXD_DATA,NTC    ;SLEEP=0,接收数据
                LACC    *               ;读入地址
                SUB     #0AAH
                BCND    TXD_FLAG,EQ     ;接收到的地址为 AAH,则发送数据
                SUB     #0001H
                BCND    SCI_RX_END,NEQ  ;地址错误,退出中断
                SPLK    #0033H,SCICTL1  ;地址为 ABH,则清 SLEEP=0
;表示准备接收数据
                B       SCI_RX_END
TXD_FLAG:                               ;置发送标志
                LDP     #5H
                SPLK    #01H,SCI_FLAG
                B       SCI_RX_END
RXD_DATA:                               ;接收 N 个数据
                LACC    *,AR3           ;读入数据
                SACL    *+,AR0          ;保存数据
                LAR     AR0,#RXD_PTR
                ADRK    #N
                MAR     *,AR3
                CMPR    00
                BCND    SCI_RX_END,NTC  ;判是否接收完
                LAR     AR3,#RXD_PTR    ;恢复接收的数据指针 #RX_PTR
                SPLK    #0037H,SCICTL1  ;置 SLEEP=1 表示准备接收地址
SCI_RX_END:                     ;       中断返回
        ;恢复现场
                LDP     #DP_EVA
                SPLK    #0FFFFH,EVAIFRA
                LDP     #0
                LAR     AR5,context+6
                LAR     AR4,context+5
                LAR     AR3,context+4
                LAR     AR2,context+3
                LAR     AR1,context+2
                LACC    context+1,16
                ADDS    context
                LST     #1, st1_temp
                LST     #0, st0_temp
                CLRC    INTM            ;开总中断,因为一进中断就
;自动关闭总中断
                RET
;（5）假中断程序
PHANTOM
                KICK_DOG                ;复位看门狗
                RET
```

END

附录二　TMS320LF2407 DSP 实验开发系统

实验是 DSP 应用技术学习的一个重要环节。通过实验，可以对 DSP 器件的功能进行实际操作，并在实际操作中熟悉有关器件的使用特性。这些都是 DSP 应用技术的基本内容，也是学习 DSP 技术的基本内容之一。

根据 DSP 器件和应用系统开发技术学习的需要，TMS320LF2407 DSP 实验开发系统提供了充足的实验资源，可以完成基本的 DSP 技术实验，可以根据所提供的目标系统，设计开发小型的 DSP 应用系统。

一．系统简介

TMS320LF2407 DSP 实验开发系统可以用于初学者的学习平台，或用于大学本科生的教学使用，它由具有 JTAG 接口的 TMS320LF240X 系列 XDS510 硬件仿真器、TMS320C2000 系列的调试环境代码编译器 CC 和本实验自制的 2407 目标系统实验箱三部分组成（详细介绍参见相关内容）。软件所配置的 CC 代码编译器 CC4.10 版本可以开发 C2000 系列的 TI DSP 芯片，标准配置支持 TMS320C2XX 系列。该实验开发系统采用 TMS320LF2407 DSP 数字信号控制器作为开发系统的核心，并提供了如下的应用资源：

•JTAG 接口：开发系统利用该接口将仿真器与目标系统链接，实现 PC 机对 TMS320LF2407 实验开发系统的控制，数据交换。

