第四章　MSP430 开发环境简介

在前面一章，我们讲解了 MSP430 的寻址方式与指令集,这里介绍 MSP430 的开发环境的使用。MSP430 的开发软件较多,但常用的或方便的要数 IAR 公司的集成开发环境:IAR 系统嵌入式 Workbench 以及调试器 C-SPY。

4.1　Embedded Workbench 嵌入式工作台

IAR Embedded Workbench（嵌入式工作台）为开发不同的目标处理器的项目提供强有力的开发环境，并为每一种目标处理器提供工具。Embedded Workbench 嵌入式工作台使用项目模式来组织应用程序。它有如下一些特点:

通用性

可以在 WINDOWS 环境下运行

分层的项目表示

直观的用户界面

嵌入式工作平台工具与编辑器全集成

全面的超文本帮助

嵌入式工作平台编辑器

可以同时编辑汇编与 C 语言原文件

汇编程序与 C 语言程序的句法用文本格式与颜色区别显示

强有力的搜索与置换命令,而且可以多个文件搜索

从出错列表直接跳转到出错的相关文件的相关语句

可以设置在出错语句前标志

圆括号匹配

自动缩进,可以设置自动缩进的空格多少

每个窗口的多级取消与恢复

下面将介绍它的安装与使用。

4.1.1　Embedded Workbench 安装

在 WINDOWS 环境下，双击 FET_R202.EXE 或 FET_301.EXE。安装过程中使用默认值。对机器没有特殊的要求,目前的计算机都能满足内存,硬盘,机器速度的要求.安装完成以后应出现如图 4.1 的界面。

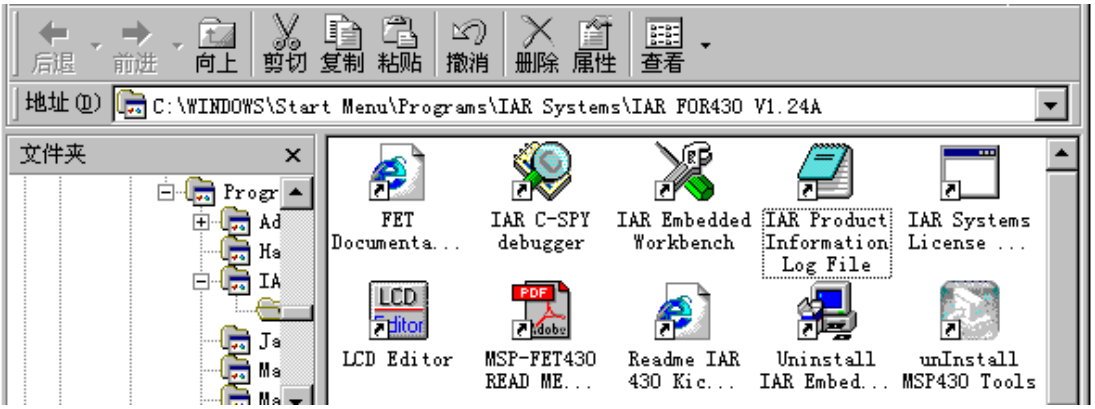

图 4.1　Embedded Workbench 安装成功图例

双击 IAR Embedded Workbench 图标，进入嵌入式工作台软件环境，可以进行程序的编辑，项目的管理，编译、连接等工作。双击 IAR C-SPY Debugger 图标，进入嵌入式调试环境，可以方便地进行程序的模拟调试，还可以下载程序到具体的器件（FLASH 型）进行联机在线调试。

4．1．2 Embedded Workbench 概述

IAR 嵌入式工作台 Embedded Workbench 为开发各种不同的目标处理器的项目提供强有力的开发环境，这一部分介绍在这个环境中使用项目模式来进行典型的用户应用程序的开发。IAR 嵌入式工作台 Embedded Workbench 被专门设计成能适合常用的软件开发项目的组织方式。设计者可能需要开发适合于不同版本目标硬件的应用程序的相应版本，但这些相应版本的应用程序可能有相同的部分源文件，那么使用项目管理方式就很方便。设计者只需要维护唯一的副本，就可以对应用程序的每一个版本进行改进。嵌入式工作台适合于维护用于建造应用程序的所有版本的源文件，允许设计者以树状体系结构组织项目，能一目了然地显示文件之间的依赖关系。

在树状结构中，目标位于最高层，它规定了设计者想要建立的应用程序的不同目标版本。对于一般的简单应用，可能只需要两个目标：DEBUG（调试）和 RELEASE（发布）。较为复杂的项目还可能有其他的目标。每一个组将一个、多个相关的源文件组合起来，每一个组有可能被包含在一个或多个目标之中。每一个源文件也可能被包含在一个或多个组中。当设计者使用项目（PROJECT）工作时，总有一个选定的当前目标（CURRENT TARGET）在项目窗口中，只有作为该目标成员（MEMBER）的组以及它们所包含的源文件才能被真正建立并由此产生目标代码。项目的树状结构如图 4.2 所示

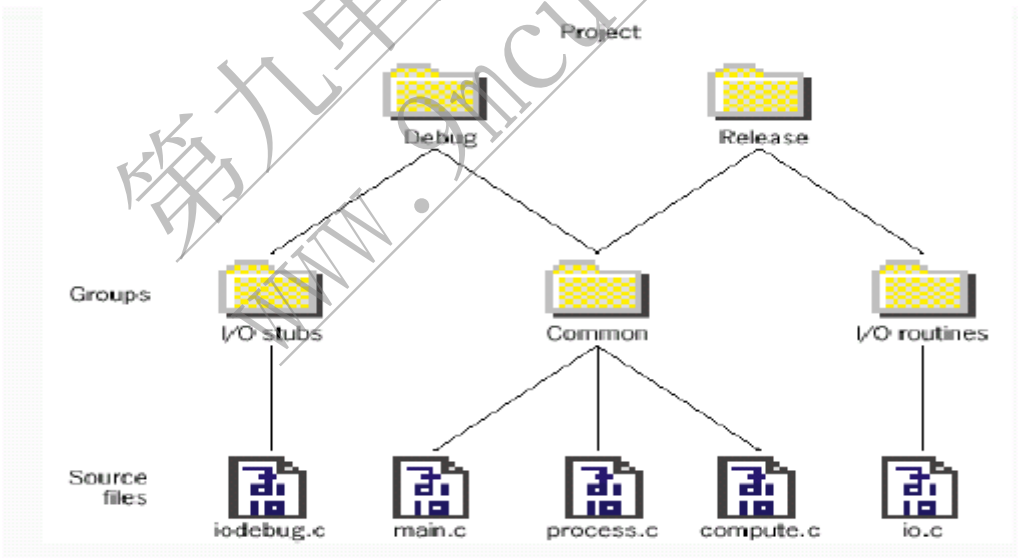

图 4.2 项目的树状组织结构

4．1．3 Embedded Workbench 使用指南

下面用一个具体的例子来说明怎样使用 IAR 的嵌入式工作台。这个例子很简单,就是工具套件中的使 P1.0 上连接的发光二极管灯闪烁的例子。

首先打开 IAR 的嵌入式工作台,在 WINDOWS 环境下依次点击:开始、程序、IAR SYSTEMS、IAR Embedded Workbench For MSP430 Kickstart、IAR Embedded Workbench,之后进入如图 4.3 所示的集成环境。

图 4.3　第一次进入 Embedded Workbench

进入该环境之后,应该首先建立一个项目。点击 File,New 之后出现图 4.4 所示界面,选择 Project,按确定按键,进入图 4.5 所示界面,然后选择目标 CPU、输入项目名称,按确定保存到你所选择的路径。

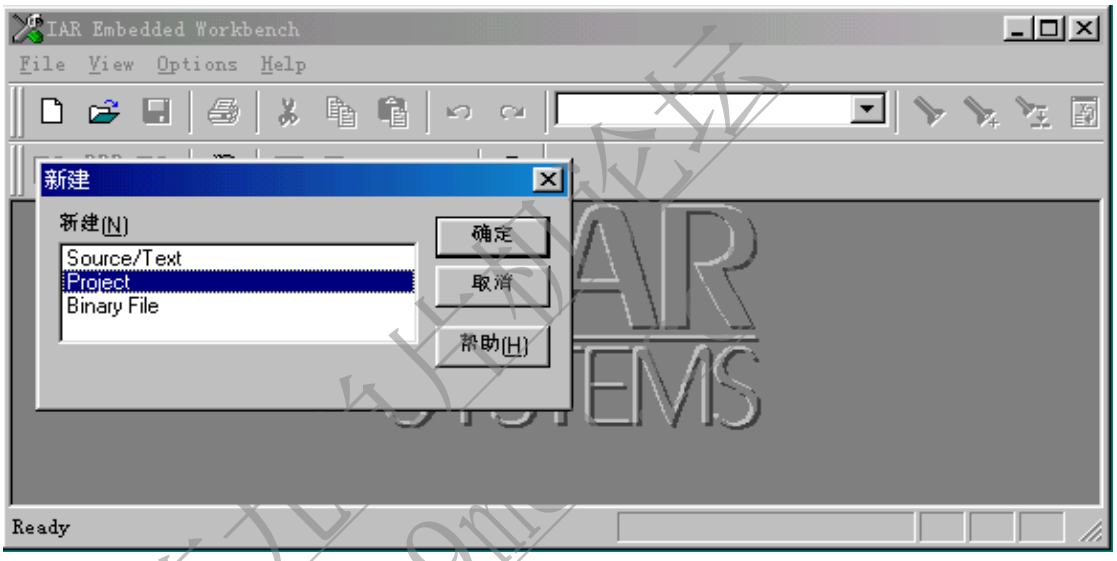

图 4.4　新建一个项目的界面

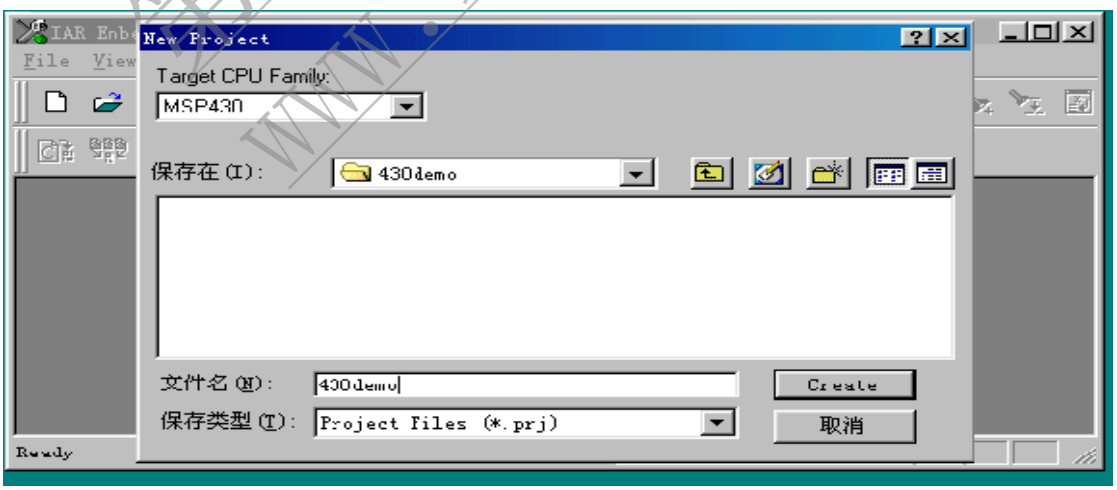

图 4.5　选择目标 CPU 并键入项目名称

此时我们可以看到在 430DEMO 项目下有一个 DEBUG 目标,但没有源文件,如图 4.6。点击 File、New 进入图 4.7 界面,选择 Source/Text 选项,按确定进入源程序编辑界面,如图 4.8,在这里进行源文件的编辑源程序编辑好之后保存为 430LED.S43。然而此时的项目 430DEMO 与源程序 430LED.S43 是孤立的,没有联系。

