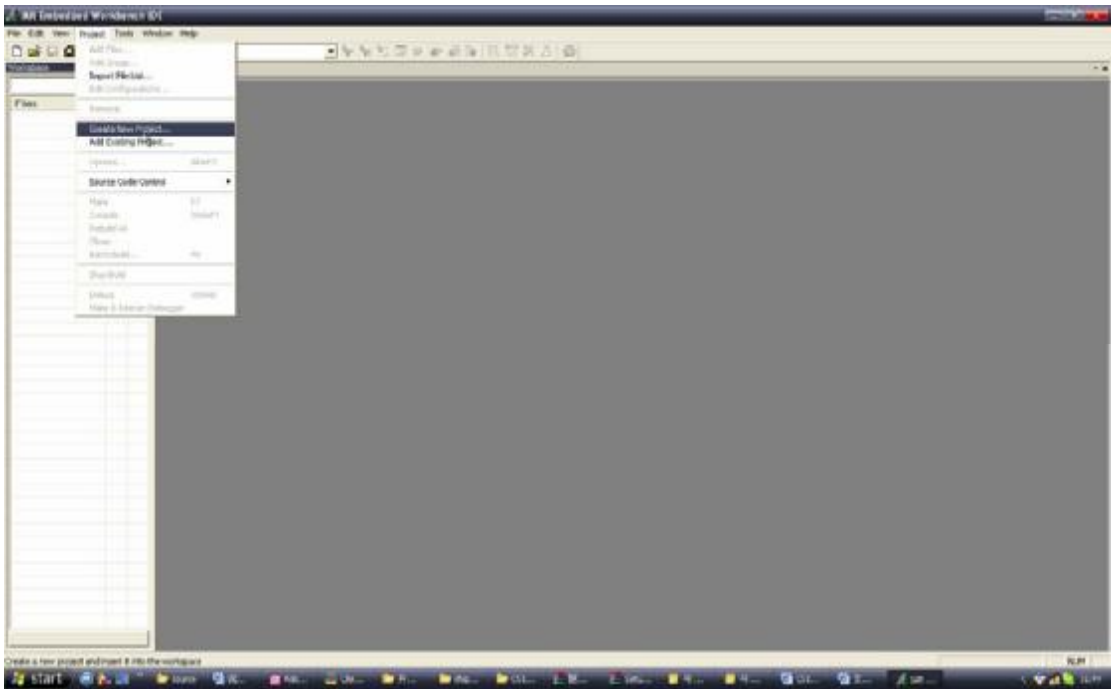
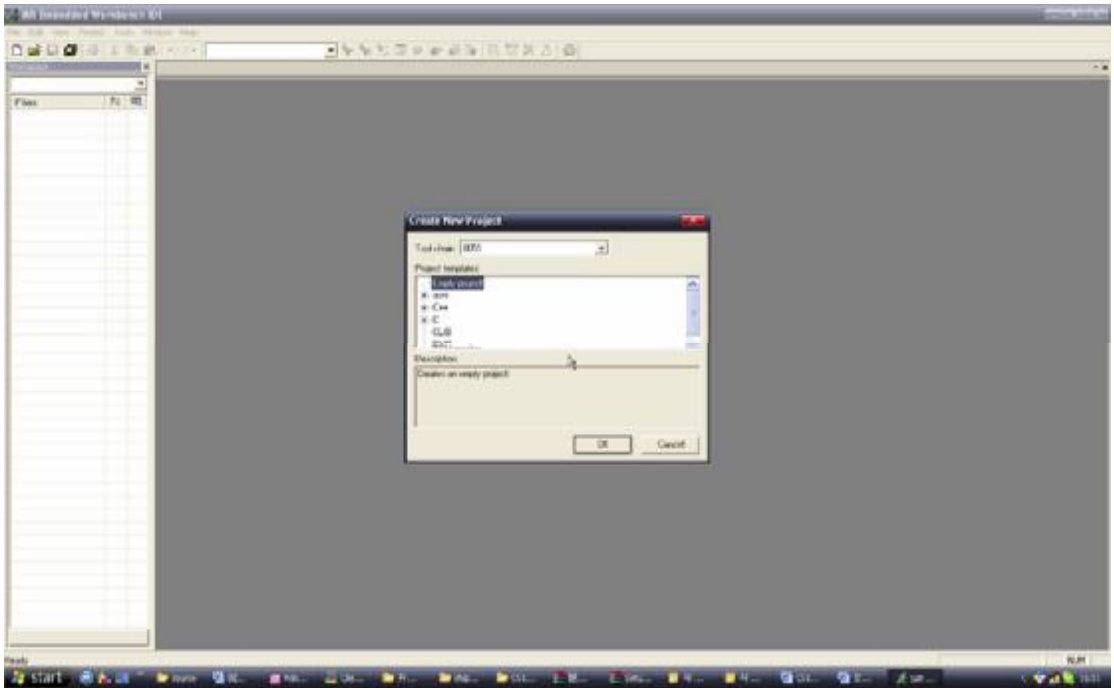
2.5 开发软件IAR使用方法