•人机交换接口：实验板上共有 5 个按键、8 只 LED 发光二极管和液晶显示（LCD）器。5 个按键，一个作为复位键，其余 4 个为功能键；8 只 LED 发光二极管由 IOPB 口，经 74HC237 锁存器控制，可以显示数字线上某一位的高低电位状态，是一种最简单的输出装置。当程序能正确执行时可以令其闪动一下，若是不能如预期工作，例如程序执行死循环，则看不到 LED 闪烁了，因此，使用 LED 可以指示系统当前的工作状态，此外 LED 也可当作电源指示灯用，当电路短路时，可以很容易的察觉。LCD 可显示 16×16 点阵的汉字、图像及其它字符，可显示较大的信息量，因此在 DSP 控制程序的开发上可用来显示程序执行时的中间过程，在除出错误时很有帮助，另外可以作产品线上功能操作提示说明用。

•串行通信接口（SCI）：这是 DSP 控制系统中使用到的装置，该接口可以通过 RS232 或 RS485 转换芯片与 PC 机进行异步通讯，作为 PC 机连线控制用或是数据传输用。在程序的开发过程中，将调试主机与目标系统经 SCI 联机的方式，可以提供一个全方位的系统开发调试环境。若通过 SCI 将执行结果传回 PC，再由 PC 的串行传输接收程序负责接收消息将其显示在屏幕上，那么一项复杂的程序开发工作可能会变得更系统化，由于屏幕可以显示很多信息，这在程序开发上是最有效率的。任何在 DSP 上执行的数值运算结果，均可传回 PC 而显示出来，可方便验证程序执行的正确性。实验板上的 SCI 引脚分别与 RS232 和 RS485 电平转换芯片相连，用户可通过跳针 JP6 和 JP7 来选择 RS232 或 RS485 通讯。

•控制器区域网络（CAN）：主要用于各种设备监测及控制的一种网络。CAN 具有独特的设计思想，良好的功能特性和极高的可靠性，现场抗干扰能力强。CAN 总线采用 CRC 检验并可提供相应的错误处理功能，保证了数据通信的可靠性。TMS320LF 自带 CAN，通过设置内部寄存器的自测试位，来实现 CAN 控制器的自发自收功能，为调试 CAN 通讯的下位机提供方便。通过 CAN 驱动器 PCA82C250T 就可以与其它节点或上位 PC 机进行通讯。

•串行外设接口（SPI）：实验板上用带电压参考的 SPI 总线 D／A 芯片 MAX5121 与该接口相连。D／A 芯片给出来的电压参考经变换后作为 TMS320LF2407 DSP 处理器的 A／D 转换的电压参考输入。

•用 I^2C 总线实现的 EEPROM 和日历时钟电路：由于 TMS320LF2407 芯片没有 I^2C 总线接口，所以只能用通用的 I／O 引脚来与 I^2C 总线的数据线 SDA 和时钟线 SCL 相连，用软

件来模拟 I^2C 时序。实验板上的 EEPROM 芯片为带写保护的 24LC256，日历时钟芯片为 PCF8583，由于共用时钟线 SCL，所以不能同时访问。

针对 TMS320LF2407 芯片，TMS320LF2407 DSP 实验开发系统提供了专用调试环境，具有用户程序载入、调试功能。在使用 TMS320LF2407 DSP 实验开发系统时，可以随时查看或修改：

•TMS320LF2407 芯片中各寄存器的内容；

•DMA 控制寄存器的内容；

•片内存储器的内容。

TMS320LF2407 DSP 实验开发系统提供了用图像窗口显示设置断点、单步运行和跟踪、全速运行、程序的分支和外部中断的计数等功能。

二．系统功能电路

TMS320LF2407 DSP 实验开发系统中，以 TMS320LF2407 DSP 控制器为核心，提供了 32K 字的片内 FLASH 程序存储器，1.5K 字的数据／程序 RAM，544 字的双口 RAM（DARAM）和 2K 字的单口 RAM（SARAM）；两个事件管理器模块 EVA 和 EVB；可扩展的外部存储器总共 192K 字空间；看门狗定时器模块（WDT）；10 位 A／D 转换器；控制器局域网络（CAN）模块；串行通信接口（SCI）模块；16 位的串行外设（SPI）接口模块等电路。通过使用调试软件，可以对这些电路实现有效控制，完成所需要的使用或开发工作。

三．系统硬件的联接

由于 XDS 仿真器中没有 DSP 器件，而是提供 IEEE 标准的 JTAG 接口对 DSP 进行仿真调试，所以仿真器必须有仿真对象，及目标系统。目标系统也就是实验室提供的实验板（上面带有 DSP 器件）。仿真器提供 JTAG 接口同目标系统的 DSP 相接，通过 DSP 实现对整个目标系统的调试。在完成仿真器的安装准备之后，进行联机调试时，需要将调试主机 PC、XDP 硬件仿真器及待调试的目标系统链接起来。PC 与仿真器的链接是通过打印机接口的 D25 插座来实现的。链接时需要一根标准的打印机链接电缆。仿真器与目标系统的链接是通过 TMS320LF2407 DSP 芯片上的 JTAG 接口。仿真器的 JIAG 接口是一个串行通信接口，PC 机的打印机接口实际上也是当做一个串行的通信接口来用的。

附录五　程序开发平台 Code Composer

由于 DSP 器件越来越复杂，DSP 系统的开发技术也变得越来越接近大型的微处理器系统开发技术。为了提高开发工作效率、缩短应用系统开发周期，TI 公司开发研制出了针对 TMS320C2000／5000／6000 器件的开发平台－－Code Composer 即 CC。