点击 Project、Files 出现图 4.9 所示界面,选中刚才所编辑的汇编源程序,按 ADD 键将源文件加入组中,在组里的文件框就可以看到所选中的文件出现在里面,如图 4.10 所示。这样我们的 430DEMO 项目里就有了 430LED.S43 程序,编译、连接之后产生的目标代码就是 430LED.S43 汇编后的代码。 按 DONE 以后,我们可以看到项目管理窗口(如图 4.11)与图 4.6 明显不同,图 4.6 中 Debug 目标里没有组,也没有源文件;而在图 4.11 中的 Debug 目标里有组 COMMON SOURCES,也有源文件 430LED.S43。

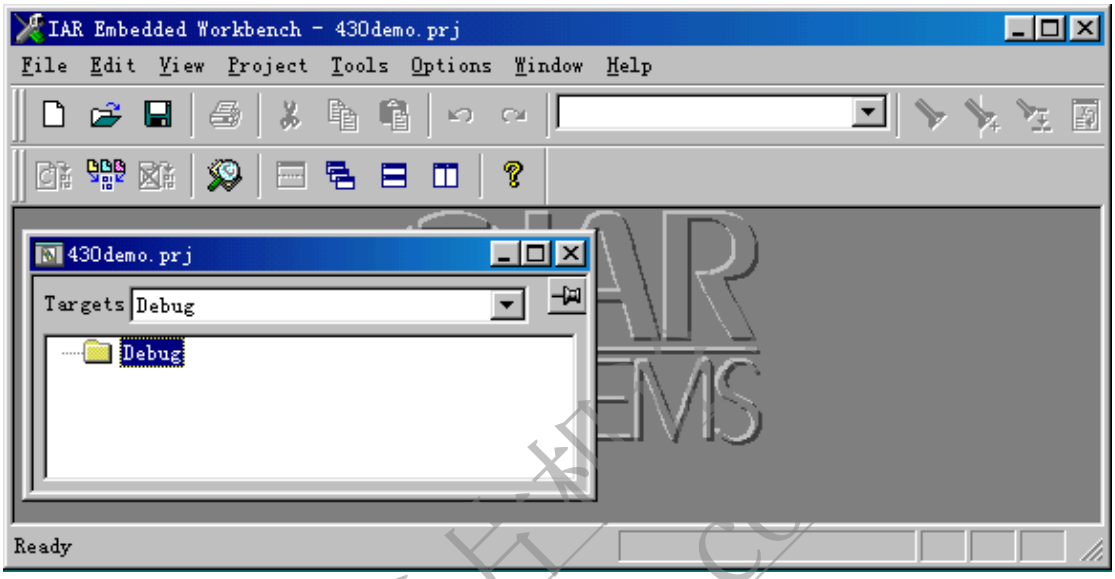

图 4.6 430DEMO 项目 但只有一个目标 没有组与源文件

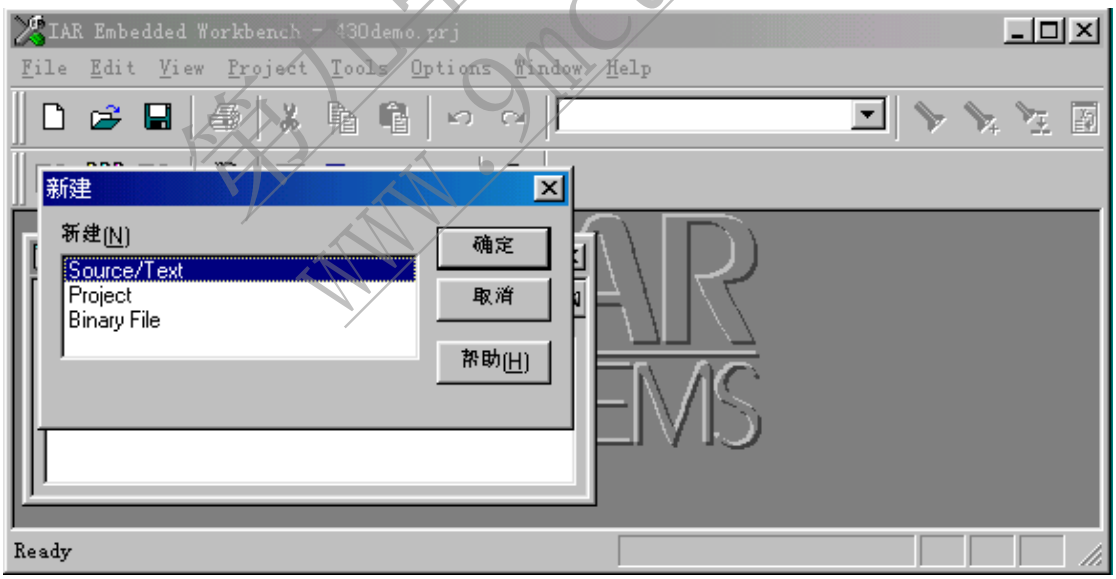

图 4.7 怎样进入编写源文件

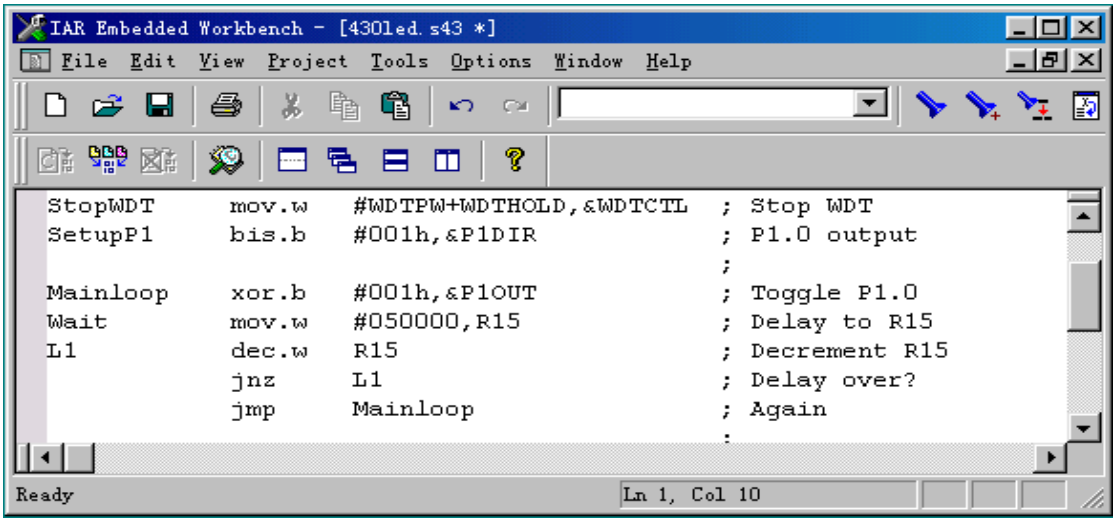

图 4.8　　源文件的编辑

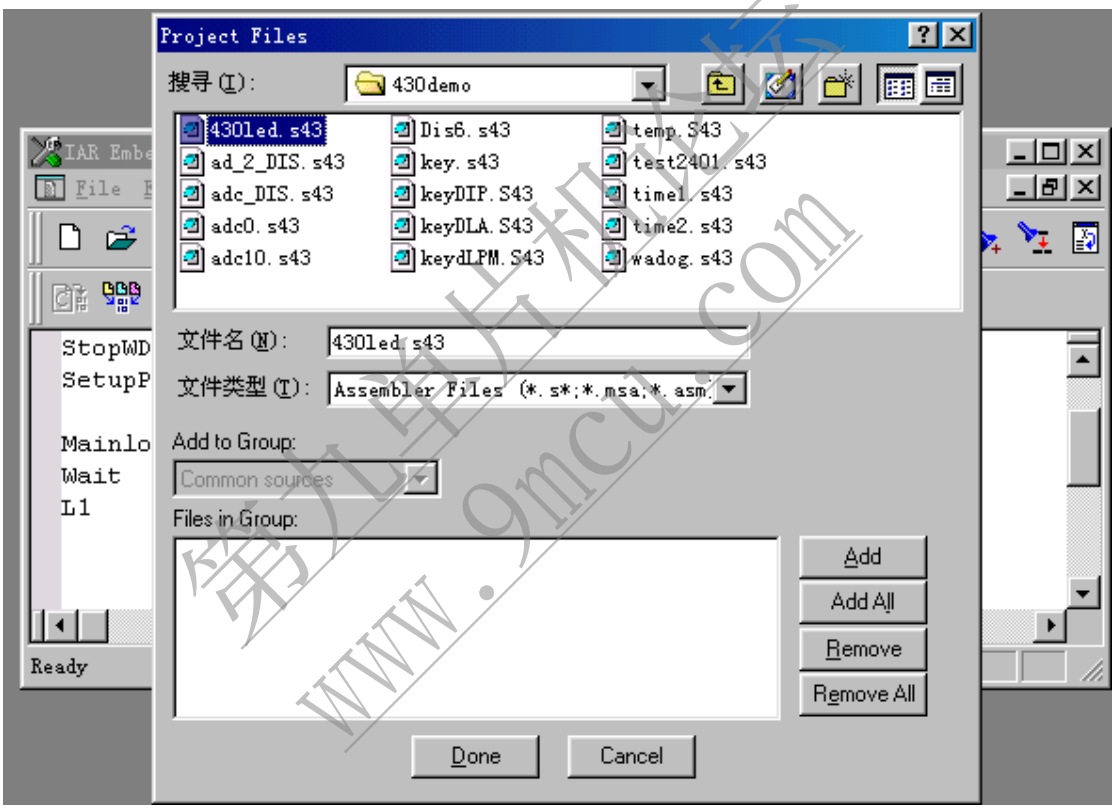

图 4.9　　将源文件加入组(Group)

程序清单如下:

```
                ORG     0F000h                  ;
RESET           mov.w   #300h,SP                ;初始化堆栈指针
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ;停止看门狗
SetupP1         bis.b   #001h,&P1DIR            ;P1.0 为输出
Mainloop        xor.b   #001h,&P1OUT            ;P1.0 求反
Wait            mov.w   #050000,R15             ;延时初值
L1              dec.w   R15                     ;
```

```
jnz     L1                      ;
jmp     Mainloop                ;反复执行
ORG     0FFFEh                  ;430 的第一条指令位置
DW      RESET                   ;
END
```