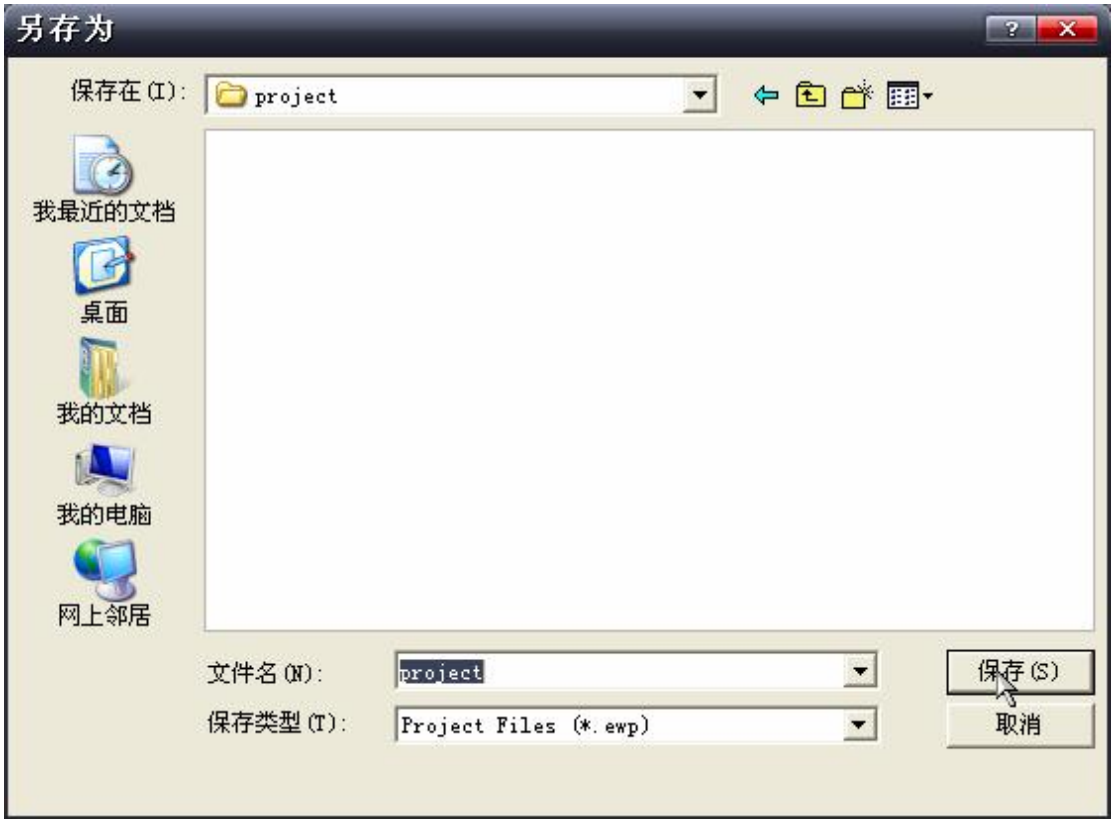
2.5.1 新建一个工程



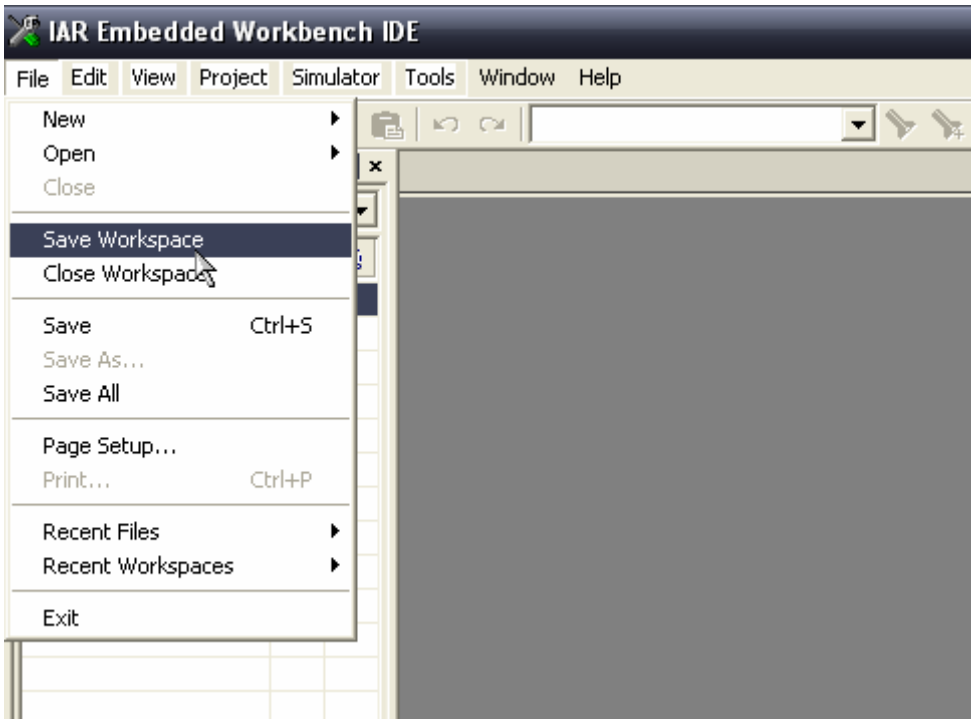
新建一个工程



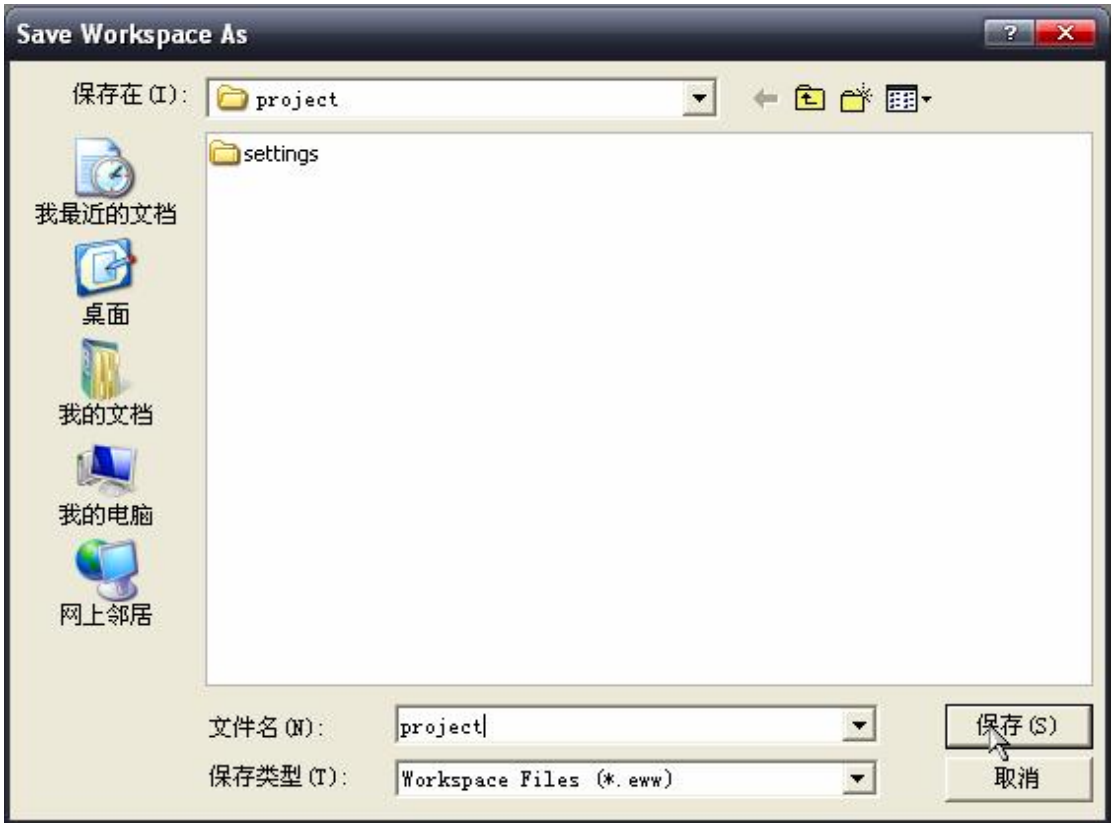
选择 Empty project 默认配置，单击 0K 弹出保存对话框



这个时候我们在桌面上建立一个名为 project 的文件夹，输入项目的文件名，并将项目也取名为“project”将此文件保存在 project 文件夹中，会产生一个 ewp 后缀的文件。