CC 是基于 PC 机的 DSP 应用系统开发平台，它提供方便且功能丰富的窗口界面，使开发和调试的效率大大提高，并能与 TI 公司生产的目标系统实用工具链接使用，直接实现用户目标系统的开发。

CC 软件可以集成在 Windows9x/ 2K/ NT4.0 环境中运行。针对用户特定的目标系统，用创建项目（Prouects）的方法来进行开发和管理，它的基本特性如下：

•可以在 Windows9x/ 2K/ NT4.0 环境下运行

•分层次的项目管理

•Windows 风格的可视化用户界面

•必要时可用 Make 重新编译、汇编和链接文件

•CC 的所有工具都完全集成在一起，以方便使用

•支持直观的拖放功能

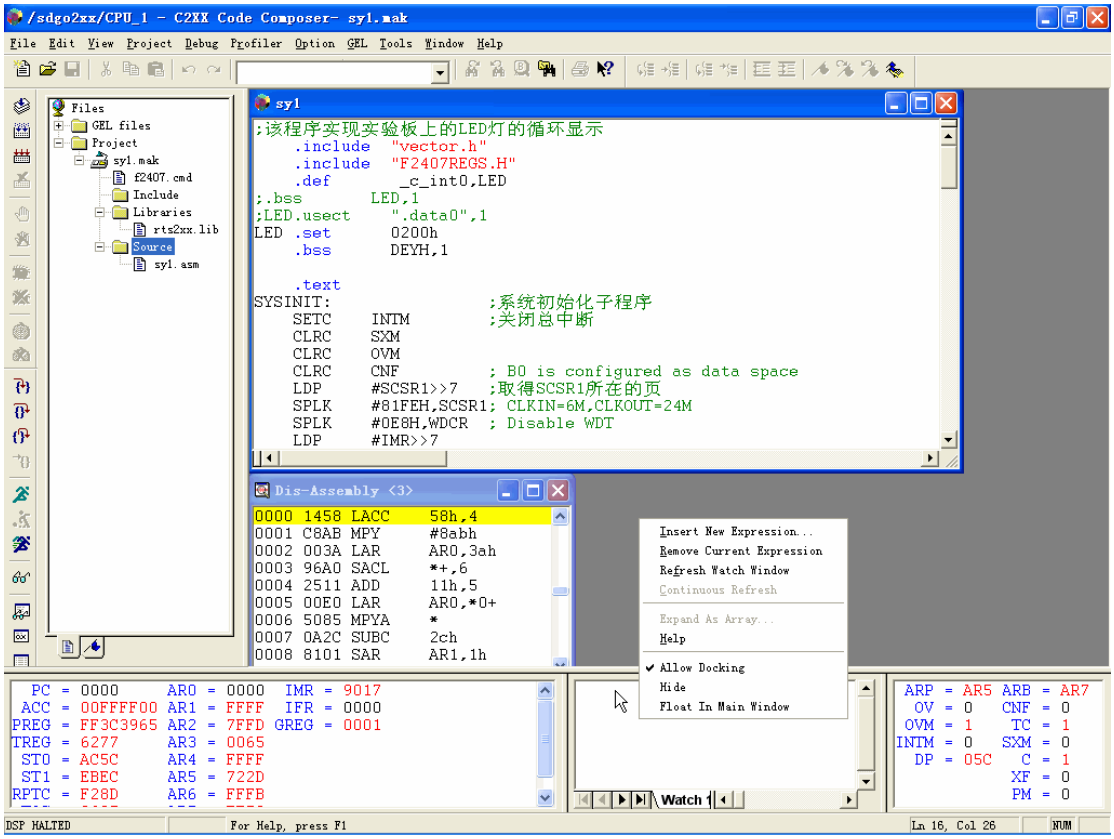
•有超文本风格的操作帮助

下面对程序开发平台环境的使用作简略介绍

当完成了安装之后，安装程序会在桌面上建立两个快捷图标“CC’C2000”和“Setup CC’C2000”。“Setup CC’C2000”用来进行仿真运行环境设置，“CC’C2000”为程序调试仿真环境。

二．CC’C2000 窗口和工具栏

CC’C2000 界面允许用户对除编辑窗口以外的其它所有窗口和工具栏进行随意设置。CC’C2000 的仿真操作窗口如图 5.9 所示。



附图 5.9

1．移动窗口

在 CC’C2000 中，可以将窗口或工具栏拖移到新的位置。还可以将窗口或工具栏移动以 CC’C2000 主窗口以外的地方。拖移的方法很简单，只要用鼠标箭头按住要拖移的窗口或工具栏即可进行拖移

2．语境菜单

CC’C2000 的所有窗口中都含有一个语境菜单“context menus”。只要在窗口中右击鼠标按键就可以打开弹出语境菜单。

在语境菜单中提供了一些选项和命令，用这些选项和命令可对窗口进行设置。例如，可以在项目窗口中右击鼠标所显示的项目文件，就可以弹出语境菜单，通过选择相应的设置操作命令就可以完成相应的设置操作（如增加“add”、移动源“remove source”、GEL 文件“GEL files”、设置选项“set build options”等）。

3．标准工具栏

在 CC’C2000 的标准工具栏中，有一些文档操作工具与 Windows 中的完全相同，如附图 5.10 所示，这里不再介绍。以下主要介绍几个特殊的文档操作命令。

附图 5.10　标准工具栏编辑栏

（1）查找字符串命令“ ”和“ ”

用来查找当前光标所在位置之后程序文档中的字符串，用来查找当前光标所在位置之前程序文档中的字符串。查找的方法如下：

•在工具栏查找域中输入所要查找的字符（附图 5.10 中的空白区）。注意：查找命令具有记忆功能，历次所查找的字符会保存在查找域中。

•用鼠标点击标准工具栏中的按钮，向后查找时按，向前查找时按。

如果把所查找到的字符串用另一个字符串替换，则可以采用菜单中的“查找 / 替换”命令。这个命令除了具有查找指定字符串功能之外，还可以在查找的同时完成字符串的替换。具体操作方法如下：

•在文档中把光标放在须要进行查找 / 替换操作的开始位置。

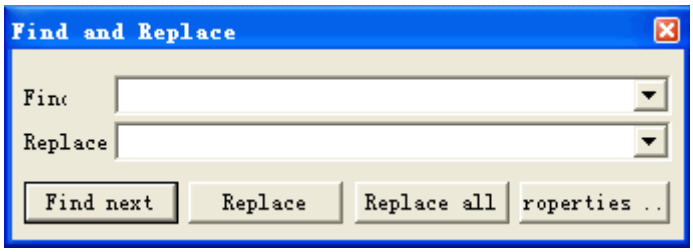
•选择菜单命令“Fdit ＞Find/Replac”（编辑＞查找 / 替换），这时会弹出一个如附图 5.11 所示的“Find and Replac”对话框。

•在对话框中输入：

Find：要查找的字符串；

Replace：替换字符串。

•点击“Find next”按钮后光标指向所查找到的字符串，这时如果确认须要替换，就点击“Replace”按钮，可以完成一次替换；如果不希望替换，则继续点击“Find next”按钮，指向下一个查找的字符串；如果点击“Replace all”按钮则会自动替换文档中的所有字符串。



附图 5.11　查找 / 替换对话框

（2）　查找指定文档中字符串的工具“ ”

这个工具可以用来查找所指定文档中的指定字符串，其使用方法与查找字符串的方法相同。

4.　“Dis-Assembly Window”（反汇编窗口）的使用

当用户把一个程序装载到目标系统 DSP 器件中后，CC’C2000 会自动打开一个如附图 5.12 所示的 “Dis-Assembly Window”（反汇编窗口）。

在反汇编窗口中将显示通过反汇编得到的指令和符号信息。这些指令和符号信息是调试

过程中所需要的。反汇编窗口中会显示出执行程序指令及其在程序存储器中的地址以及与这些指令有关的操作码（机器码）。

如果用户程序是使用 C 语言编写的，则可以选择 C 语言程序和汇编程序混合观察的方式。

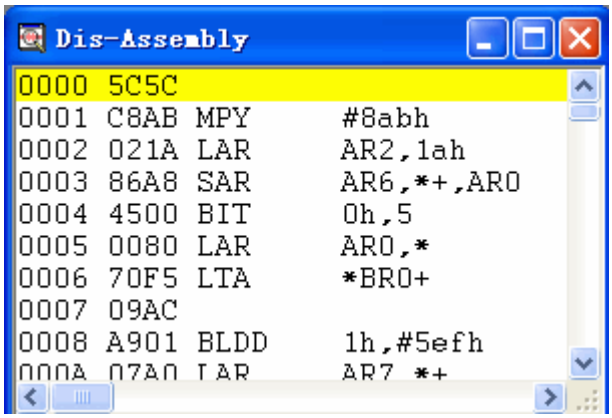
（1）打开多个反汇编窗口

通过选择菜单命令“View＞Dis-Assembly”（观察＞反汇编），或者选择调试工具栏中的  图标按钮，可以打开多个反汇编窗口。

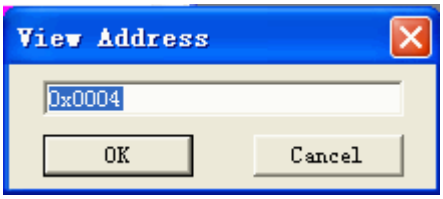
（2）修改程序起始地址