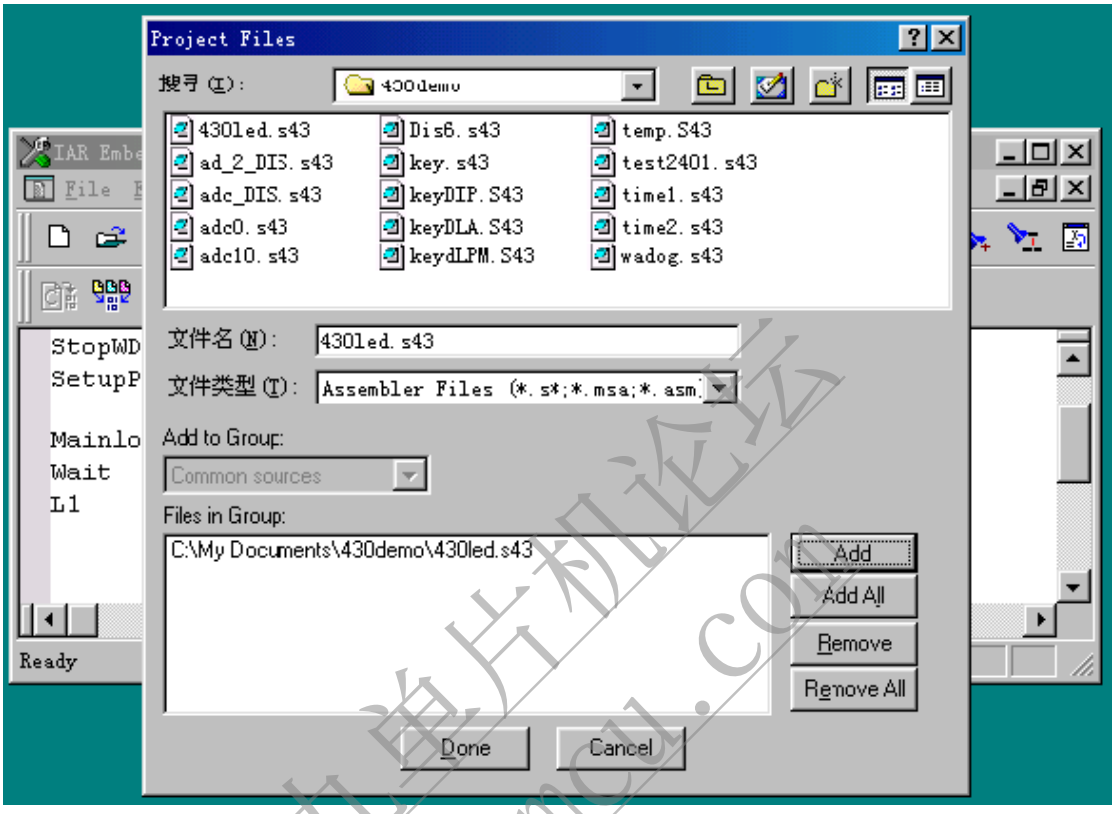

图 4.10 按 ADD 键加入源文件

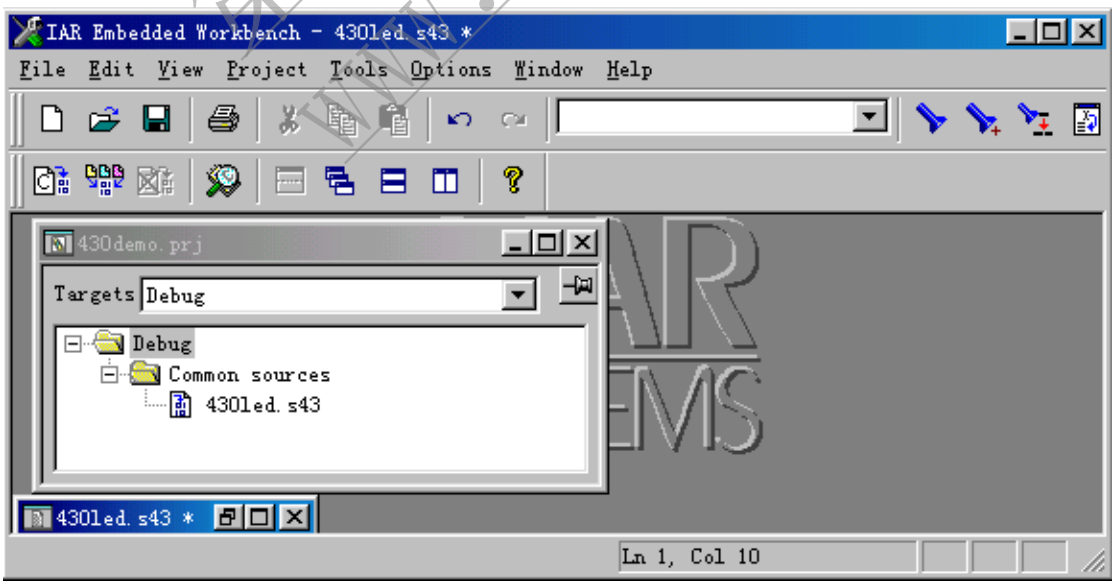

图 4.11 源文件已经加入项目中

点击 Project、compile 或 Ctrl+F9 或单击  进行文件编译或汇编，出现了图 4.12 所示的错误提示，任意点击某一个错误提示，系统回自动给出有错误的语句行，如图 4.13 所示。仔细阅读错误提示，发现是一些特殊功能寄存器没有定义。在源文件的开始加一句

```
#include  "msp430x11x.h"
```

如图 4.14 所示，再编译则没有错误了，因为在"msp430x11x.h"等等包含文件里已经有了关于器件中的特殊功能寄存器的地址说明。

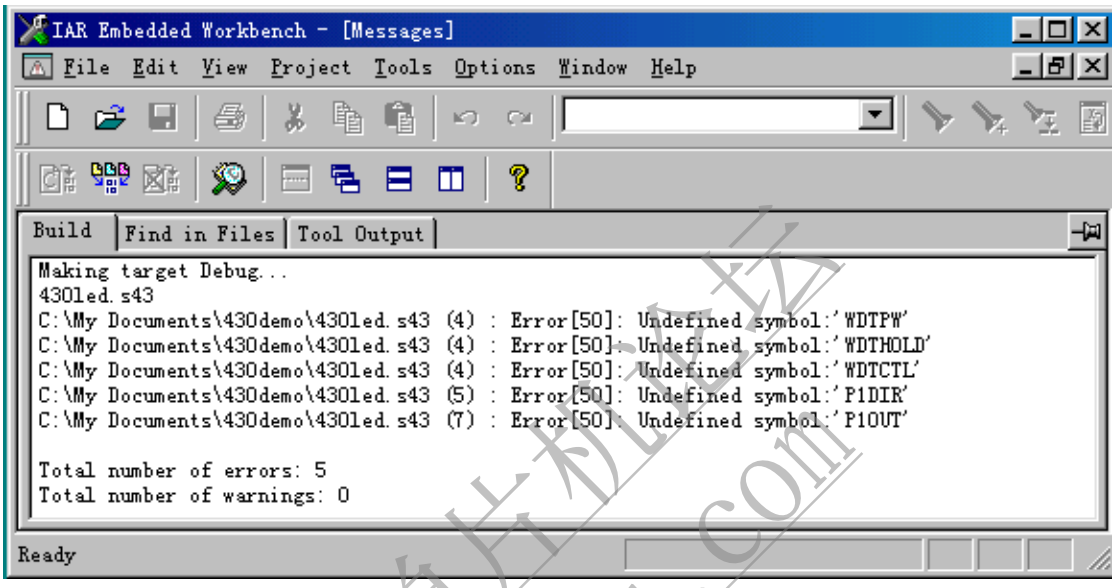

图 4.12　汇编之后产生的错误

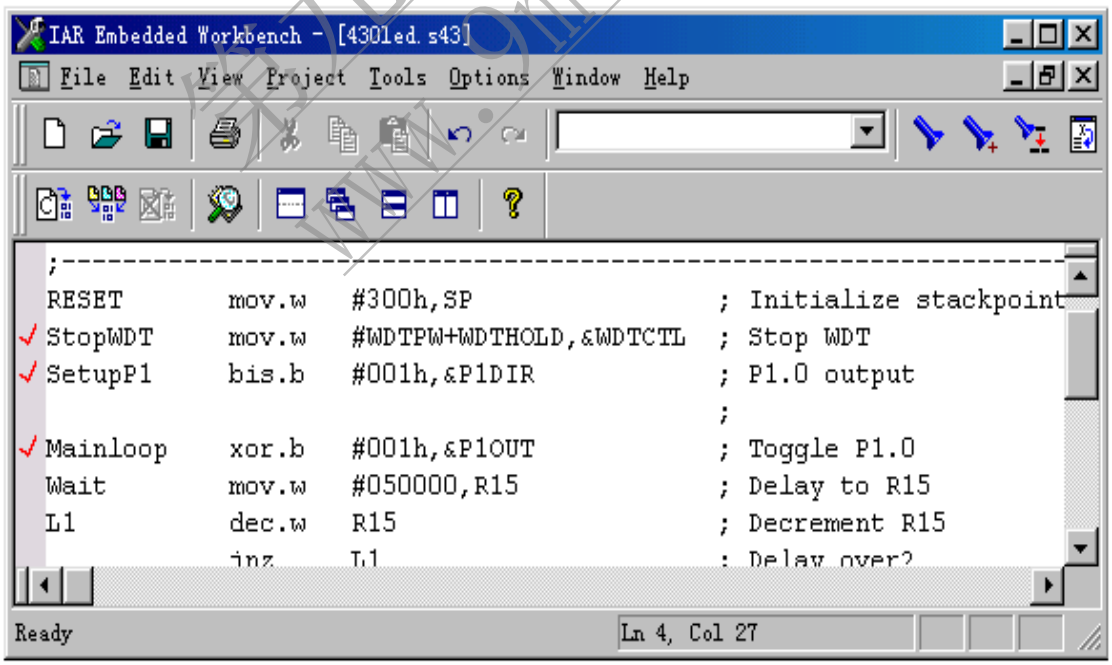

图 4.13　系统自动指出有错的语句行

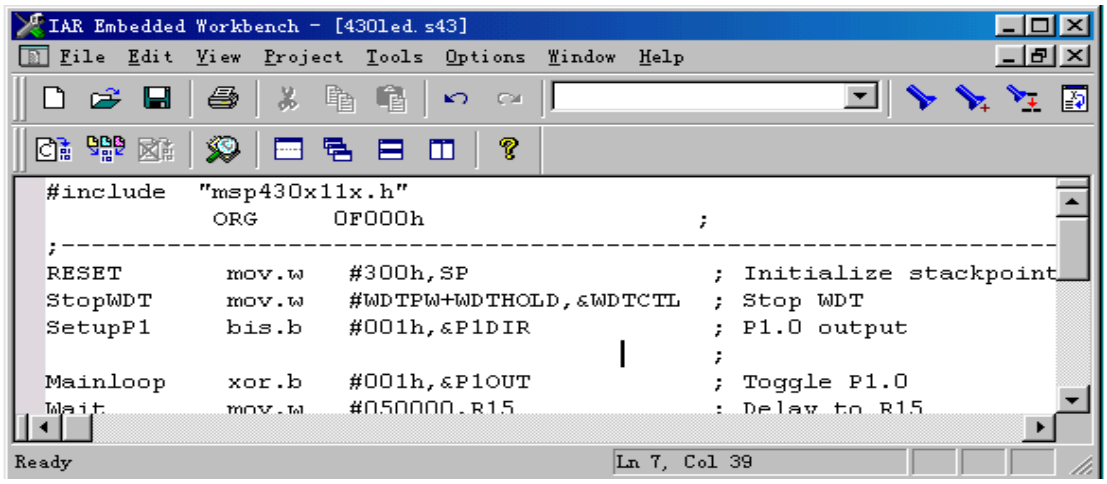

图 4.14 修改后的源文件

源文件编译通过之后,将要最终生成目标代码,这时可以点击 Project、Make 或 F9 或单击  进行连接以生成目标代码,出现了图 4.15 所示的错误提示,这是 因为环境的设置不对, 点击 Project、Options 进入设置界面后,点击 XLINK 进入如图 4.16 所示界面。在这