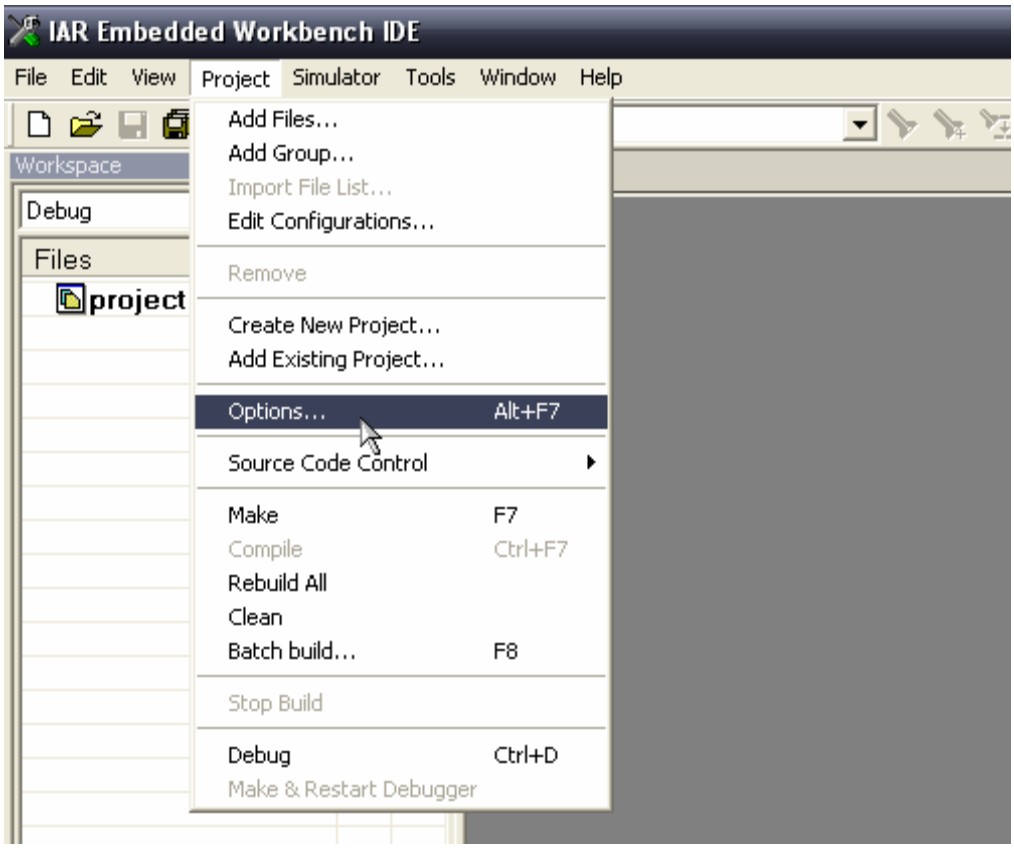


保存工程，弹出保存工程对话框

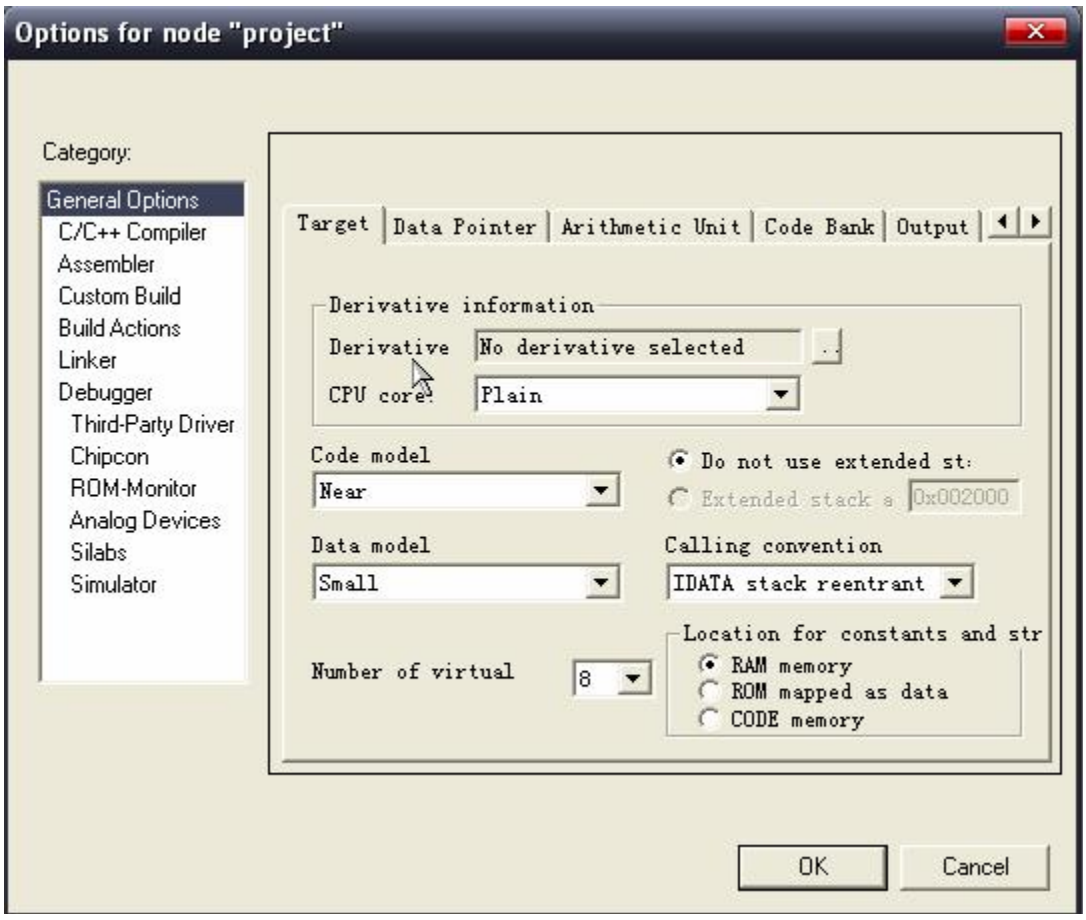


输入工程文件名，单击保存退出，系统将产生一个eww为后缀的文件

这样，我们就建立了IAR的一个工程文件，接下来，我们对这个工程加入一些特有的配置。



打开工程选项



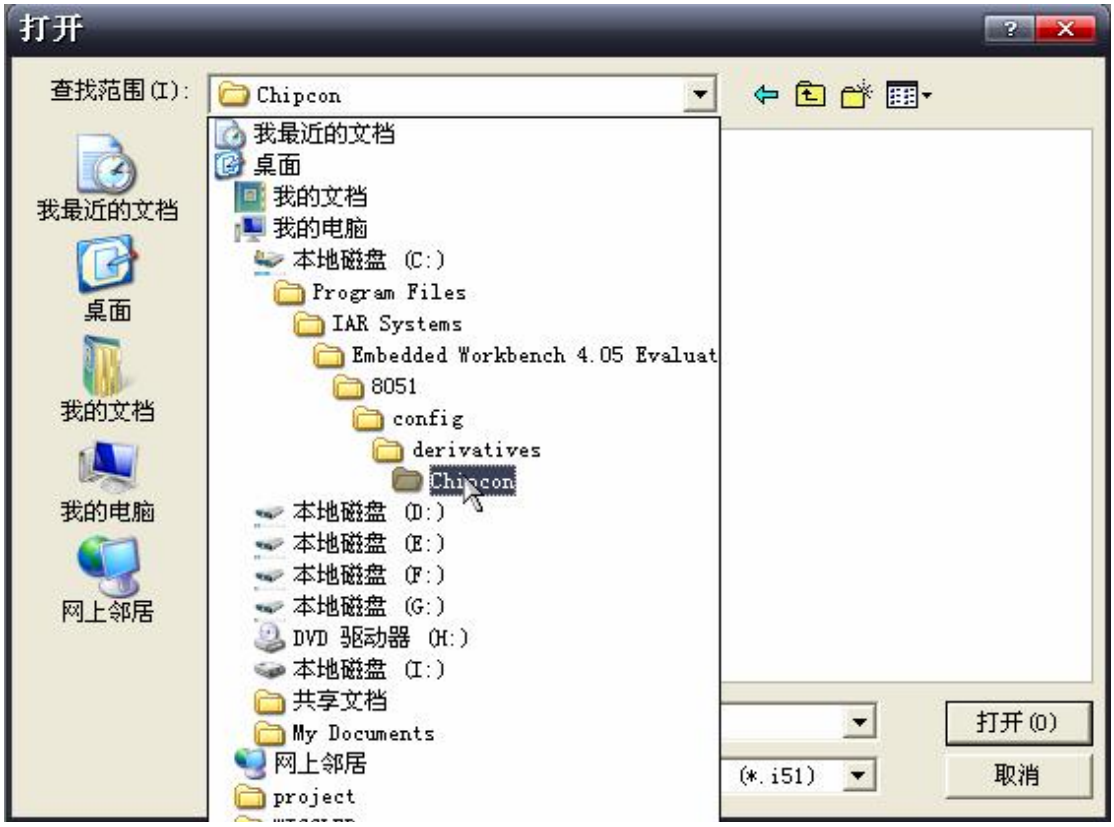
工程选项页面

工程选项页面中需要设置很多必要的参数，下面针对 CC2430 我们一起来配置这些参数.