在实际程序调试时，有时需要修改执行程序的起始地址，这可以在反汇编窗口中完成。修改的方法如下：

•在反汇编窗口中双击程序的地址域，这时会弹出一个如附图 5.13 所示的“View Address”（反汇编起始地址修改）对话框。



附图 5.12　反汇编窗口



附图 5.13　反汇编起始地址修改对话框

•在对话框中输入所需要的程序地址。所输入的地址可以是一个绝对数，也可以是一个 C 语言表达式。

•点击“  ”。

（3）在反汇编窗口中管理断点、探测点、描述点

在 DSP 系统调试过程中，经常须要设置断点（breakpoints）、探测点（Probe Point 信号观察点）和描述点（profile point 判断程序结构等需要的观察点），通过这些特殊的设置可以加快系统的调试和仿真速度，发现系统中硬件和软件存在的问题。

在反汇编窗口中可以完成断点、探测点和描述点的设置或清除工作。具体的方法如下：

•利用调试和结构观察菜单，从调试“Debug”或结构观察器“Profiler”菜单中选择相应的命令就可完成设置。

•利用项目条（project）  中相应的按钮完成设置。

•双击要设置断点、探测点、描述点的行，双击后该行会出现背景色，表示设置完成。

（4）设置反汇编窗口风格的选项

CC’C2000 提供了几种不同的选项，通过使用这些选项可以在反汇编窗口中设置不同的信息观察方式。反汇编窗口选项对话框允许用户输入特殊的观察选项调试过程，例如可以选择十六进制或十进制观察地址数据。设置反汇编风格选项的方法如下：

•选择菜单命令“Option＞Dis-Assembly Style”（选项＞反汇编风格），或者在反汇编窗口中右击鼠标，弹出语境菜单，选择其中的“Properties-> Dis-Assembly Options…”，如附图 5.14 所示。

•进入所选择的反汇编窗口风格选项对话框，选择所需要的显示格式，如附图 5.15 所示。

•点击“OK”。

上述操作完成后，反汇编窗口会立即变为用户所设置的风格。

（5）观察 C 源程序与汇编程序的混合代码

在反汇编窗口中，CC’C2000 调试器允许对 C 程序插入汇编程序的反汇编结果。也就是说，可以观察到 C 程序与汇编程序通过反汇编统一在一起的指令结果。

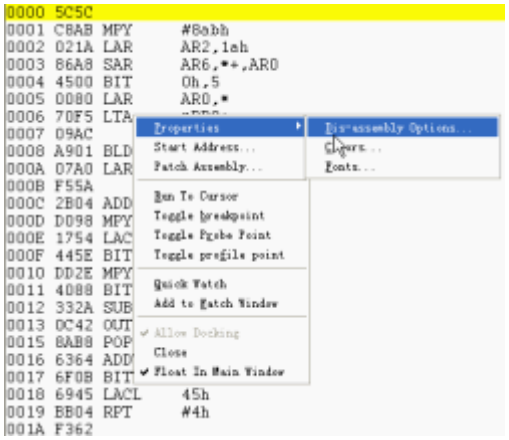
把程序装载到目标系统或仿真器中后，观察 C 程序与汇编程序混合反汇编的方法如下：

•选择菜单命令“View＞Mixed Source/ ASM”（视图＞）。

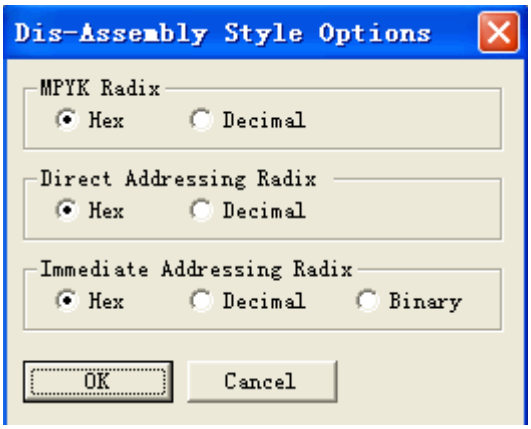
•选择菜单命令“Debug＞Go Main”（调试＞移动到 Main）。

这之后，调试器开始执行用户程序，并在“main()”处停止执行，同时，C 程序的源文件也会显示在编辑窗口中，并会用黄色背景显示出 PC（指令计数器）。

此外，用户还可以选择是否用汇编指令显示 C 源程序。方法是选择菜单命令“View”＞Mixed Source/ ASM”（视图＞），或在编辑窗口中右击鼠标后，在语境对话框中选择“Mixed Mode”或“Source Mode”。



附图 5.14　语境设置菜单



附图 5.15　语境选择对话框

5．使用存储器窗口

与单片机和微处理器系统开发过程相同，往往要直接观察存储器的内容。例如在应用系统中的显示器控制程序调试中，开发者往往要直接观察到显示字符的转换过程，以便确定程序的正确性。CC’C2000 开发环境中的存储器窗口就是为此目的而设置的。

（1）观察存储器内容操作

观察存储器内容的方法如下：

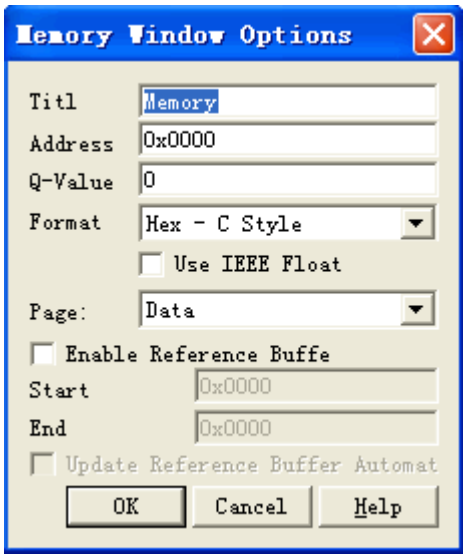
•选择菜单命令“View＞Memory”（观察＞存储器），或者点击调试工具栏中的  图标按钮 。此项操作之后会出现一个如附图 5.16 所示的“Memory Window Options”（存储器选项）对话框。这个对话框可用来选择存储器窗口中的字符变量。

•在对话框中键入所希望的字符。

•点击“OK”后，就会出现存储器窗口。

如果须要调整已经激活的存储器窗口，可以通过在存储器窗口中右击鼠标的方法调出语

境菜单，然后在菜单中选择相应的命令属性“Properties”，这时会弹出如附图 5.16 所示的存储器选项对话框，，在对话框中输入相应的字符即可完成调整。



附图 5.16　存储器选项对话框

如果要修改存储器中的某些数据，可以在存储器窗口中双击须要修改数据的地址，然后对数据进行修改。也可以选择菜单命令“Edit＞Memory-Edit”（编辑＞存储器）。地址？

三．项目管理和文件编辑

在 CC’C2000 开发平台中，由于允许使用 C、C＋＋和交叉汇编方式设计用户系统，因此，采用了 VC 等现代计算机软件系统的集成化管理方法，把用户系统叫做“项目”。这种方法不仅可以方便用户对系统软件和硬件的调试，同时对于比较大的系统也有利于实现集成化开发。这种软件开发方式与小系统软件中的模块化开发方法相类似，但比模块化方法灵活，同时还具有软件的遗传应用特征。

1．项目管理