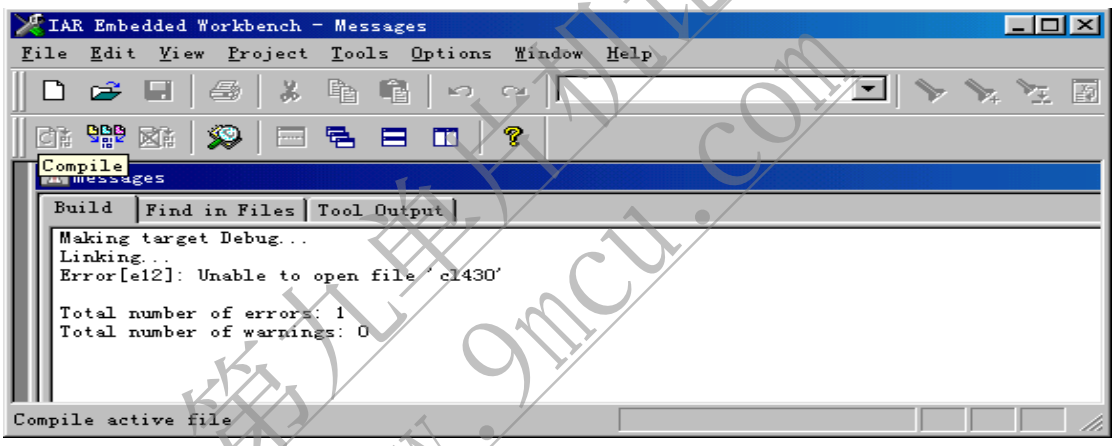

图 4.15 连接后出现的错误

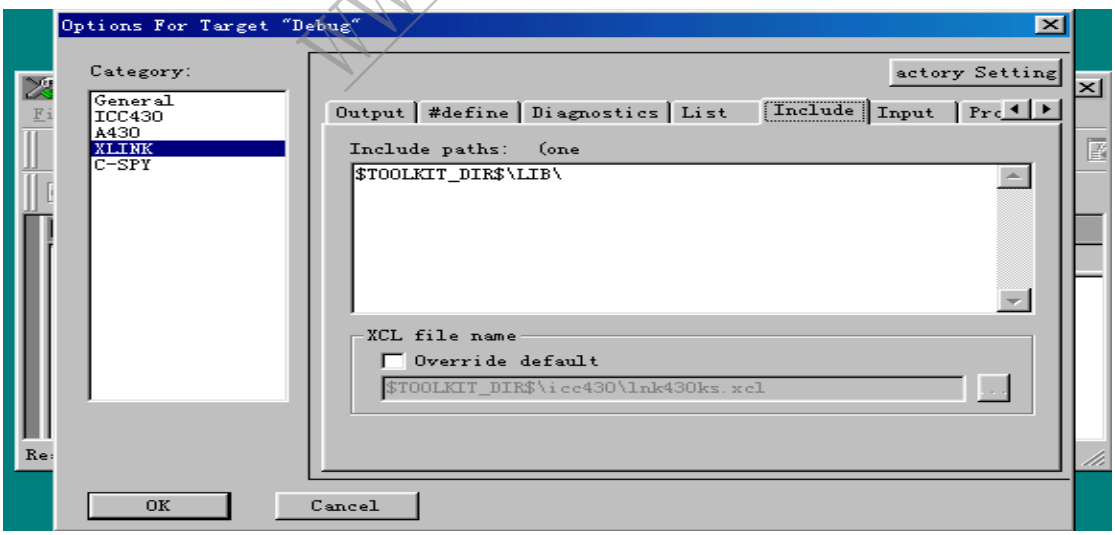

图 4.16 环境设置界面

些设置中,最关心的是 Include 选项,其中 Include 路径可以用默认的,不用改动,但

XCL file name 则需要改动,首先点中 override default,然后再根据设计者所使用的目标处理器(MPU/CPU)选择相关的 XCL 文件名,在这个设计中用的是 MSP430F1121,而且是汇编语言,所以这里选择 msp430F1121A.xcl。但如果用的是 C 语言源文件,则 msp430F1121C.xcl。设置好之后点击 OK 键以保存设置。这时再点击 Project、Make 或 F9 进行编译连接,则完全通过。剩下的事情就是对所编程序的进行调试以验证设计的正确。

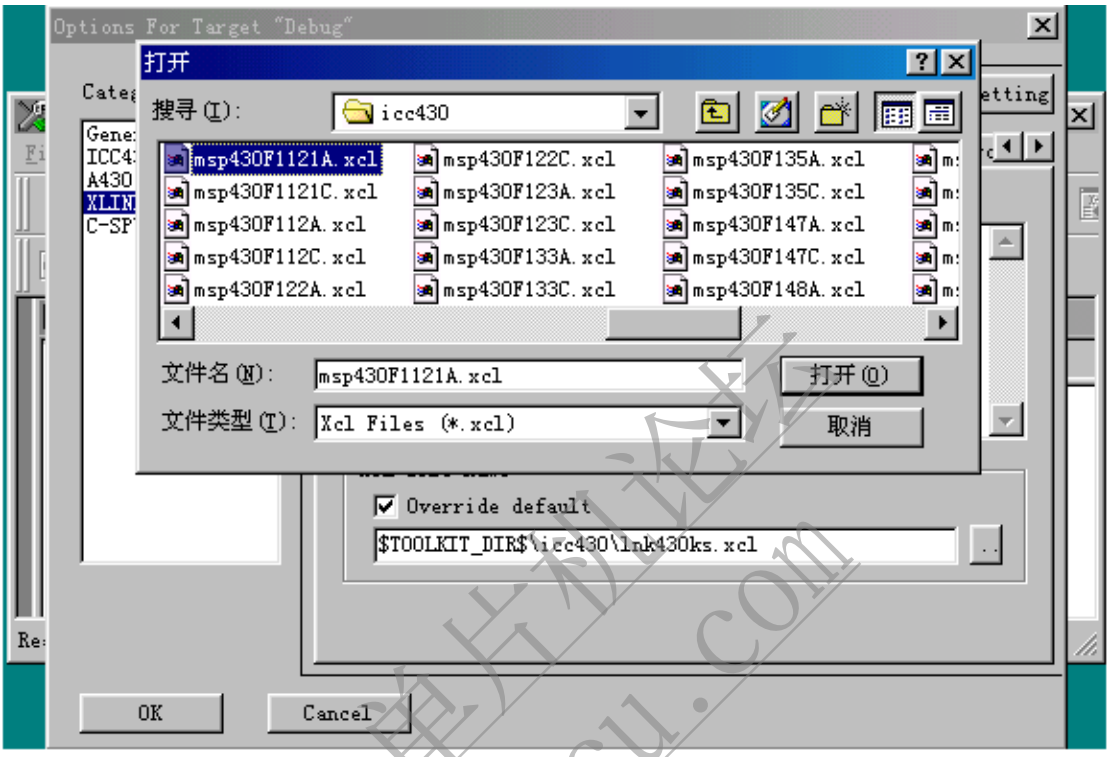

图 4.17　选择 msp430F1121A.xcl

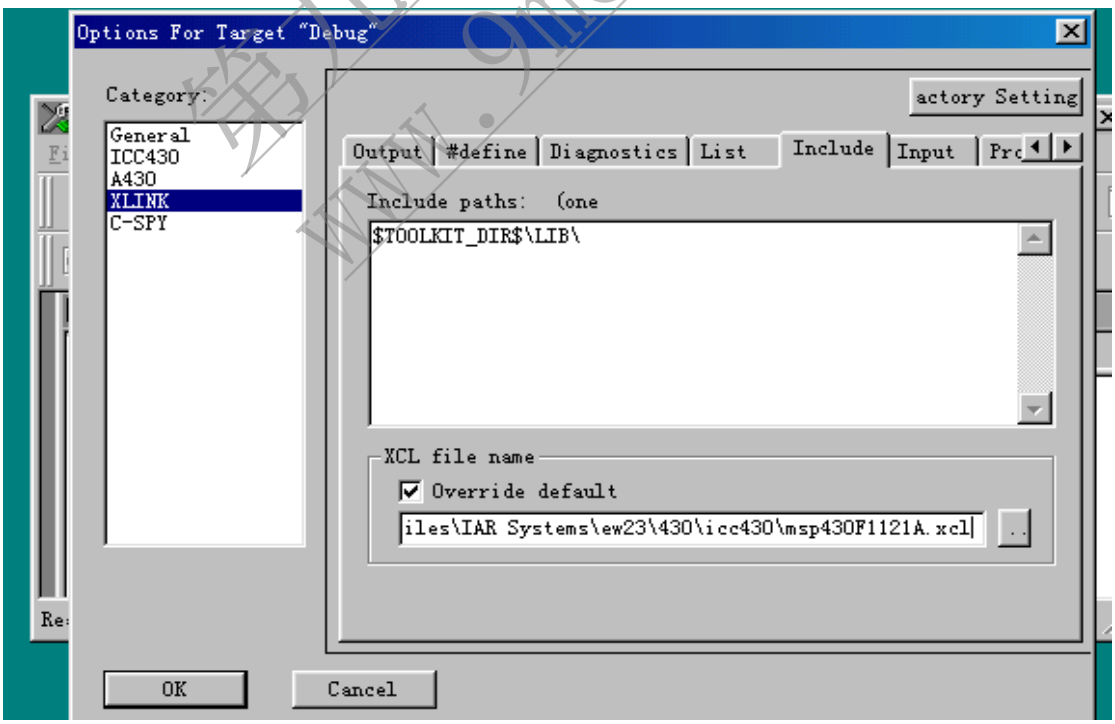

图 4.18　设置 XLINK 之后的情况

点击 Project、Options 进入设置后,点击 C-SPY 进入如图 4.19 所示 界面。在驱动选项中有 3 个选项,Simulator/Flash Emulation Tool/ROM-Monitor。它们分别是软件模拟 / Flash 型仿真工具 / ROM 监控方式。一般情况下,如果你使用的是 Flash 工具套件，比如德州仪器的 **MSP-FETP430P140、MSP-FETP430P110、MSP-FETP430P410 或笔者自己设计的系列 430Flash 工具套件**，你须选择 Flash Emulation Tool 选项作为驱动。如果使用软件模拟则选择 Simulator。

在晶片描述（Chip Description）一栏正确选择你所使用的芯片，比如这个例子我们使用的是 430F1121，那么就选择 MSP430C1121.DDF 文件，注意文件的路径。在进入到调试环境之后，芯片的情况就按这个文件进行描述，主要是器件的 0000H—01FFH 之间的功能寄存器的名称描述。这些设置好之后按 OK 键保存。

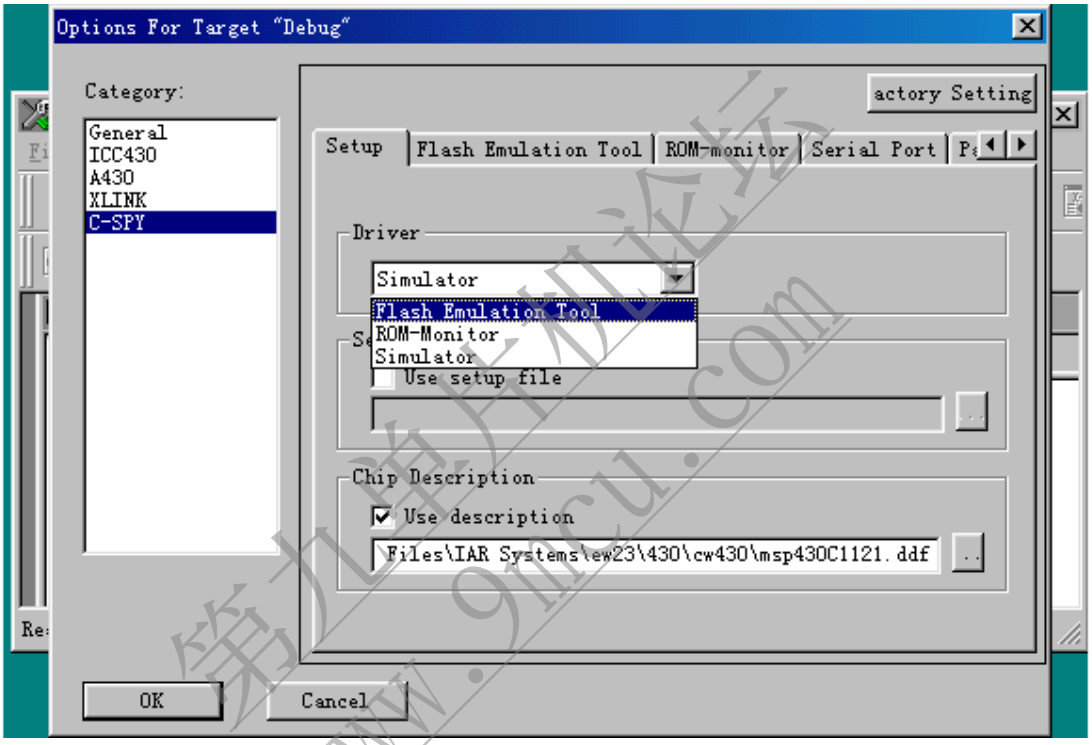

图 4.19 设置 C-SPY 参数

4．1．4 Embedded Workbench 综述

File(文件)菜单 提供打开项目与原文件、保存、打印以及退出此 Embedded Workbench 工作平台的命令。

Edit（编辑）菜单 提供在编辑窗口中编辑与搜索的命令。

View（视图）菜单 此菜单提供的命令允许用户改变显示在 Embedded Workbench 工作平台窗口中的信息。

Project（项目）菜单 提供将文件添加到项目、创建组，以及在当前项目上运行 IAR 工具的命令。

Tools（工具）菜单 此菜单是用户可配置的菜单，用户可将与嵌入式工作平台 Embedded Workbench 一起使用的工具添加到此菜单中。

Options（选项）菜单 此菜单允许用户定制嵌入式工作平台 Embedded Workbench，对各种环境参数进满足用户要求的配置。

Windows（窗口）菜单　此菜单上的命令允许用户管理嵌入式工作平台 Embedded Workbench 的窗口并改变它们的屏幕排列：用户可以打开需要的工作窗口，并随意在屏幕排列打开的窗口。