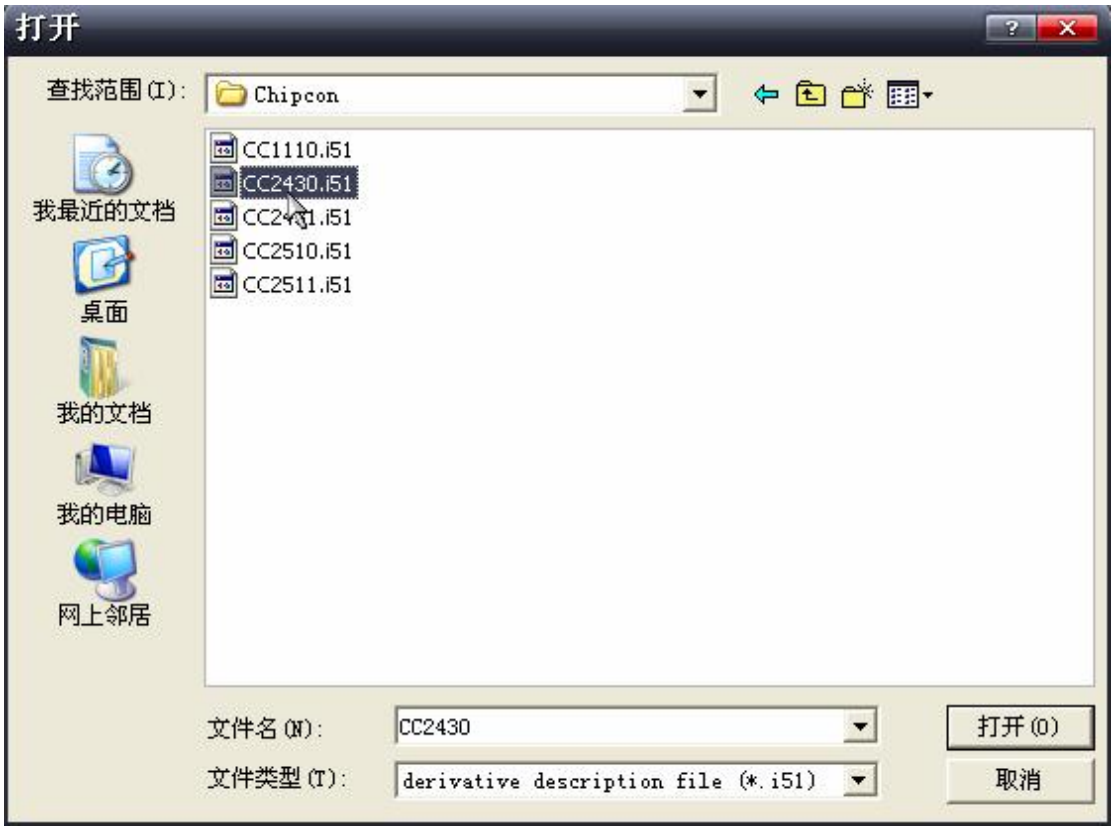
2.5.2 参数设置

1. General Options设置

在 General Options->Target 选项中 Derivative 选择为 CC2430，如图所示。

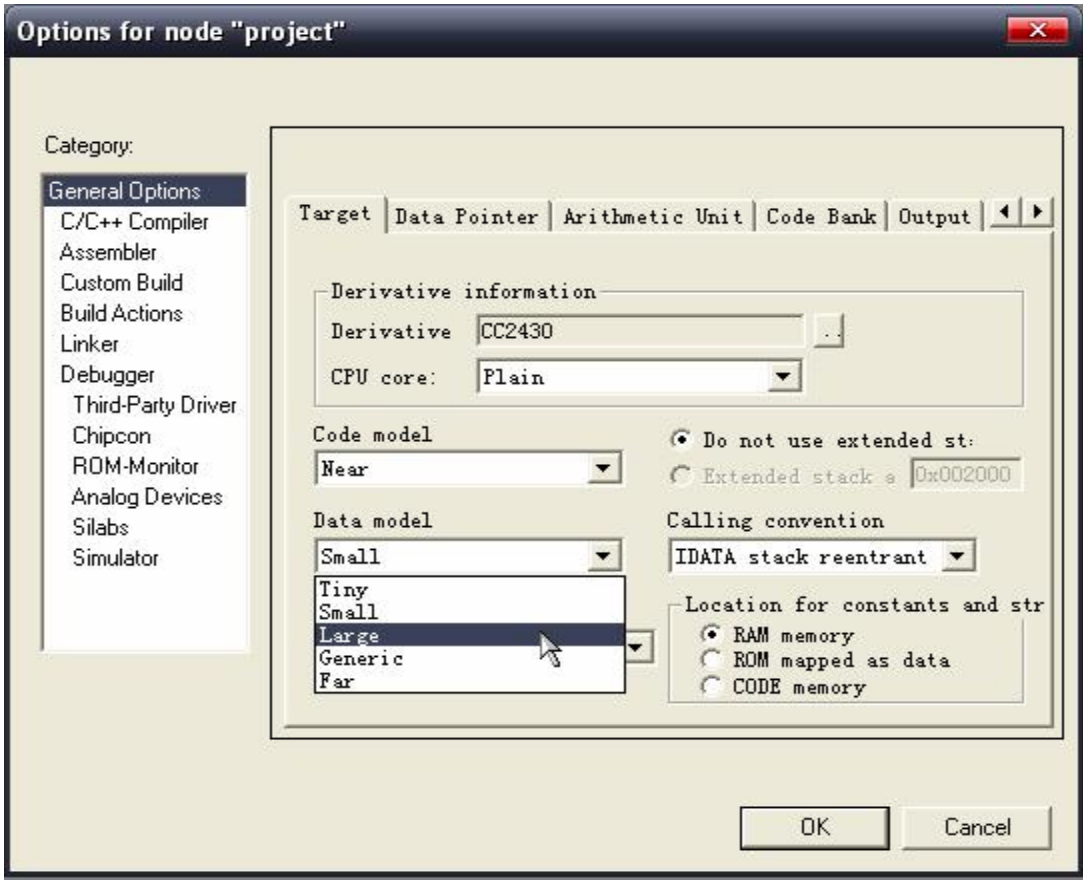


找到 Chipcon 文件夹

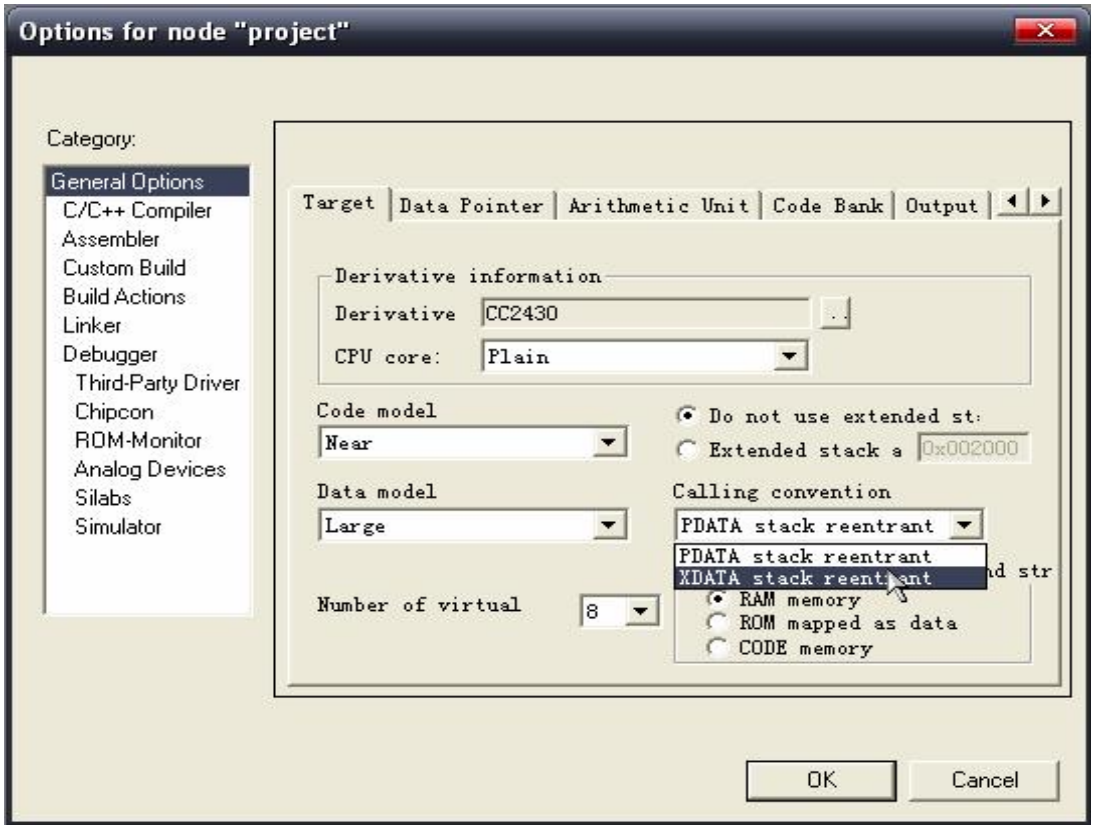


选择需要的芯片

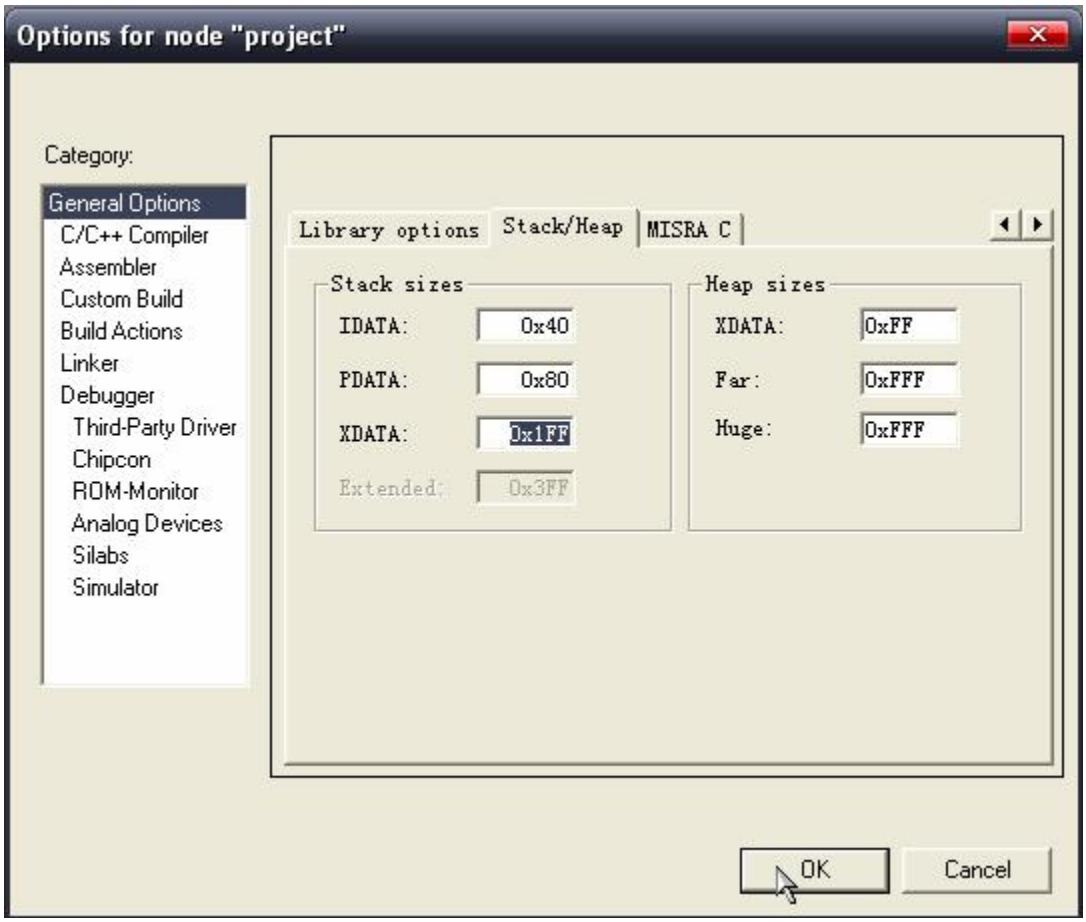
在 General Options->Target 选项中 Data model 选择为 Large，如图所示。