CC’C2000 开发平台对用户系统实行项目管理，使用户系统的编制和调试过程变得简单明了。CC’C2000 开发平台中会建立不同独立程序的追踪信息，通过追踪信息对不同的程序进行分类管理，建立起不同的程序库和目标程序。在一个项目中将记录如下内容：

•源代码文件名和目标代码文件库；

•编译器、汇编器和链接选项；

•所包含的独立文件。

启动 CC’C2000 后，会自动显示出项目管理窗口。如果没有显示项目管理窗口，可以使用菜单命令“View＞Project”（观察＞项目）激活。可以在窗口中通过用鼠标点击文件管理页或书签页底部的选择标记选择不同的页。

使用项目管理窗口可以十分方便地完成程序管理，项目管理窗口如附图 5.17 所示。在项目管理窗口中会显示出项目完整的内容、项目的组织情况以及与项目关联的程序。所有的项目操作都可以在项目管理窗口中完成。同时，由于在 CC’C2000 项目管理中提供了相关的操作命令，大大地提高了用户系统的开发速度。例如，在调试一个由 5 个独立程序组成的用户系统时，如果在一次调试中只修改了其中的一个程序，利用项目管理命令就只须要对新修改的程序进行重新编译，而不须要对所有的程序都进行重新编译。

2．项目管理窗口的功能

项目管理窗口分为两页，一页是文件观察页（File View），另一页是书签页（Bookmark）。

（1）文件观察页

文件观察页中的第一个文件夹是 GEL－－General Extension Language 文件夹。GEL 是一种解释性计算机语言，可以用来编写有关集成开发环境 IDE 和访问目标存储器设置的功能函数。在 CC'C2000 中，GEL 文件用来修改和保存用户所设置的开发环境，例如窗口的位置、自动打开的窗口、工具栏的位置等。

文件观察页中显示的另一个内容是项目所包含的所有文件库和文件。在一个项目中，文件被分成几个不同的库进行管理，如附图 5.17 所示。

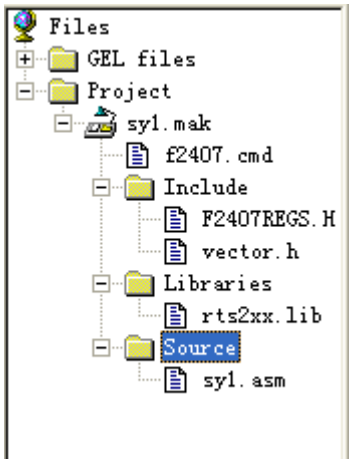
•Include（包含）：Include 中包含有以.h 为扩展名的文件（即 C 语言文件中的头文件等）。

•Libraries（库）：Libraries 中存放了所有以.lib 为扩展名的库文件。

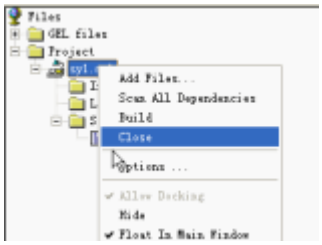
•Source（源）：Source 中包含了所有以.C 和*.asm 为扩展名的源文件。

程序中的链接命令文件（扩展名为.cmd）会直接显示在项目文件下。项目文件是项目的管理文件，以.mak 为扩展名。

一旦成功地创建了项目，将在源文件下以树型结构显示它所包含的所有包含文件。双击目标图标或点击靠近源文件的“＋”符号，可以扩展树型结构，显示包含文件。 在项目管理窗口中对文件的操作方法如附图 5.18 所示，具体方法如下：



附图 5.17 项目管理窗口



附图 5.18 文件操作按钮菜单

•要增加一个项目文件，用鼠标右键点击“Project”，显示出如附图 5.19 所示的按钮菜单，选择其中的“Open project”，再根据给出的“Project Open”对话框打开所需要的文件。

•要编辑一个文本文件，可用鼠标右键点击“Project”所包含的文本文件，显示出如附图 5.20 所示的按钮菜单，选择其中的“Open”命令。这时在工作界面中会显示出文件编辑窗口，就可以进行文件编辑操作了。此外，还可以用鼠标直接双击文本文件，即可打开所选文本文件。

•在对不同源文件设置编译选项时，可以在附图 5.20 中选择“File Specific Options”，这时会弹出如附图 5.21 所示的编译先项对话框。可以根据需要选择对话框中的参数，最后点击“ 确定 ”退出。

•要编译一个文件，用鼠标右键点击选定的文件，显示出如附图 5.20 所示的按钮菜单，在其中选择“Compile file”。

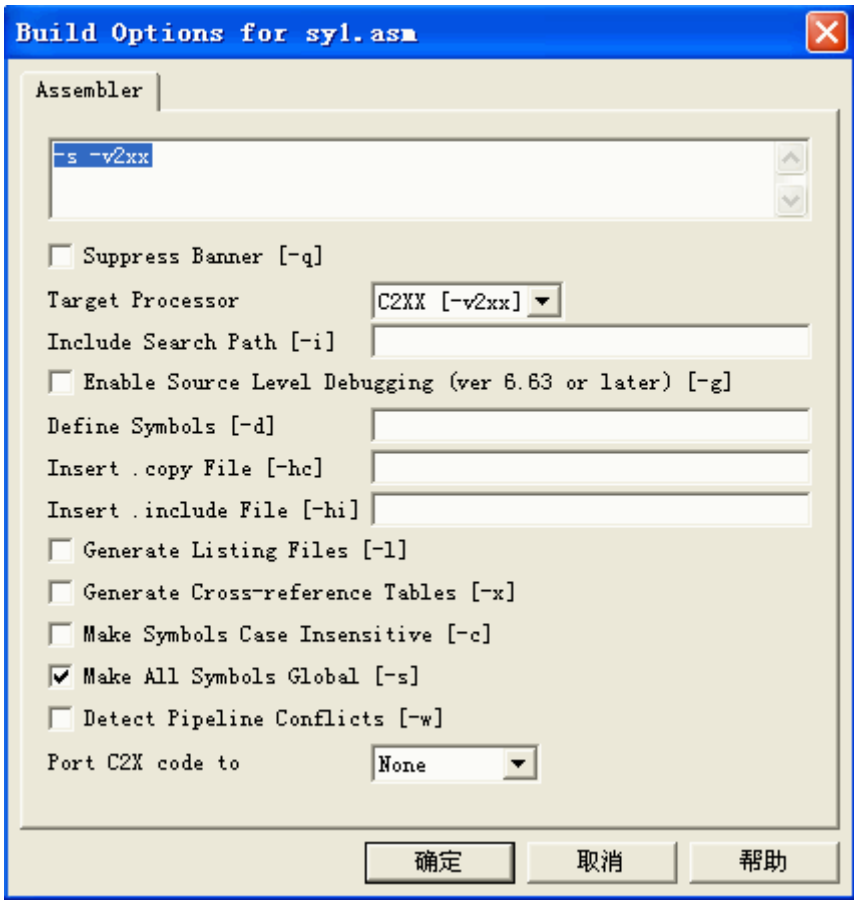
•要建立一个完整的项目，用鼠标右键点击选定“Project”的文件，显示出如附图 5.22 所示的按钮菜单，在其中选择“Build”即可完成。

•要从项目中移走一个文件，用鼠标右键点击选定的文件，显示出如图 5.16 所示的按钮菜单，在其中选择“Remove from project”。

附图 5.19　项目操作按钮菜单

附图 5.20　文件操作按钮菜单

附图 5.22　项目操作按钮菜单



附图 5.21　编译选项对话框

（2）书签页

书签页中显示了文件编辑，以帮助程序管理和调试。书签页的操作可以使用编辑工具。编辑工具栏如附图 5.23 所示，其中各命令按钮的功能如下：

附图 5.23　编辑工具栏

-  ——设置括号标志，以便能实现括号匹配。
-  ——寻找下一个开始括号（例如，如果找到了，对括号内的文字做标记。如果再次按此命令按钮，可以查找所标记文字中的下一层括号）。

•  ——把光标移动到配对的括号中。

•  ——把光标移动到下一个单括号中。

•  ——左移选定文字块一个制表位。

•  ——右移选定文字块一个制表位。

•  ——在文件窗口中光标所指当前行上设置或撤销书签，即如果已经有书签则撤销书签，如果没有书签则设置一个书签。

•  ——在当前文件窗口中查找下一个书签。