Help（帮助）菜单　提供此工作环境的帮助。

IAR 嵌入式工作平台 Embedded Workbench 快捷按钮如图 4.20 所示，左边为文件的建立、打开、保存、打印等操作与文件内的剪切、拷贝、粘贴等等操作，右边为查找、替换等与整个环境的界面控制操作等等以及快捷的帮助按钮。通过这些快捷按钮可以方便地进行操作。主要是文件的编辑与项目的组织。

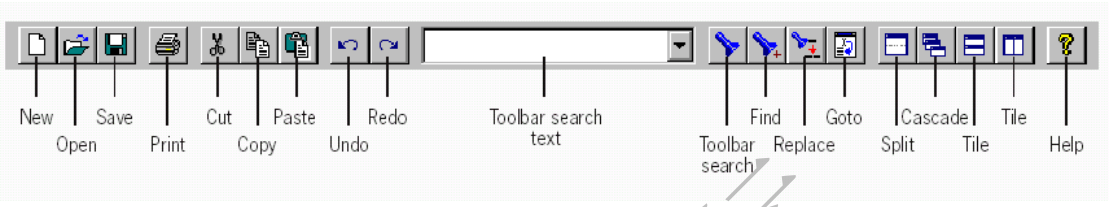

图 4.20　IAR Embedded Workbench 快捷按钮

4.2　C-SPY 使用指南

在 IAR 嵌入式工作平台 Embedded Workbench 中可以方便地进入 C-SPY 调试环境。但这必须在你设计的程序通过了 Make(编译、汇编,连接)，生成了目标代码之后。有三种方式可以进入 C-SPY 调试环境，分别是：在 Embedded Workbench 中点击 Project、Debugger；或在快捷按钮中点击  按钮；或在 WINDOWS 环境下依次点击:开始、程序、IAR SYSTEMS、IAR Embedded Workbench For MSP430 Kickstart、IAR C-SPY debugger,之后进入如图 4.21 所示的集成环境，这里可以非常方便地调试你所设计的程序。在图 4.21 中,已经打开了若干窗口:源程序窗口、寄存器窗口、存储器窗口、观察窗口等等,该环境的最顶上为系统主菜单(下拉式)，紧接着为快捷菜(如图单图 4.22 与图 4.23)，最主要部分为各种打开的窗口。下面介绍 C-SPY 调试环境的主要功能。

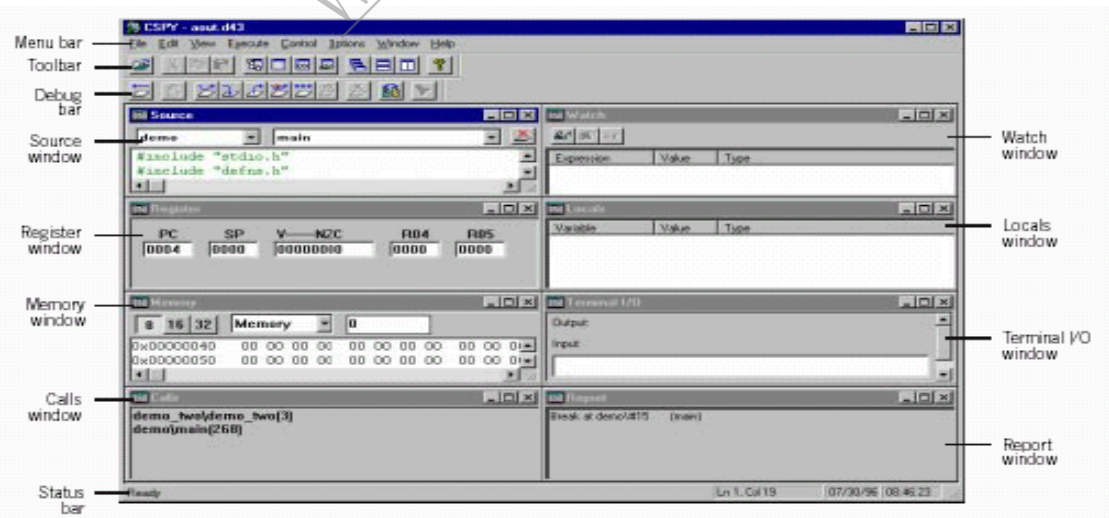

图 4.21　C-SPY 环境简介

图 4.22 与图 4.23 解释了各种按钮。下面我们来看看它们的执行情况。在 C-SPY 环境里,可以打开调试程序所需的若干窗口(如图 4.24 的 Window 菜单,快捷按钮如图 4.22):源文件窗口、寄存器窗口、观察窗口、存储器窗口、特殊功能寄存器窗口等等,这些窗口可以以多种排列方式在集成环境中显示。图 4.21 为常用的一种显示方式。

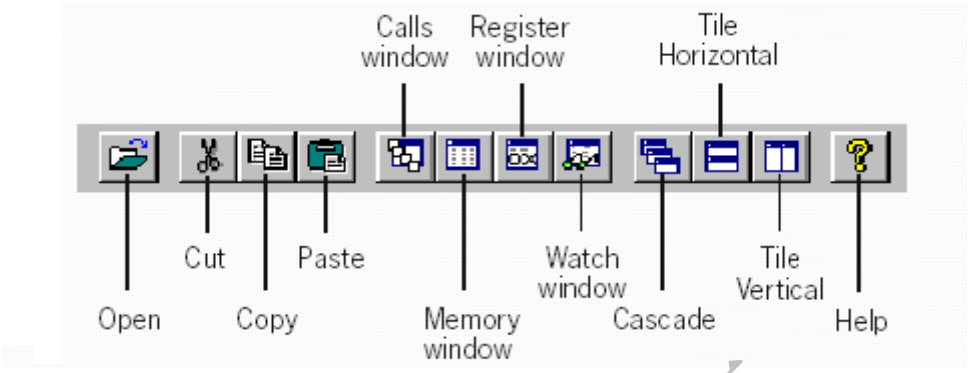

图 4.22 C-SPY 快捷按钮之一

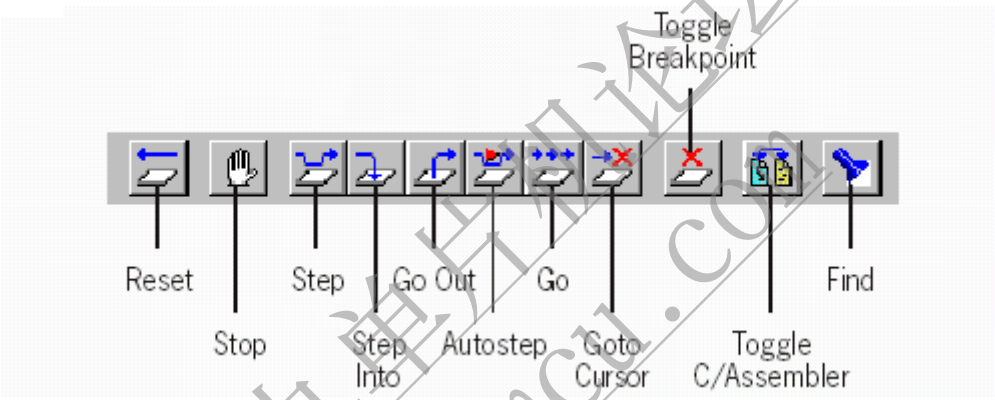

图 4.23 C-SPY 快捷按钮之二

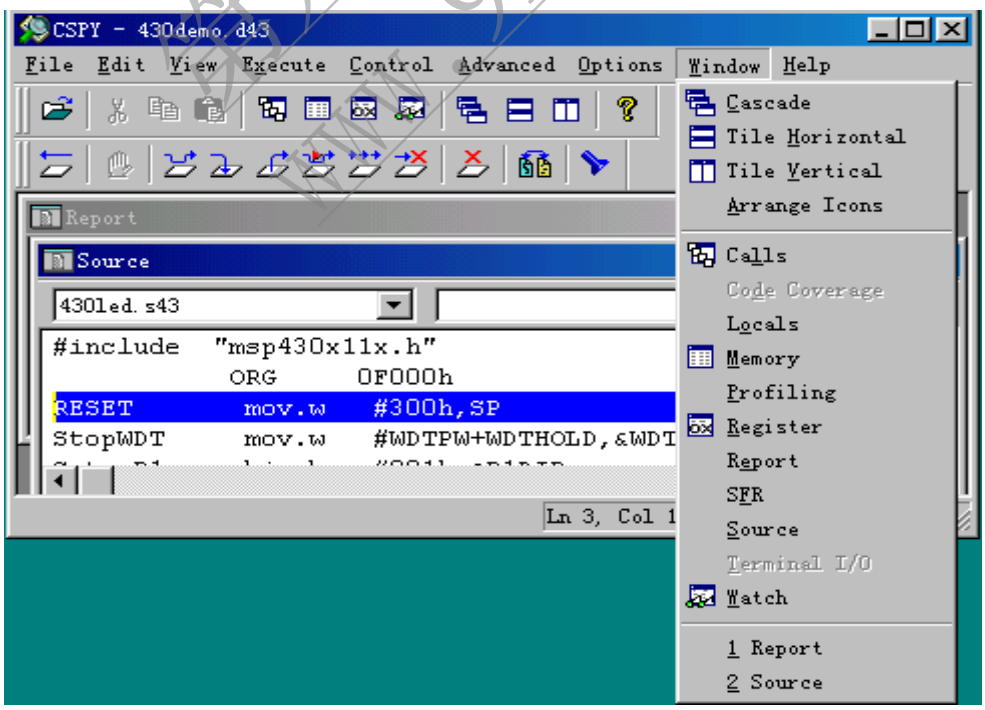

图 4.24 下拉式菜单举例:Window 窗口菜单

在这个调试环境中，经常用到的是与程序的执行、一些窗口有关的命令，图 4.23 为常用的调试工具，图 4.22 为常用的与窗口有关的命令 。下面来看看常用的窗口。

存储器窗口是调试程序常用的窗口，如图 4.25。此窗口的打开方法为：点击图 4.24 中的 Memory,或图 4.22 中的 Memoy Window 按钮。MSP430 的存储器为线性统一地址，小模式下地址范围 0000H-0FFFFH。在此窗口可以观察其存储器的全部内容，而且可有三种数据格式：8 位、16 位、32 位。