在 General Options->Target 选项中 Calling cinvention 选择为 XDATA，如图所示。



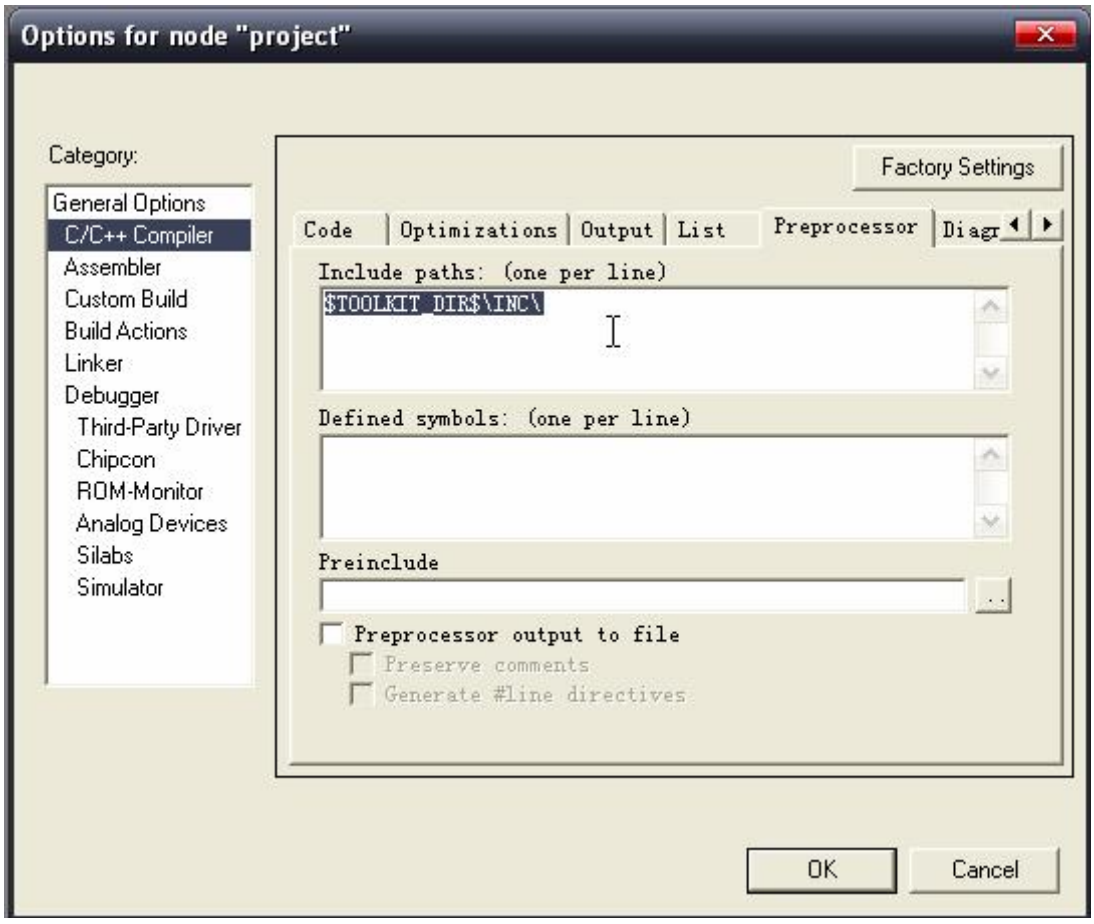
在 General Options->Target 选项中 Stack/heap 中的堆栈大小做适当修改，如图所示。



修改堆栈

2.C/C++ Compiler设置

在C/C++ Compile->Preprocessor选项中有两个很重要的选项，它们分别是Include paths和Defined symbols。Include paths表示在工程中包含文件的路径，Defined symbols表示在工程中的宏定义。



1) 在定义包含文件路径的文本框中，定义包含文件的路径有两种很重要的语法，一是**$TOOLKIT_DIR$**，这个语法表示包含文件的路径在**IAR**安装路径的**8051**文件夹下，也就是说如果**IAR**安装在**C**盘中，哪么它就表示**C:\Program Files\IAR Systems\Embedded Workbench 4.05 Evaluation version\8051**这个路径。二是**$PROJ_DIR$**，这个语法表示包含文件的路径在工程文件中，也就是和**eww**文件和**ewp**文件相同的目录。我们刚此建立的**project**项目中，如果使用了这个语言，哪么就表示现在这个文件指向了**C:\Documents and Settings\Administrator**桌面**\project**这个文件夹。

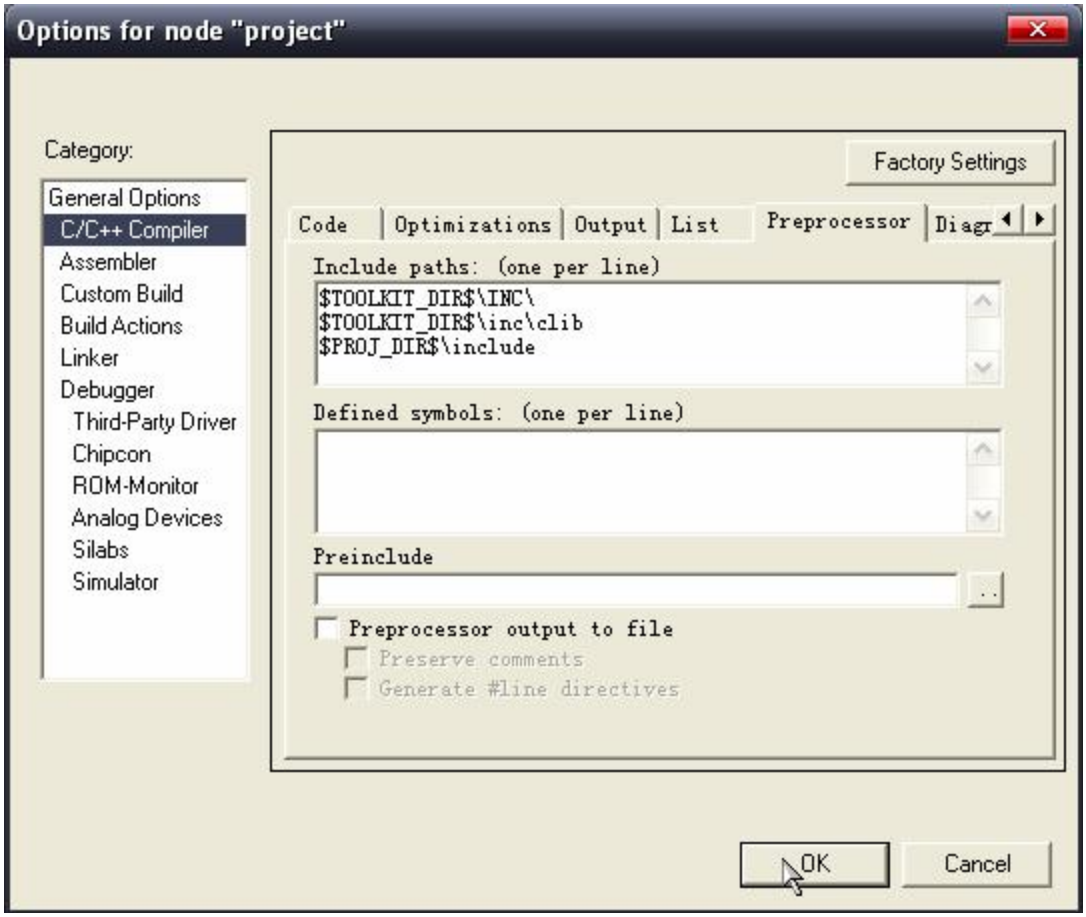
和这两个语言配合使用的还有两个很重要的符号，这就是“**\..**”和“\文件夹名”。
\..：表示返回上一级文件夹
\文件夹名：表示进入名为“文件夹名”的文件夹。
我们来具体看两个例子。

$TOOLKIT_DIR$\inc：这句话的意思是包含文件指向**C:\Program Files\IAR Systems\Embedded Workbench 4.05 Evaluation version\8051\inc**。

$PROJ_DIR$\..\Source：这句话的意思是包含文件指向工程目录的上一级目录中的**Source**文件夹中。例如：假设我们的工程放在**D:\project\IAR**中，哪么**$PROJ_DIR$\..**就将路径指向了**D:\project**中，再执行**\Source**，就表示将路径指向了**D:\project\Source**中。

继续回到我们的工程，下面我们通过上面的方法设定一些必要的路径。如下图所示，在图中的，有一个包含在工程中的**include**文件夹，这个文件夹需要自己在工程文件中创建，这里面放置的是这个工程的**h**文件，我们先将它设置在这里，**inc**中存放了**CC2430**的**h**文件**clib**