•  ——在当前文件窗口中查找上一个书签。

•  ——打开书签设置对话框。

3．项目操作

有关项目管理的信息都存放在以.mak 为扩展名的项目文件中。在进行用户系统开发时可以通过以下方法建立新的项目管理文件，或打开已存在的项目管理文件。

（1）打开已有项目

要打开一个已经存在的项目，可以选择菜单命令“Project＞Open”（项目＞打开），弹出文件选择的对话框，这时可以选择文件所在的文件夹，而后选择已经存在的项目管理文件。也可以用前面介绍的项目管理窗口打开一个已经存在的项目。

（2）建立新项目

在开发新的应用系统时，须要建立新的用户项目。具体操作方法如下：

•选择菜单命令“Project＞New”（项目＞新建），在弹出的文件选择对话框中选择要保存项目文件的文件夹，输入项目文件名，再用“保存”退出。注意：由于不同项目使用的源文件以及 C 语言头文件不尽相同，所以最好每个项目选择一个文件夹，以便把不同的项目区分开。

•在项目管理窗口中，可以察看建立的项目以及项目所包含的源文件、目标文件等是否加入到项目管理文件的相应文件夹中。

（3）关闭正在操作的项目

如果须要退出目前正在操作的项目，则只要用鼠标右键点击项目管理窗口中的“Project”，在显示出的按钮菜单中选择“Close”项；也可以选择菜单命令“Project＞Close”。

四．CPU 寄存器操作

在带有 CPU 的系统开发调试过程中，总须要对 CPU 或 DSP 的内部寄存器进行直接调整（即修改其内容），这样不仅可以对用户程序进行分块调试，也可以对用户的硬件目标系统功能进行分块调试，是提高系统开发速度的一种重要技术。为支持这种技术，在 CC’C2000 开发平台中，提供了寄存器窗口用于观察目标系统 DSP 器件的 CPU 寄存器和外设寄存器，还可以通过编辑寄存器对话窗口对目标系统 DSP 器件寄存器的内容进行编辑修改。

1．观察寄存器内容

要观察目标系统 DSP 器件 CPU 寄存器的内容，可以用以下两种方法之一调出 CPU 寄存器窗口：

•选择菜单命令“View＞CPU Registers＞CPU Registers”（观察＞CPU 寄存器＞CPU 寄存器）；

•在调试工具栏中选择  按钮。

在 CC’C2000 界面中 CPU 寄存器显示窗口如附图 5.24 所示。

2．编辑寄存器工作

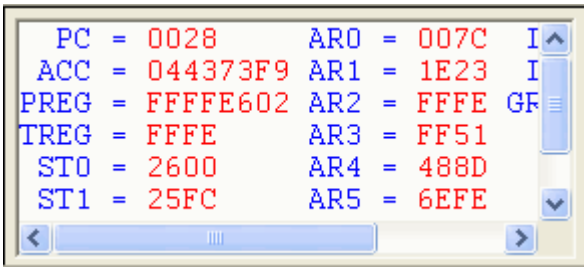
编辑寄存器时，须要打开编辑寄存器对话框。可以用以下三种方法之一完成编辑寄存器对话框的操作：

•选择菜单命令“Edit＞Edit Register”（编辑＞编辑寄存器），出现编辑寄存器对话框后进行编辑；

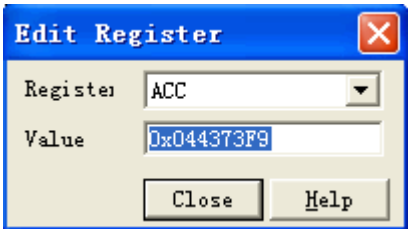
•在寄存器窗口中双击寄存器名，出现编辑寄存器对话框后进行编辑；

•在寄存器窗口中右击鼠标，在语境菜单中选择编辑寄存器命令“Edit Register”，出现编辑寄存器对话框后进行编辑。

编辑寄存器对话框如附图 5.25 所示。编辑寄存器对话框提供如下选项：



附图 5.24　寄存器显示窗口



附图 5.25　编辑寄存器对话框

•Register（寄存器）：输入想要编辑的寄存器名，或者在下拉菜单中选择要编辑的寄存器。

•Value（数值）：在编辑寄存器对话框中以十六进制（hex）显示选定寄存器的当前值，可以在此输入新值（格式是带前缀 0x 的十六进制数），也可以输入 C 表达式。

调整完寄存器数值后，点击“Close”，会自动保存新数值。注意：上述功能不能用于目标仿真器中的外设寄存器。

五．装载 COFF 文件

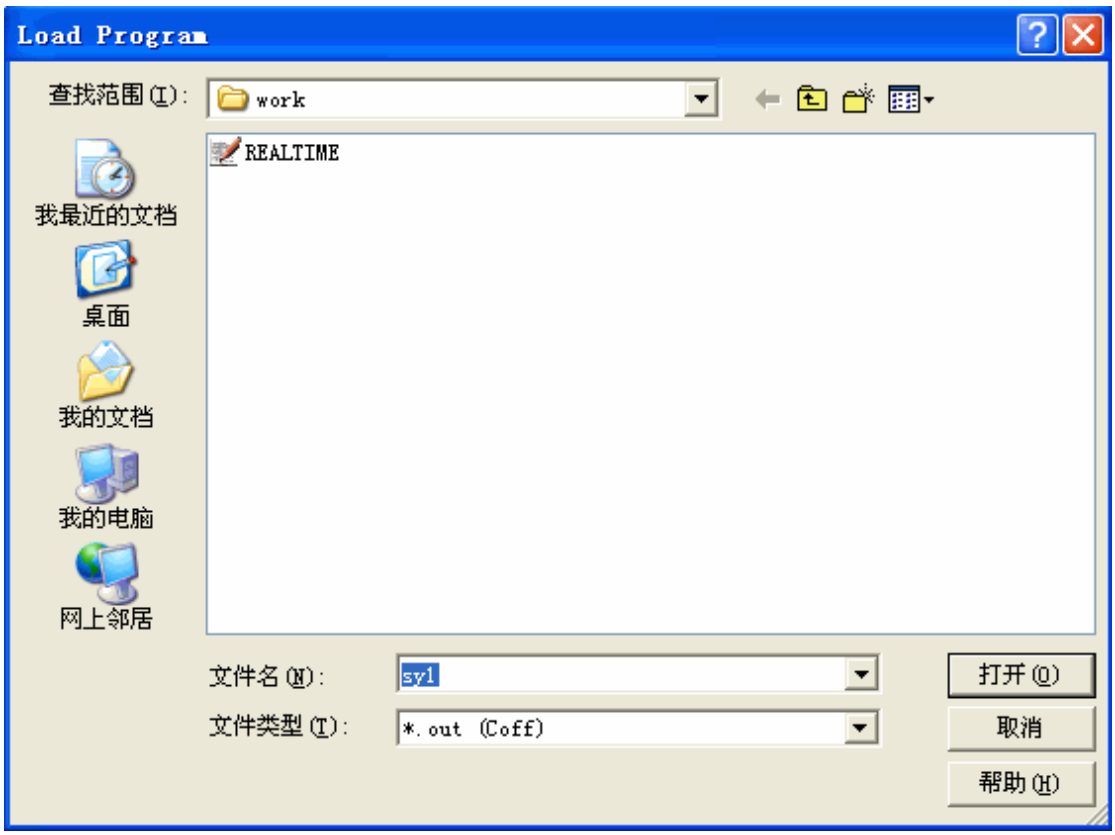
在现代 DSP 系统软件编制和调试过程中，有一种十分重要的技术叫做“分段模块”技术。分段模块的概念是：如果程序或数据代码能够在目标存储器中占据一个链接的地址空间，并能被重新放置在新的地址上，这种程序或数据模块就叫做“分段模块”。

由此可见，分段模块实际上是借鉴了微机操作系统编程的模块技术（例如文件）。在现有微处理器地址管理方式下，分段模块技术具有提高执行速度的特殊功效。

在处理分段模块化的程序模块时，要使用 COFF 文件格式。COFF 是一般目标文件格式的英文 Common Object File Format 的缩写，是一种支持采用分段模块概念程序的二进制文件格式。所有的 COFF 分段模块都可以相互独立放置在存储器中，用户可以把任何一个分段模块放置在存储器的任何位置上。

1．把 COFF 文件放置在目标板中的操作

（1）用菜单命令“File＞Load Program”（文件＞装载程序），打开如附图 5.26 所示的“Load Program”（程序装载）对话框。



附图 5.26　程序装载对话框

（2）在对话框“Load Program”中，选择所需要的文件，然后点击“ 

”按钮，这样就可以把所选定的数据和来自 COFF 文件的符号信息装载进目标系统中。

4．设置程序装载选项的操作

选择菜单命令“Options＞Program Load”（选项＞程序装载），可以设置与程序装载有关的一些选项。这些选项是：