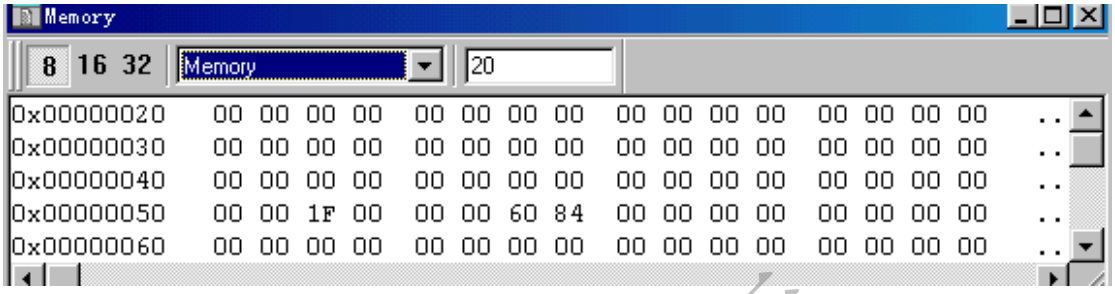

图 4.25　　存储器窗口

源程序窗口是调试程序必不可少的，一般情况会默认打开，但假设没有可以点击图 4.22 中的 Source 以打开，如图 4.26 所示。其中的兰色指示为即将执行的指令语句条。随着程序运行，此兰色指示会程序流程而改变。点击图 4.23 中的 Toggle C/Assemble 快捷按钮则可以观察到 4.27 所示的地址-代码-源程序窗口。此时的兰色指示条指示在具体的物理地址处，同时显示格式有所改变，先显示源程序，再另行显示地址-标号，地址-代码-指令。而且可以看出每一条指令都开始于偶地址处。

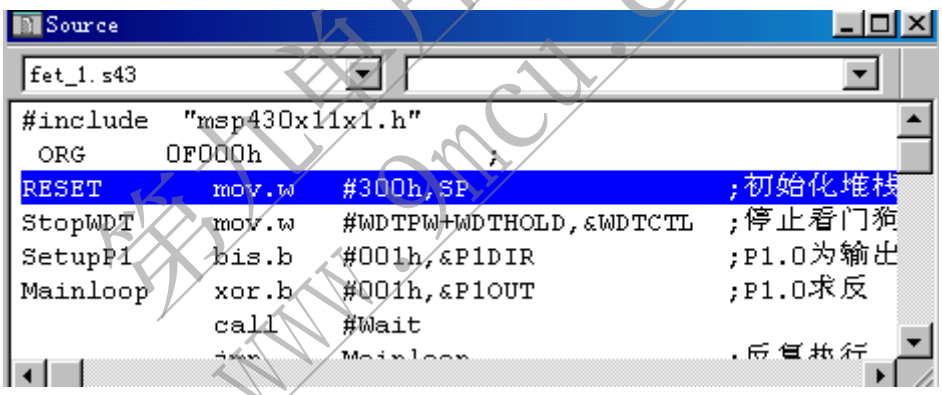

图 4.26　　源程序窗口

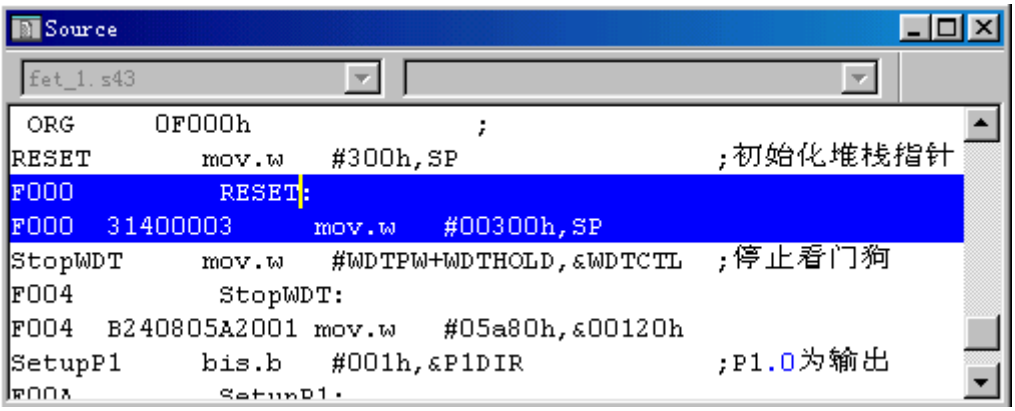

图 4.27　　源程序/代码窗口

寄存器窗口也是调试程序另一个常用的窗口，如图 4.28 所示。其打开方式为：点击图 4.24 中的 Register，或点击图 4.22 中的 Register Window。其中 R0-R2 分别是前三个 PC（程序计数器）、SP（堆栈指针）、SR（状态寄存器），R3 为常数产生器，模拟指令使用，这里用户看不到 ，R4-R15 为用户使用，可以查看与修改，CYCLES 为程序执行所用的机器周期数，通过它可以方便知道执行代码的时间。

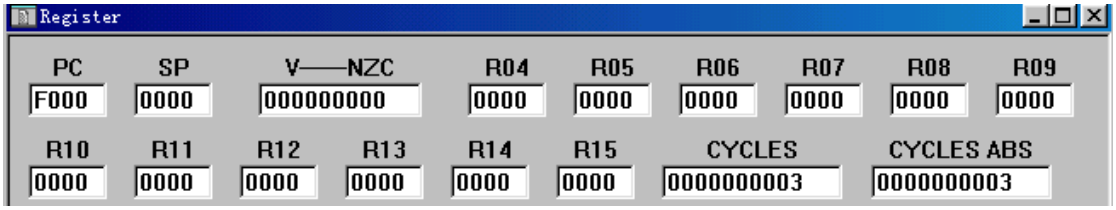

图 4.28 寄存器窗口

特殊功能寄存器窗口也是调试程序另一个常用的窗口，如图 4.29 所示。其打开方式为：点击图 4.24 中的 SFR。这个窗口有一些选项，图中选的是 Ports，所以打开的是与端口有关的一些特殊功能寄存器，同样可以打开其它一些相关的特殊功能寄存器。

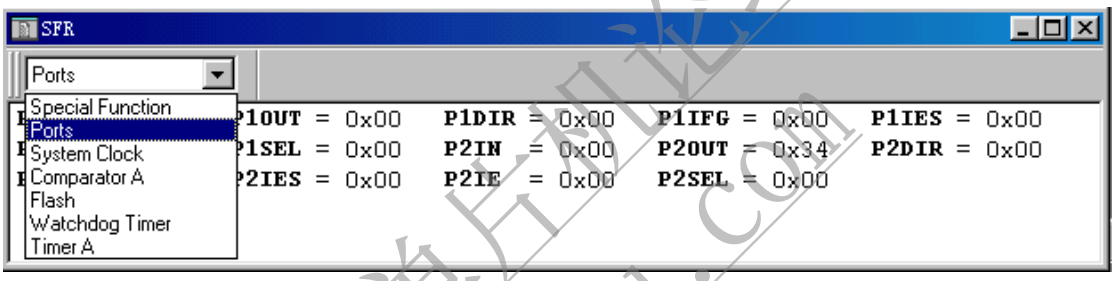

图 4.29 特殊功能寄存器窗口

点击图 4.24 中的 Watch 可以打开如图 4.30 所示的观察窗口，这里可查看用户想知道的变量的值。左边是变量名称，右边是变量的数值。已经有三个变量，要添加其他变量，可在下面的虚线框内按鼠标右键，再点击 Add,然后输入变量名称，或直接在源程序中于要观察的变量处按鼠标右键，再点 Quick Watch。观察窗口中的不想查看的变量名称也可方便地删除，在不想查看的变量按鼠标右键，再点击 Remove。

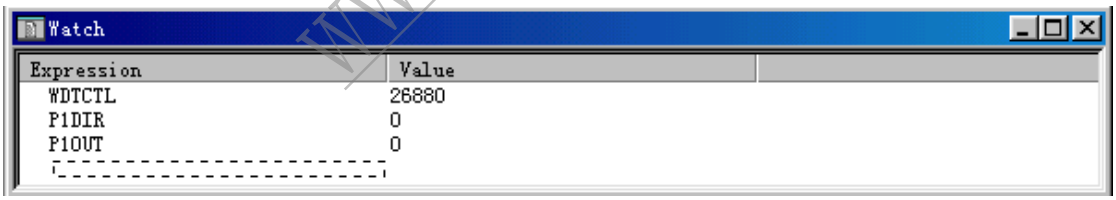

图 4.30 观察窗口

4．3 汇编程序调试举例

以前面在 Embedded Workbench 环境中写的程序为例说明怎样在 C-SPY 中调试。源程序如下：

```
ORG     0F000h                  ;
RESET       mov.w   #300h,SP                    ;初始化堆栈指针
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL      ;停止看门狗
SetupP1     bis.b   #001h,&P1DIR                ;P1.0 为输出
```

```
Mainloop    xor.b   #001h,&P1OUT          ;P1.0 求反
Wait        mov.w   #050000,R15           ;延时初值
L1          dec.w   R15                   ;
            jnz     L1                    ;
            jmp     Mainloop              ;反复执行
            ORG     0FFFEh                ;430 的第一条指令位置
            DW      RESET                 ;
            END
```

MSP430 的数据与程序存储器为统一线性编址，首先来看程序代码在存储器中的分布。程序的第一句

```
ORG     0F000h
```

表明程序代码应放在 0F000H 开始的空间，如图 4.31

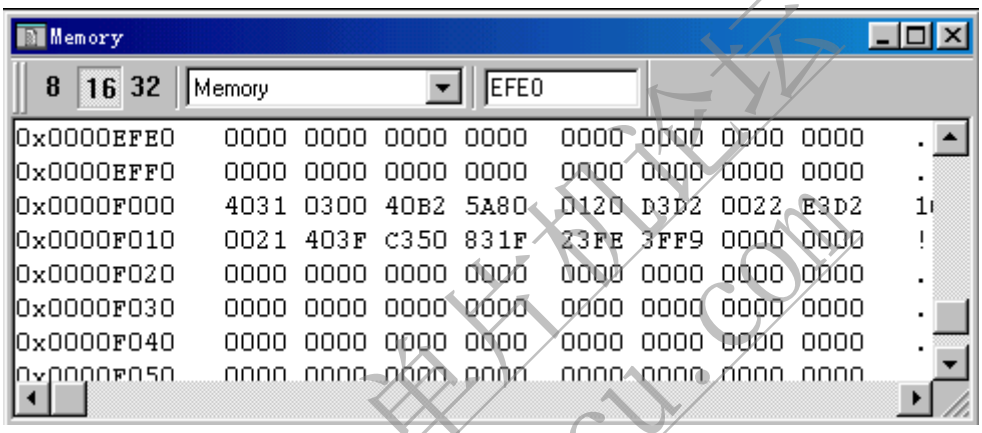

图 4.31　程序代码

程序的后面两句

```
ORG     0FFFEh
DW      RESET
```

指示程序复位之后应该执行的第一条指令在那里，RESET 是一个地址标号:0F000H,这个数据在地址 0FFFEH 中。如图 4.32。在这个例子里用的是 430F1121 芯片，它的程序存储器的地址为 0F000H-0FFFFH，其中 0FFE0H-0FFFFH 为中断向量，而地址 0FFFEH 处的中断向量为器件复位之后执行的第一条语句的物理地址，将它放在 0F000H 处，故在 0FFFEH 处的数据为 0F000H。

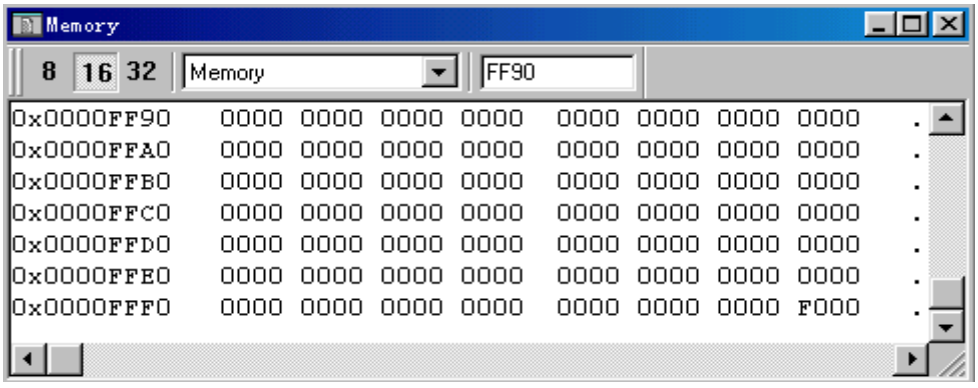

图 4.32　MSP430F1121 复位中断向量的内容 0F000H

在这里有几种方式运行我们的程序：单步，连续，断点，运行到光标等等，下面分别阐述。先单步执行第一句

```
mov.w   #300h,SP
```

单步运行程序一定要使之处于非实时状态，点击 CONTROL，再取消 REALTIME，这样就能单步运行程序了。单步执行可点击  按钮，或依次点击 EXECUTE，STEP 或 F2，执行之后 SP 的内容变为 300H，为了查看可以打开寄存器窗口：点击  或依次点击 WINDOW，REGISTER，我们可以看到 SP 的内容发生了变化，而且为红色，表示刚刚改变。而源程序指示到下一句，表示将要执行的语句，如图 4.33，也在 PC 的内容上反应出来，指向了 0F004H。