中有很多常用的**h**文件。

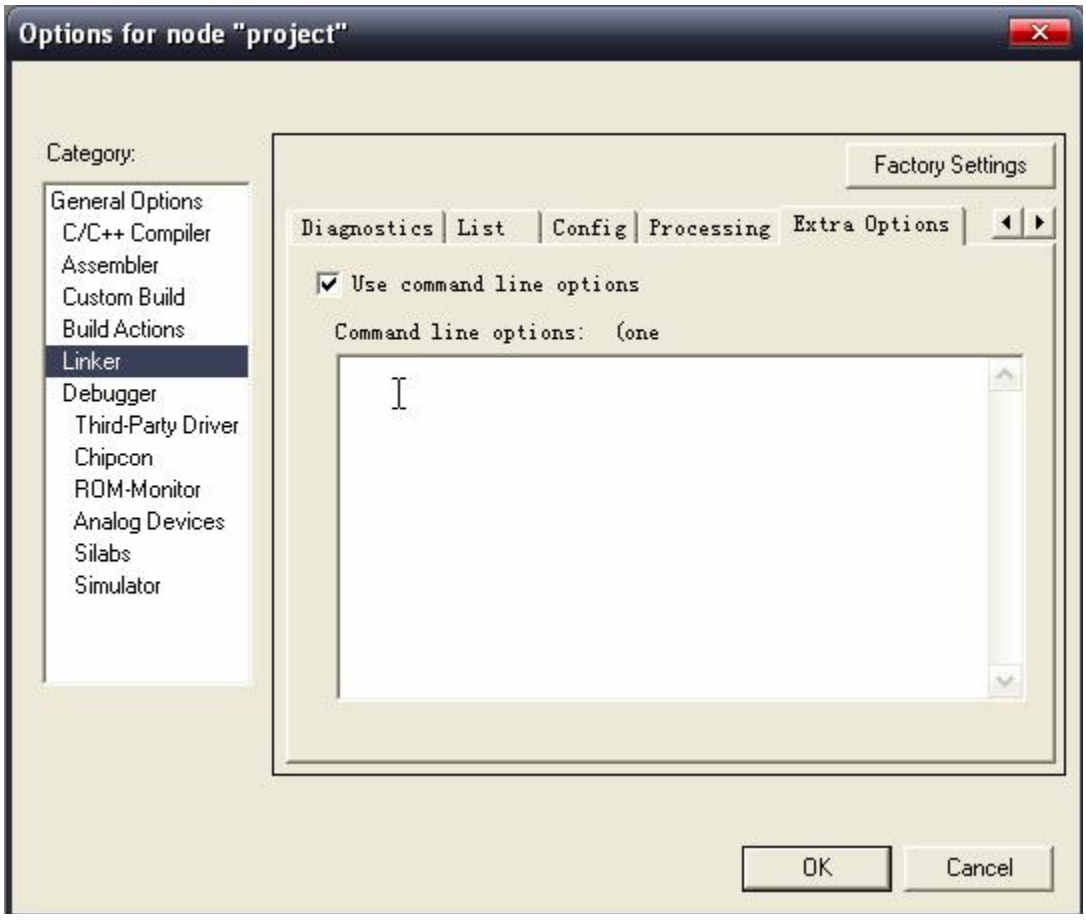


2) 在宏定义文件的文本框中，是用于用户自定义的一些宏定义，他的功能和**#define**相似，在具体应用中多做位条件编译使用，在这里就不多讲，在后面的应用中，会根据具体的使用给出使用方法。

3.linker设置

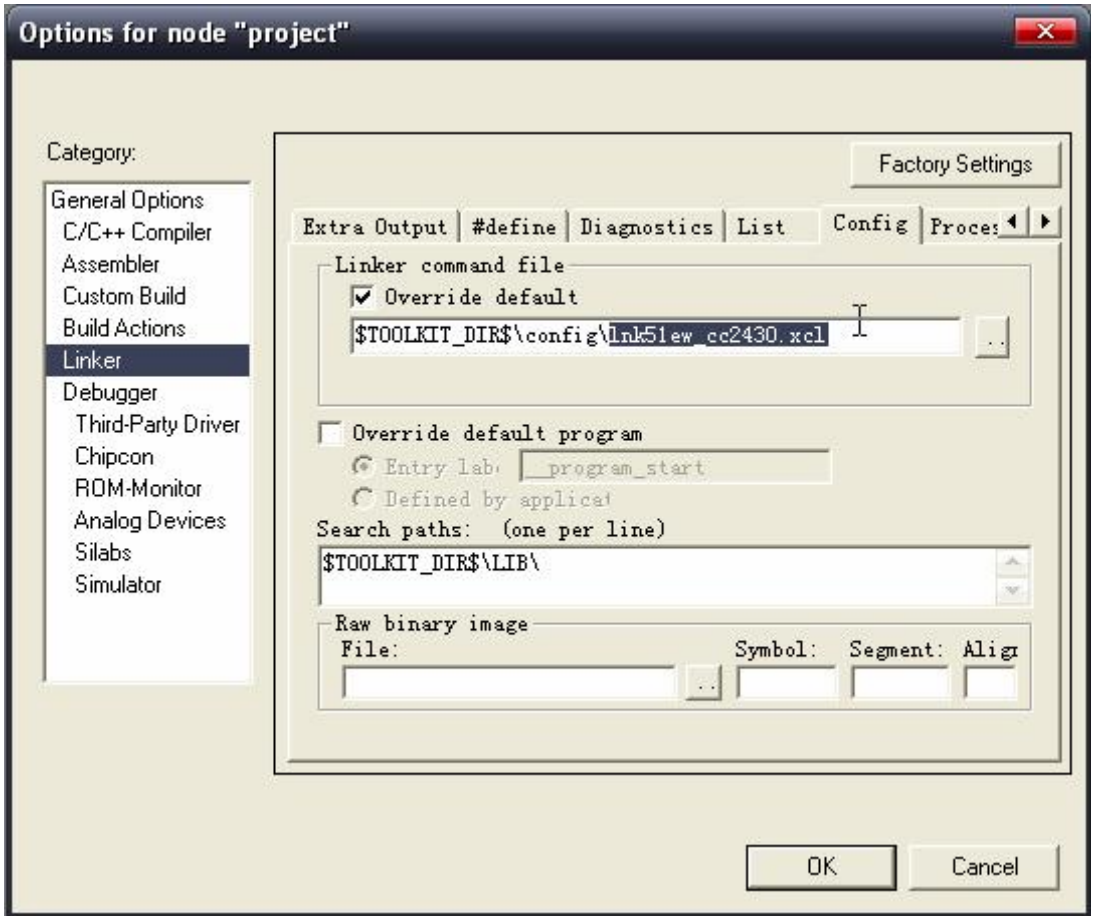
Linker->Extra Options中是用于包含一些必要的外部选项的，这里定义了各个设备的特殊功能选项，是一个用户自定义选项，在后面的应用中，会根据具体的使用给出使用方法。