•Perform verification after Program Load（程序装载后性能验证）：缺省条件下，CC’C2000 会检查这个检查框，目的是确认装载的程序正确无误。如果关闭这个选项，则 CC’C2000 将不再对装载的程序进行正确性检查。使用这个选项可以保证程序能正确装载，但对一些比较大的程序，由于要进行确认检查，所以需要较长的装载时间。

•Load Program After Build（创建后装载程序）：如果检查了这个选项与建立项目有关的可执行程序会被立即装载。这可以保证目标处理品中具有项目创建所形成的最新的符号信息。

六．程序运行操作

在 CC’C2000 中，提供了多种调试程序的运行操作方法。这些操作方法都以调试工具的方式存放在调试工具栏中。调试工具栏如附图 5.28 所示。此外还可以使用调试菜单“Debug”中相应的命令控制程序运行。

附图 5.28　调试工具栏

1．单步执行操作

在工具栏中提供了相应的按钮，可以实现单步调试操作。

（1）  Step Into（单步执行）：在调试过程中，可以使用这个按钮完成单步执行操作，即按下这个按钮后，可以完成一条汇编指令的执行。此外，还可以通过选择菜单命令“Debug＞Step Into” 完成相同的操作。如果所调试的是 C 程序代码，则这个命令将执行一条 C 指令。

（2）  Step Over（单条执行）：单条执行的意思是每发出一个单条操作命令，能够执行一条或一段程序。例如，对于一般的程序指令，按一次单条执行的结果就执行一条程序指令（如一条汇编指令或 C 语句）。如果所执行的是一个程序调用语句，则单条执行命令在执行完所调用的子程序后才返回，也就是把所调用的程序作为一条指令完成。这是与单步执行指令的根本区别。单步执行命令在执行程序调用指令时，仍然是执行完一条指令后即停止（进入所调用的程序），而不是把调用指令和调用程序作为一条指令执行。

（3）  Step Out（跳出子程序）：这条命令用于子程序的执行操作中。在调试过程中，有时会进入某个子程序执行程序。当执行到子程序的一部分时，可能不再关心该子程序其它部分的运行情况，这时就可以使用跳出子程序命令，使其直接返回到调用该子程序的位置上，然后自动执行程序的后续指令，直到返回到调用该子程序的指令。此外，还可以通过选择菜单命令“Debug＞ Step Out” 完成相同的操作。

（4）  Run to Cursor（执行到光标）：在调试过程上，执行到光标命令可以提供十分方便的调试程序方法，只要在返回窗口中设置一个光标（点击该指令所在行），就可以从程序执行的当前位置一直执行到反汇编窗口中光标所在位置。

此外，还可以通过设置实现一个操作命令的重复操作，具体的操作方法如下：

•选择菜单命令“Debug＞Multiple Operations”（调试＞操作设置），这时会弹出一个如附图 5.29 所示的“Multiple Operations”（操作设置）对话框，；

•在下拉菜单中选择所需的操作方法；

•Count（计数）：设置相应的操作次数；

•点击“ 

”。

2．程序的实时运行

如果需要一次命令执行多条指令，或让所调试的程序处于完全运行状态（不是一步一步地执行），可以选择调试工具栏中的实时运行命令。

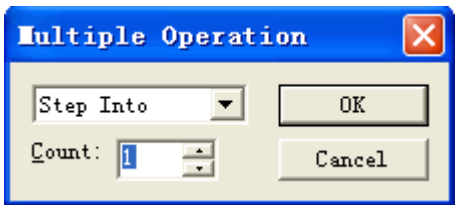
（1）  Run（运行程序）：这个命令的功能是，从当前程序指针（PC）所指的位置上开始连续执行程序，直到遇到一个断点或到达程序结束为止才能停止运行。这种运行方式可以使程序在更接近实际工作时的情况下运行。除了使用调试工具栏中的命令按钮外，还可以通过选择菜单命令“Debug＞Run” 完成相同的操作。

（2）  Halt（停止）：这个命令用来调整正在执行的程序。也可以通过选择菜单命令“Debug＞Halt”完成相同的操作。

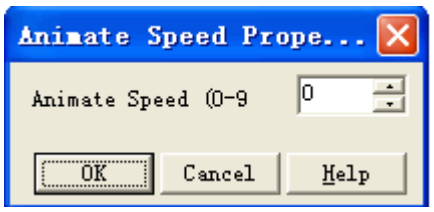
（3）  Animate（动画）：这是一个在断点支持下快速调试程序的命令。在执行该命令前

预先设置好程序断点，然后，每按动一下  按钮，就会从当前程序位置执行到所遇到的第一个断点。之后，继续按动就会执行到第二个断点所在的程序位置，如此下去直到遇到最后一个断点。执行到最后一个断点时应当注意，如果再次按动画按钮，程序就会因为找不到断点而一直执行下去，能否停止与程序的结构有关。如果程序不能自动停止执行，则须要使用“Debug＞Halt”命令终止程序的执行，这是在调试中应当注意的。也可以通过选择菜单命令“Debug＞ Animate”完成相同的操作。

动画的执行速度可以通过设置选项进行设置。设置的方法是选择菜单命令“Option＞Animate Speed”（选项＞动画设置），这时会弹出一个如附图 5.30 所示的“Animate Speed Prope…”（动画速度设置）对话框，。



附图 5.29　操作设置对话框



附图 5.30　动画速度设置对话框

通过对话框中输入或选择相应的数据，最后点击 

按钮，就可以完成动画速度的设置。