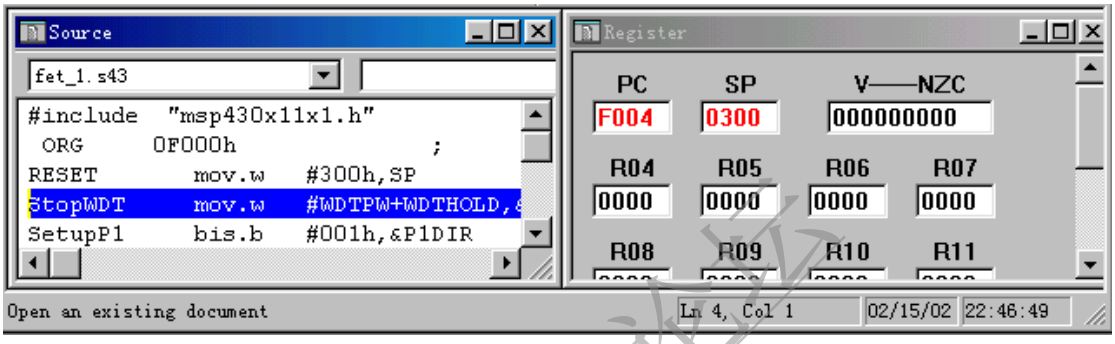

图 4.33 第一句执行后的情况

继续使用单步执行第二句，第二句为对看门狗的寄存器操作，可以依次点击 WINDOW，SFR 打开寄存器窗口，然后找到看门狗定时器，这时就可以观察 WDTCTL 的内容，执行之后如图 4.34，同时 PC 指向下一句。

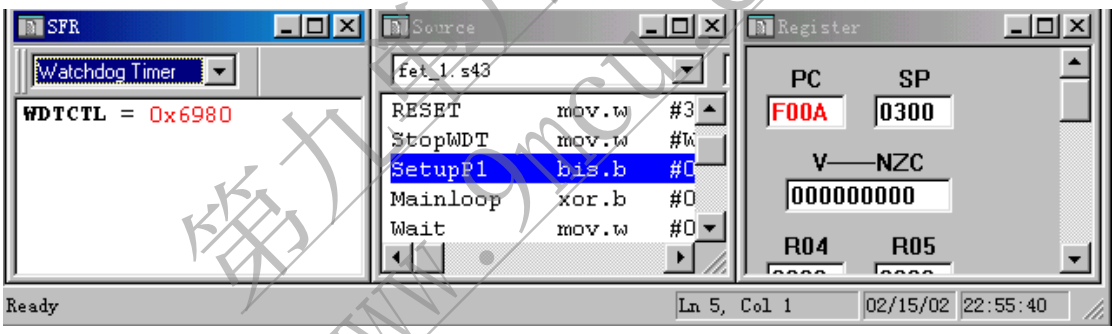

图 4.34 第二句执行后的情况

继续使用单步执行第三，四两句，这二句为对端口操作，同样依次点击 WINDOW，SFR 打开寄存器窗口，然后找到 PORTS，这时就可以观察端口一、端口二的全部内容，执行之后如图 4.35，红色表示刚刚改变，同时 PC 指向下一句。下面是循环结构，意于 CPU 等待一段时间，再将 P1.1 反向输出，这样就可以看到 P1.1 上连接的发光二极管闪烁的效果。对于循环结构再用单步执行就不妥当了，改用其他的方式来运行我们的程序：设置断点或运行到光标处都可以。

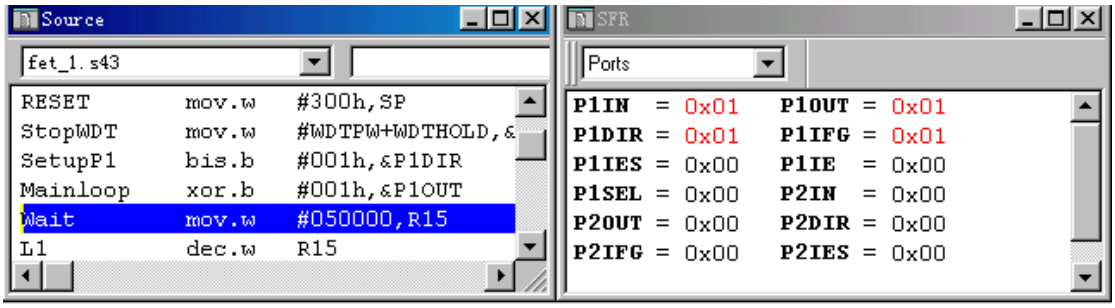

图 4.35　第四句执行后的情况

使用断点方式。首先得设置断点，将光标条移到要设置断点的语句处

……

```
Mainloop    xor.b    #001h,&P1OUT          ;P1.0 求反
Wait        mov.w    #050000,R15           ;延时初值
L1          dec.w    R15                   ;
            jnz      L1                    ;
            jmp      Mainloop
```

……

在“JMP　MAINLOOP”处设置断点，在源程序中将光标条移到这一句。然后按 F5 或依次点击 CONTROL，TOGGLE BREAKPOINT 或点击  按钮。设置断点的语句条就会用醒目的红色显示，如图 4.36 所示。

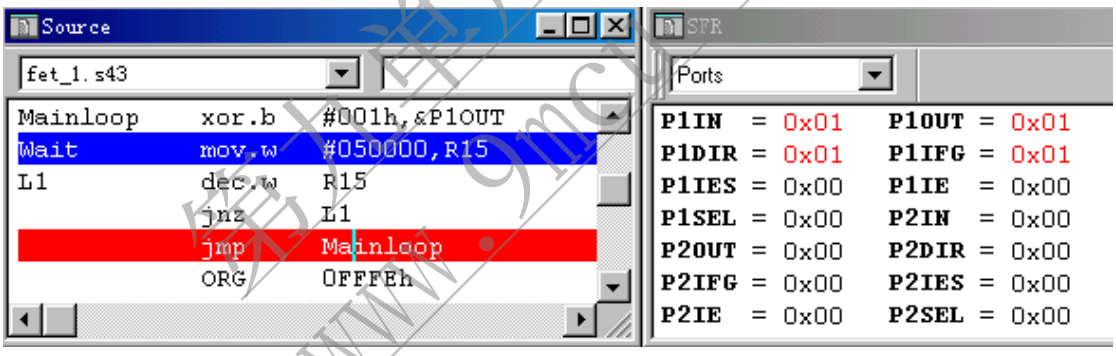

图 4.36　设置断点

再使用连续运行方式，程序运行到断点处就停下来。50000 条 R15 减一判断语句就执行完了。连续运行方式怎么操作，按 F4 或  按钮或依次点击 EXECUTE，GO。

这是可再使用连续运行方式反复运行，就可以看到 P1OUT 端口的数据 00H，01H 交替变化。P1.1 的变化如图 4.36 与图 4.37。也就是如果在 P1.1 上接一发光二极管，则它闪烁。

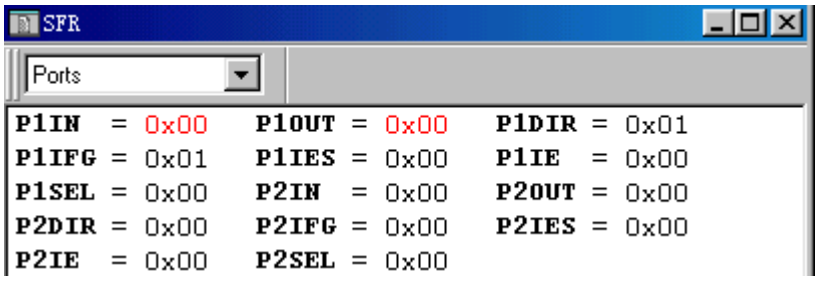

图 4.37　P1OUT 为 0

执行到光标处又是怎么一回事呢，很简单，这时先要取消断点：与设置断点一样的方法，F5，取消断点之后，将光标移到想要运行到的语句条，再点击  或依次点击 EXECUTE，GO TO CURSOR。

在程序中的延时用的是软件的方法：执行若干条需要一定时间的语句。这里用的是循环，R15 设置一初始值，再减一判断是否减完。在执行这些指令的过程中的一些现场情况也可以看到。为了比较，先记录下复位时的情况，如图 4.38。在运行到

```
Wait        mov.w    #050000,R15
```

这一句执行之后再看现场的情况，如图 4.39。R15 已经是程序所赋予的数据 50000，十六进制为 0C350H，这一句执行的是数据传送操作，状态标志不受影响 Z=1 是以前的结

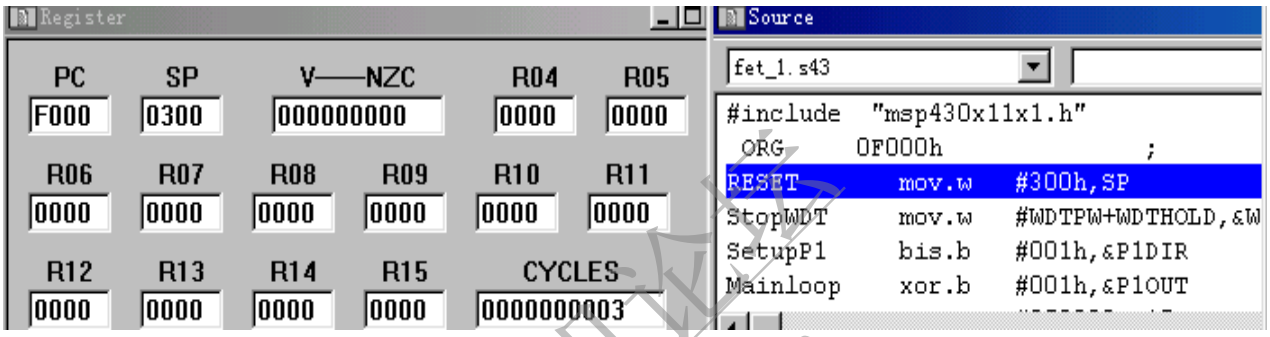

图 4.38 复位时的情况

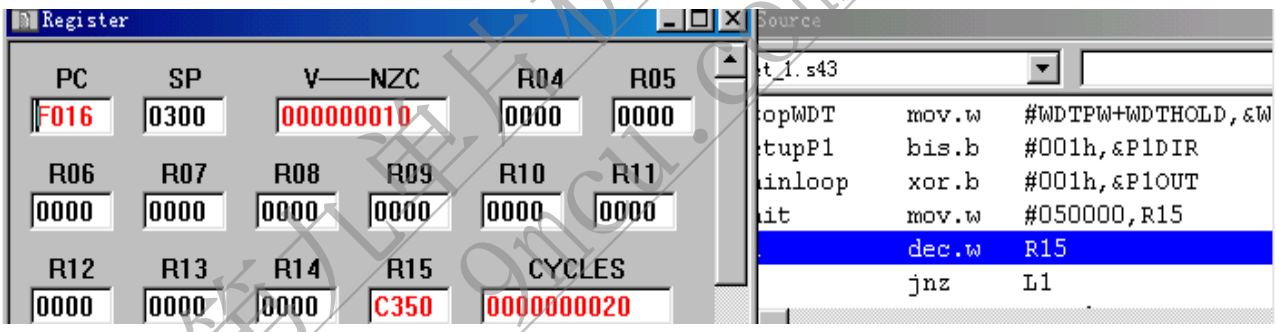

图 4.39 执行 WAIT 语句行之后

果。紧接着是一句减一操作，看看如何对标志产生影响。执行之后如图 4.40。可以看到状态标志有三位发生了变化：N，Z，C，程序将根据 Z 的取值情况来决定程序的取向，0C350H 减一后为 0C34FH，不为 0，因而 Z=0，执行下一句之后将会跳转，继续减一判断。直到 R15 为 0 时，Z=1，程序再往下执行。如图 4.41 所示。此时 R15 中的数据已经减完了，为 0，则状态寄存器中的标志位 Z=1， JMP L1 语句执行的结果为不跳转到 L1 处，而将执行其下一句。21 个机器周期是循环之前花掉的，实际上循环体用掉了 150000 个 机器周期。150000 T 就是发光二极管闪烁的延时时间(亮与熄之间的时间间隔)。