地



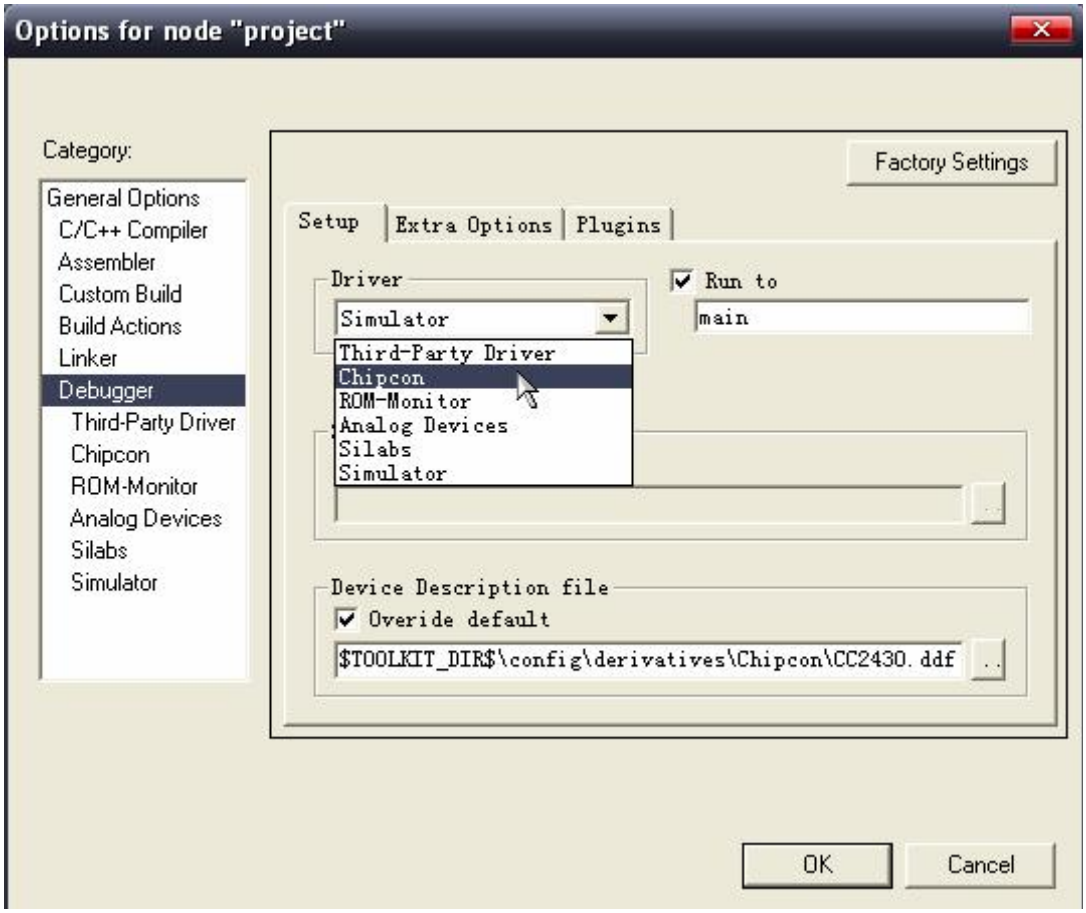
库函数添加

在Linker->Config中linker command file选择lnk51ew_cc2430.xcl。



4.Debugger设置

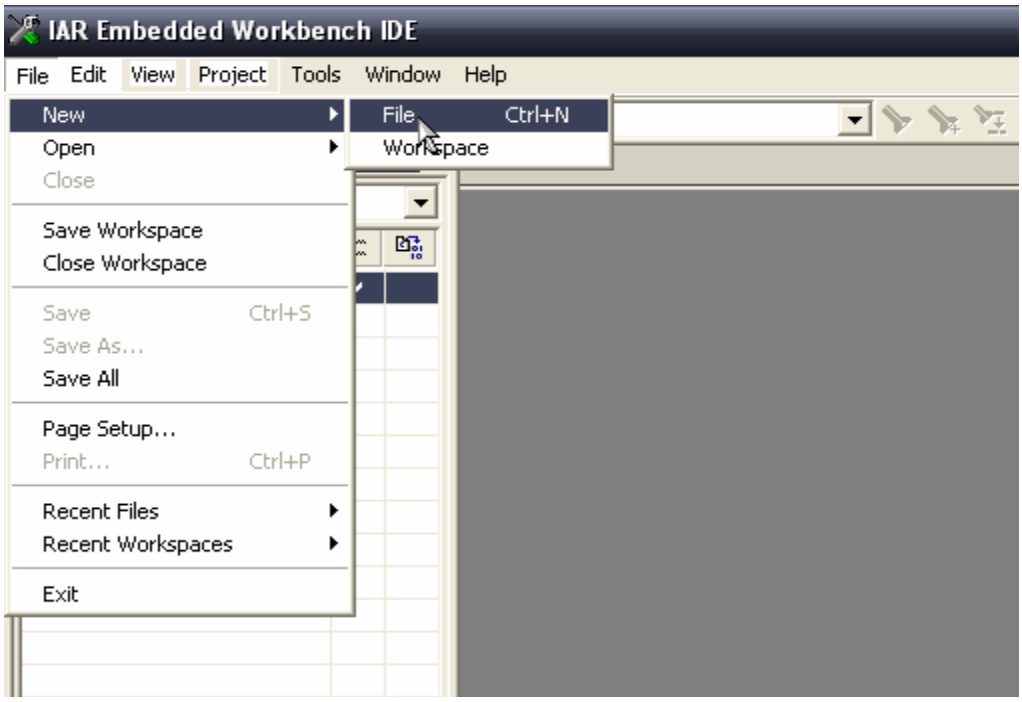
在Debugger->Setup中Driver项中选择Chipcon。



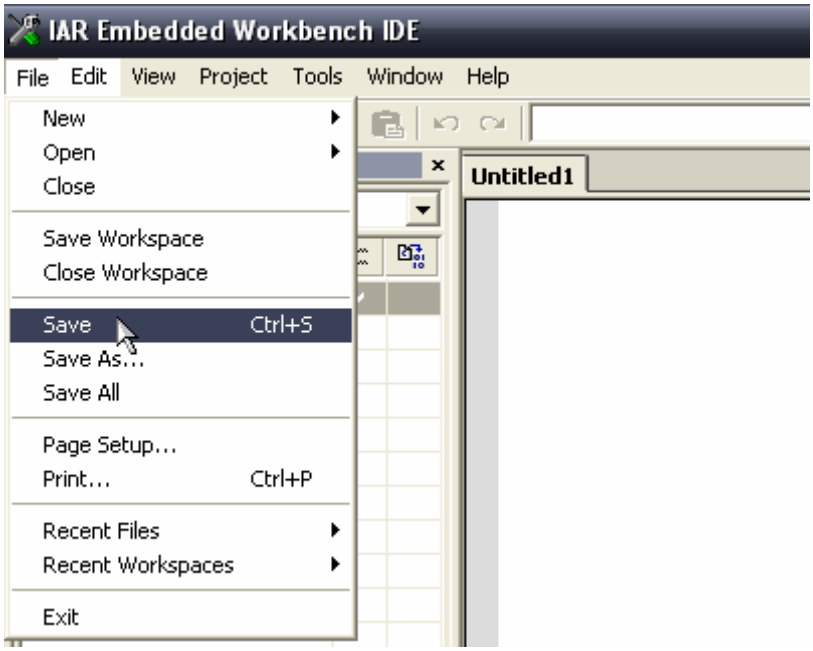
这个我们对于整个项目的基本设置就完成了。现在开始我们第一个项目开发。