（4）Run Free（自由运行）：这是一条全速执行用户程序的操作命令，与运行程序命令不同的是，自由运行命令忽略所有的断点，从当前用户程序指针所指位置开始全速执行用户程序。要执行这个操作命令，只要选择菜单命令“Debug＞Run Free”即可。只能使用“Debug＞Halt”命令停止程序的自由运行。如果用户使用的是利用 JTAG 器件链接 PC 机和开发系统，则在自由运行时 PC 机与开发系统之间没有通信联络。

七．其它有关的基本操作

1．对目标处理复位

在调试中，有时要对 DSP 器件进行复位操作。这种操作可以通过 CC’C2000 实现。在 CC’C2000 中提供了一些有关目标系统 DSP 器件的复位命令。

（1）Reset DSP（DSP 复位）：选择菜单命令“Debug＞Reset”（调试＞DSP 复位），可以完成对 DSP 的复位操作，使目标系统 DSP 器件恢复到上电初始状态，并且终止当前执行的用户程序。如果用户使用的是基于内核驱动的器件，并且目标板对复位命令没有响应，则 DSP 的内核会被破坏，这就须要再次进行内核装载。

（2）Load Kernel（装载内核）：在基于内核的调试系统中，内核的任务是完成开发系统与 PC 机之间的通信任务。如果用户使用的不是基于 JTAG 的调试器，而是基于内核的调试系统，则当内核被破坏后器件驱动无法与目标器件之间进行通信，开发系统也就无法正常工作了。这时，就必须利用菜单命令“Debug＞Load Kernel”（调试＞装载内核），再次向开发板装载内核程序以恢复正常的工作通信。

（3）Restart（重新启动）：这是一个使 CC’C2000 程序指针恢复到用户程序入口地址的命令。通过选择菜单命令“Debug＞Restart command”（调试＞重新启动），可以把程序指针 PC 恢复为用户程序的入口，但并不执行程序。

（4）Go Main（转移到 main）：这是用于调试 C 语言用户程序时的命令。通过选择菜单命令“Debug＞Go Main”（调试＞转移到 main），CC’C2000 把一个临时性的断点设置在用户程序关键字“main”的位置上，并从此开始执行用户程序，一直到遇到一个用户设置的断点或执行“main( )”命令时，才停止用户程序，并且撤销临时断点。如果用户程序的执行停止在

“main( )”，则相关的其它源文件会被自动的装载到 CC’C2000 中。

5．刷新窗口

在仿真开发中，所有窗口中的数据都代表目标系统当前的状态；但如果把窗口链接在探测点“Probe Point”上时，则窗口中的数据就可能不是最新的数据（因为其不能随数据的变化而变化）。为了能观察到最新的系统状态，就必须对窗口进行刷新。

刷新窗口的方法很简单，只要选择菜单命令“Windows＞Refresh”（窗口＞刷新），即可以完成窗口刷新，这时通过窗口观察到的就是执行刷新命令时目标系统所处的状态。

7．保存和恢复操作界面

用户在操作中会对 CC’C2000 的操作界面进行相应的调整，以便把工具栏和窗口放置在自己觉得方便的位置上。当结束一次开发工作时，总希望能把当前的界面结构（工作界面）保存下来，以便下一次操作时继续使用，这时就要保存操作界面。同样，在使用中有时也须要恢复为原有的操作界面，这时就要恢复界面。

（1）操作界面的有关信息保存在以“.wks”为扩展名的文件中，文件中记录了如下内容：

•父窗口（包括尺寸和位置）。

•子窗口（包括尺寸和位置）。

•断点（breakpoints）、探测点（probe points）和描述点（profile points）。

•描述选项（profile points）。

•当前项目。

•当前装载的 GEL 功能。

•存储结构。

•动画速度选项。

•文件 I/O 的设置。

（2）以下内容不会被记录在操作界面文件中，也就是在下次调用该界面时不会被恢复。

•当前字体。

•当前色彩选择。

•目标存储器、程序和处理器的状态。

•编辑和查找／替换工具中的内容。

•“build”窗口中的错误和进程信息。

•GEL 输出窗口。

•扫描过程中的临时性从属窗口。

•反汇编格式选项。

（3）在使用 CC’C2000 中，可以通过如下操作保存或恢复操作界面。

①保存操作界面操作：

•选择菜单命令“File＞Workspace＞Save Workspace”（文件＞操作界面＞保存操作界面），这时会弹一个如附图 5.34 所示的“Save Workspace”保存操作界面对话框；

•在对话框中输入一个文件名作为所保存工作界面的文件名；

•用鼠标点击“  ”。

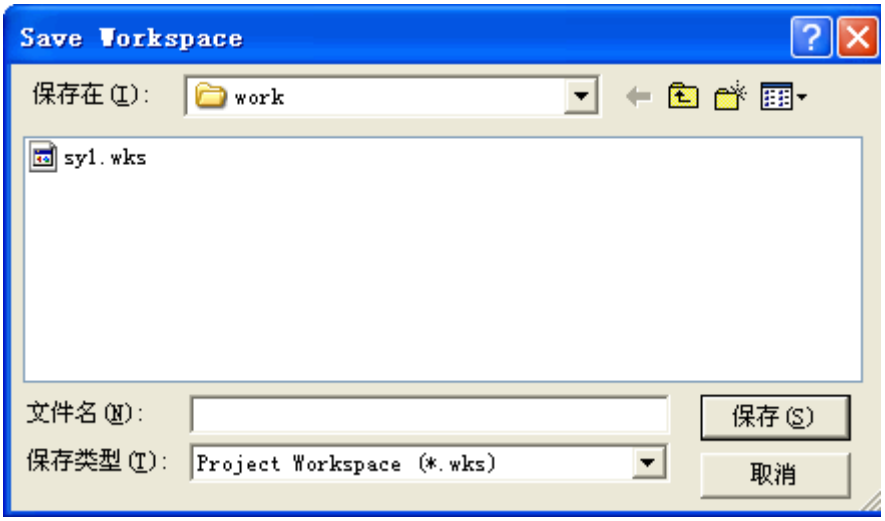
一般情况下，在退出 CC’C2000 时，系统会自动地把当前操作界面保存在“default.wks”文件中。

②载入操作界面操作：

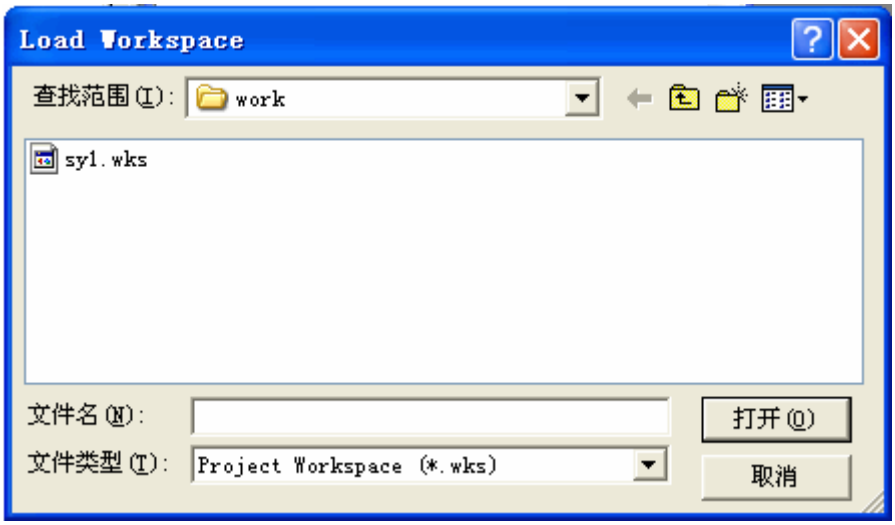
•选择菜单命令“File＞Workspace＞Load Workspace”（文件＞操作界面＞载入操作界面），这时会弹一个如附图 5.35 所示的“Load Workspace”载入操作界面对话框；

•在对话框中输入要装载的工作界面的文件名；

•用鼠标点击“ 打开(O) ”。



附图 5.34　保存操作界面对话框



附图 5.35　载入操作界面对话框