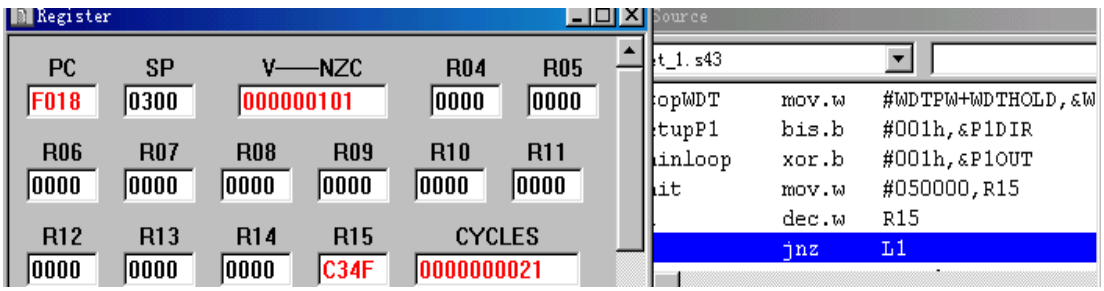

图 4.40 执行减一操作之后

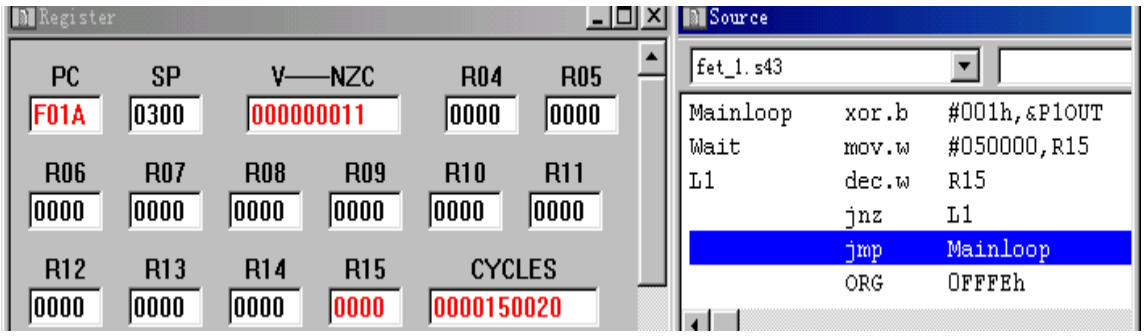

图 4.41　　Z=1 时跳出循环

4．4　C 程序调试举例

C 语言程序的编辑调试与汇编的差不多。这里还是实现与前面汇编一样的功能：让接在 P1.0 的发光二极管闪烁。先编辑源程序，点击 File、New 进入图 4.7 界面，选择 Source/Text 选项，按确定进入源程序编辑界面，源程序如下：

```
  #include  <msp430x11x1.h>
// 函数原型声明
void delay(void);    // 软件延时
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; //停止看门狗
  P1DIR = BIT0;      // P1.0 输出
  while(1)           // 主循环
  {
    P1OUT |= BIT0;  // P1.0=1
    delay();
    P1OUT &= ~BIT0; // P1.0=0
    delay();
  }
}
//延时.
void delay(void)
{
  unsigned int i;
  for (i = 65000; i > 0; i--);
}
```

其中 BIT0 在 msp430x11x1.h 文件中定义：

```
#define BIT0                  0x0001
#define BIT1                  0x0002
#define BIT2                  0x0004
#define BIT3                  0x0008
#define BIT4                  0x0010
```

```
#define BIT5                0x0020
#define BIT6                0x0040
#define BIT7                0x0080
#define BIT8                0x0100
#define BIT9                0x0200
#define BITA                0x0400
#define BITB                0x0800
#define BITC                0x1000
#define BITD                0x2000
#define BITE                0x4000
#define BITF                0x8000
```

源程序编辑好之后，再将它加入项目、组。编译、连接，连接未通过，提示如图 4.42 所示，这是因为连接命令的参数设置不对，设置成如图 4.43 所示环境参数即可。这时编译、连接全通过。C-SPY 中用的器件设置与前述一样。点击  按钮之后进入调试环境，运用单步、断点、连续等等手段调试，查看执行情况也与汇编一样，这里不多说。

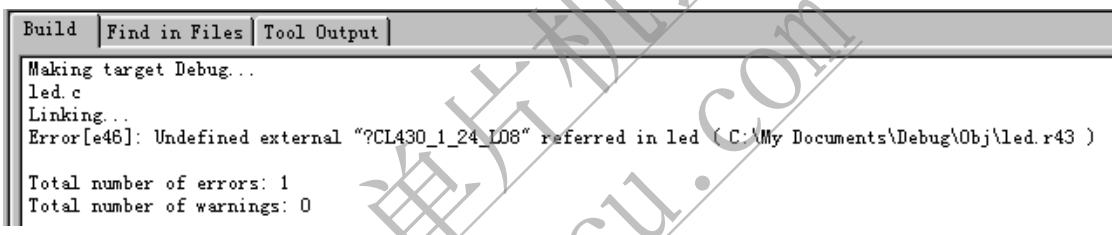

图 4.42 连接后的出错信息

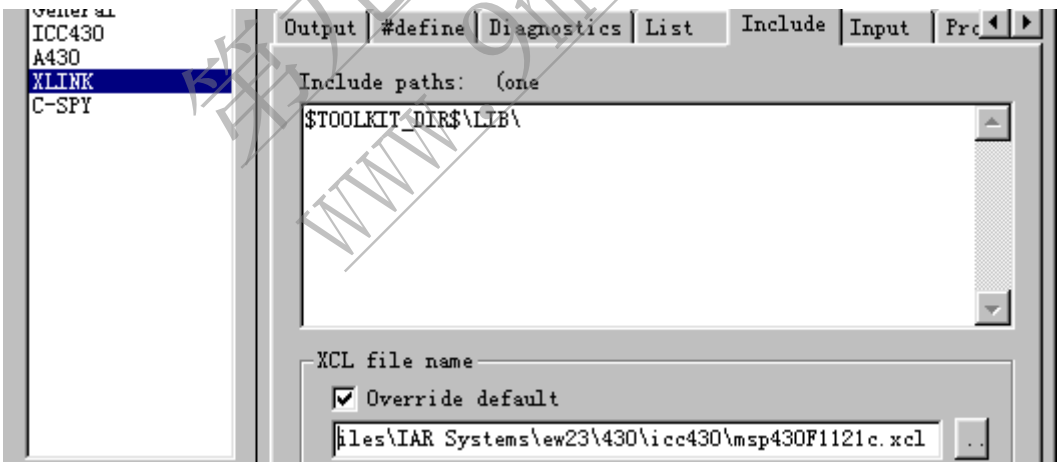

图 4.43 连接命令的参数设置

对于 C 语言的调试，还提供了一个非常方便的手段：除了使用 WATCH（观察窗口）外，LOCALS（局部变量观察窗口）的使用能给你的调试带来非常的方便。下面举例说明，使用的例程还是实现在 P1.0 上的灯闪烁。这里为了体现出该调试窗口的方便，特地增加了一些变量：

在主程序中，定义了两个变量：freq 控制灯闪烁快慢,flash 记录闪烁次数。Freq 参数将传送到延时程序，延时程序中，该参数决定延时次数，作为该程序执行时间的控制参数。Flash

参数用来记录闪烁次数，灯的状态每改变一次，该变量加一。程序如下：

```
#include   <msp430x11x1.h>
void   delay(unsigned int x)
    {
      unsigned int i;
      i = x        ;                  // 延时
      do (i--);
      while (i != 0);
      }

void main(void)
{ unsigned int freq,flash ; // 定义全局变量：
                          //freq 控制灯闪烁快慢,flash 记录闪烁次数
  WDTCTL = WDTPW + WDTHOLD; // 停止看门狗
  P1DIR |= 0x01;                  // P1。0 定义为输出
  freq=50000;                     //闪烁频率由软件延时时间决定
  flash=0;                        //初始化闪烁次数，开始时没有闪烁
  for (;;)
  {
    delay(freq);                  // 软件延时
    P1OUT ^= 0x01;                // 目标 P1.0 反向输出
    flash++;
  }
}
```

在调试时打开 LOLALS 窗口，最初试时总是主程序中用到的变量，如图 4.44 所示。要注意：窗口中的变量值是随机的，从程序可看出，随后会给它们赋予需要的值。

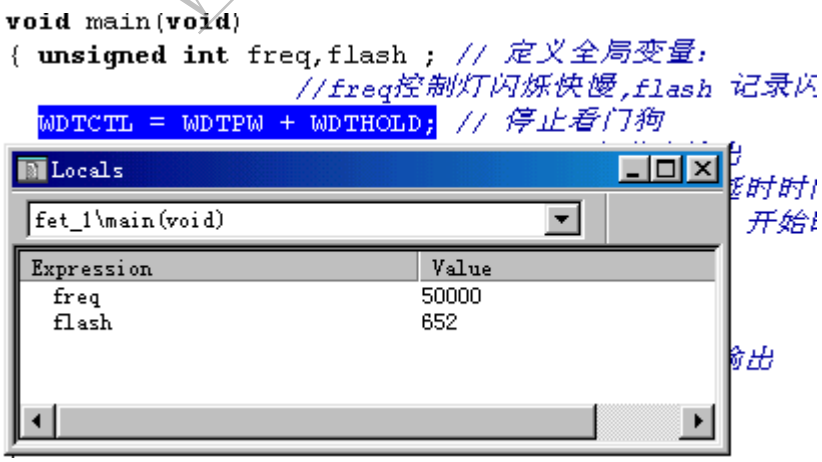

图 4.44　复位时的 LOLALS 窗口

而当程序执行到子程序时，该窗口的内容会发生改变，变为正在执行的子程序中的变量

以及其值。如图 4.45 所示。

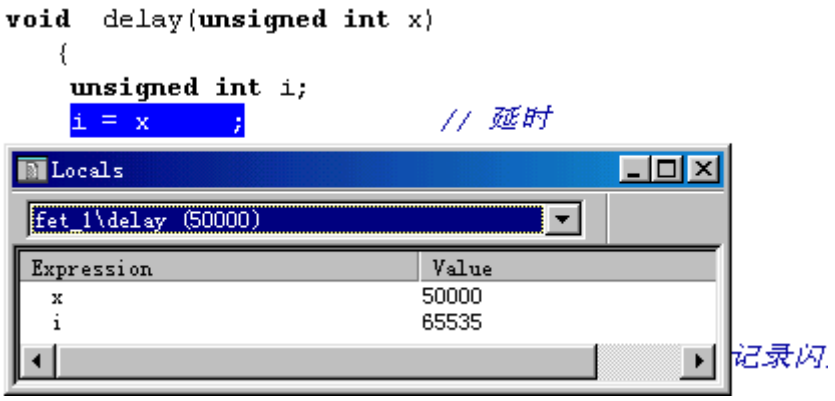

图 4.45 执行到 delay 子程序时的 LOLALS 窗口

由此，C 语言程序的调试使用 LOLALS 窗口是相当方便的。