2.5.3第一个项目

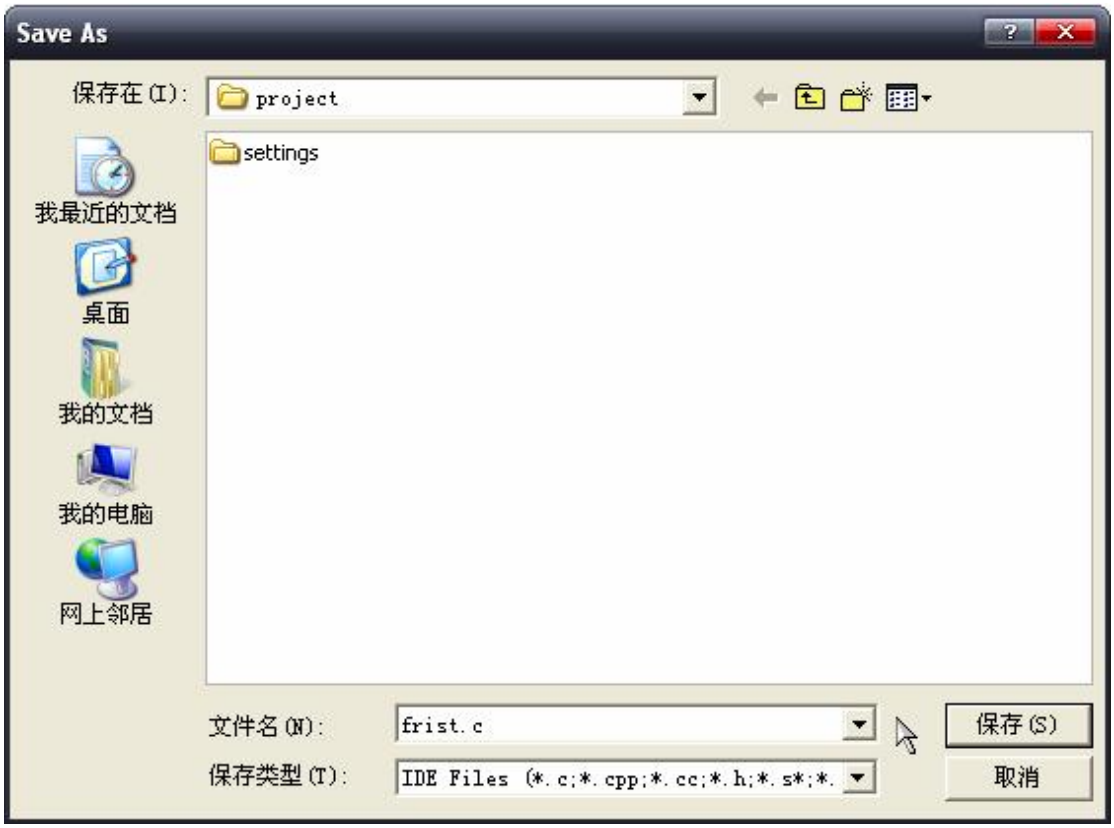
1.新建一个C文件，按图示步骤执行



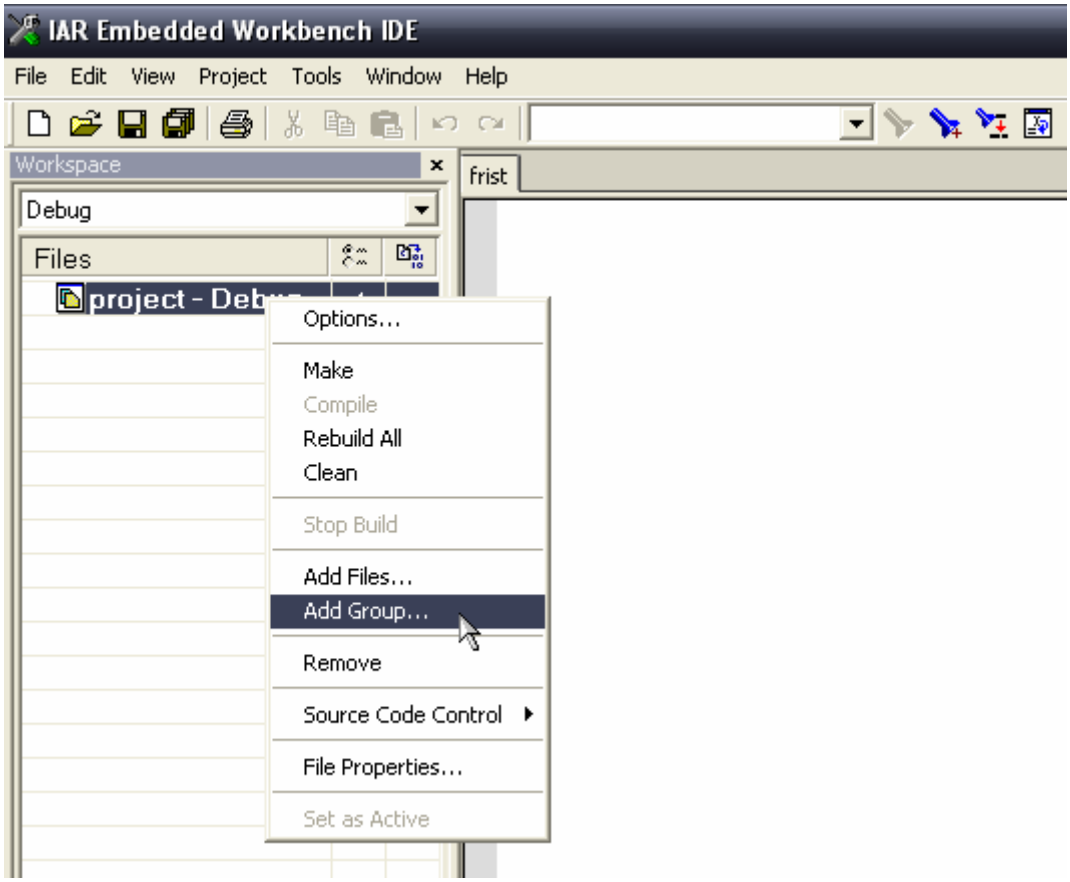
新建一个文件



保存文件

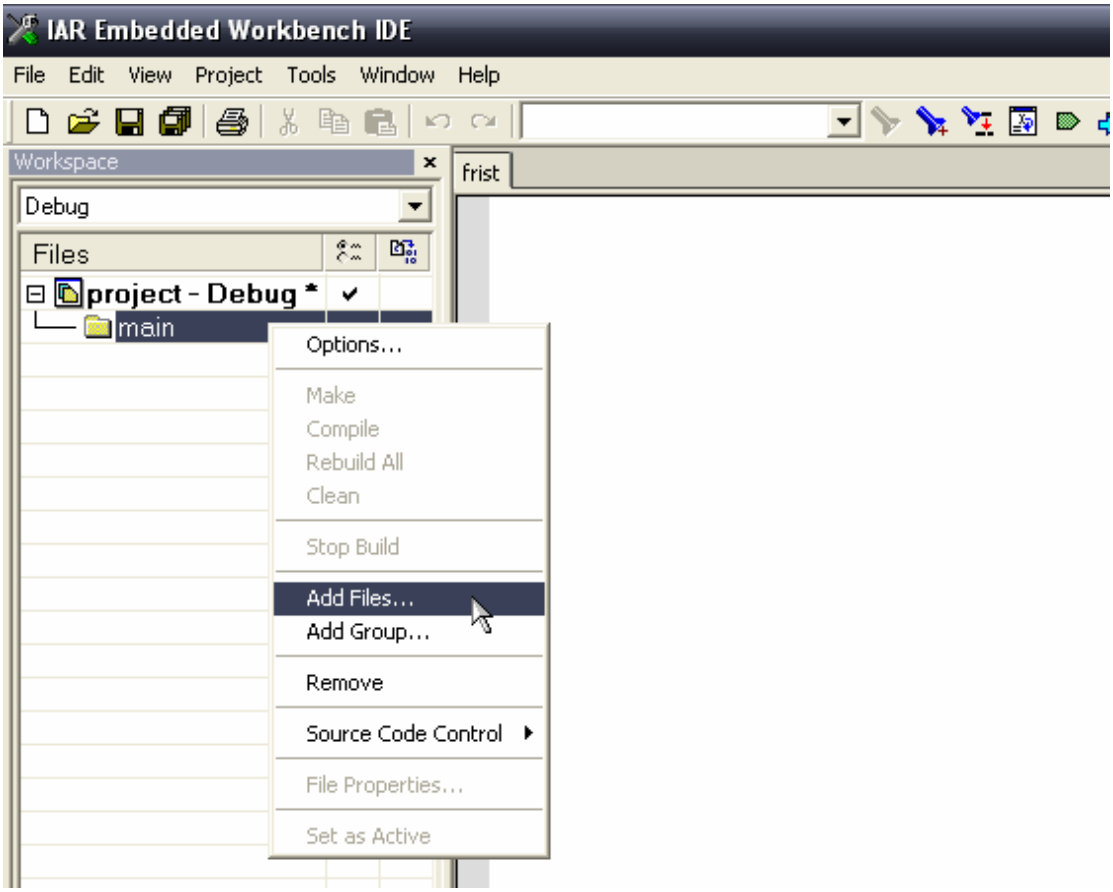


输入文件名，点击保存如果是C文件请务必后缀，否则会以文本文件存档。

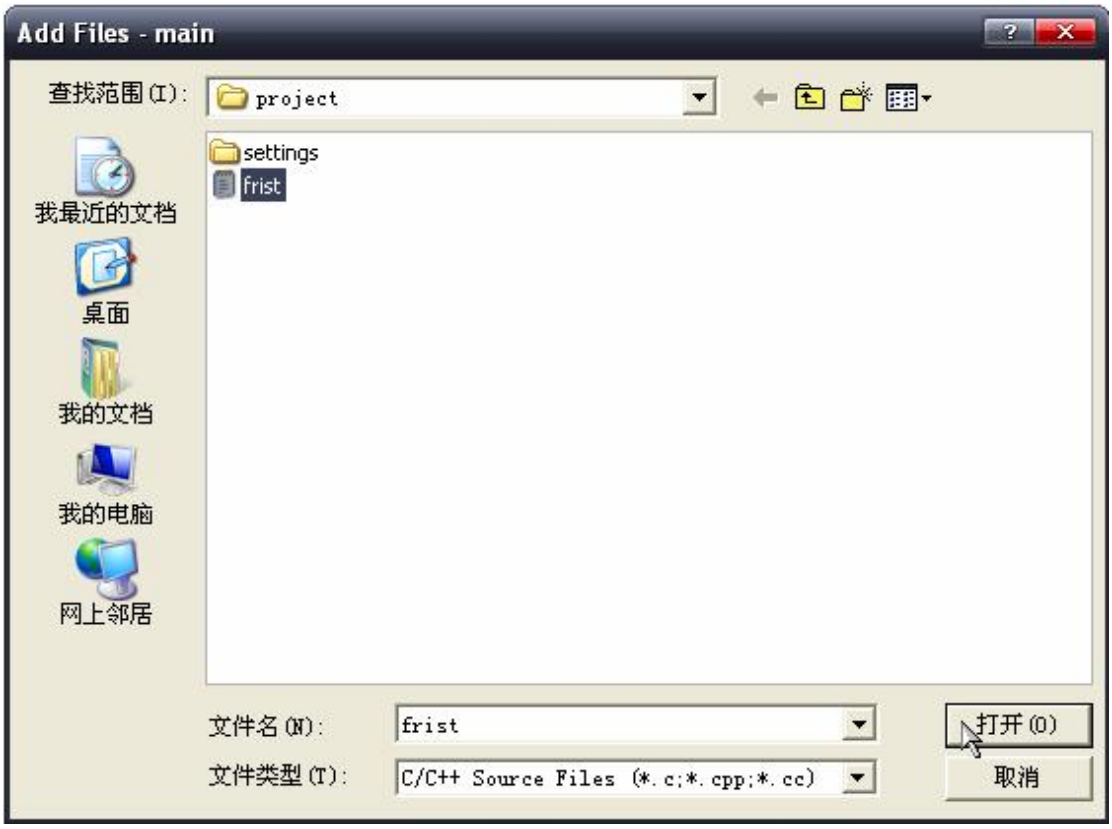


创建一个文件组

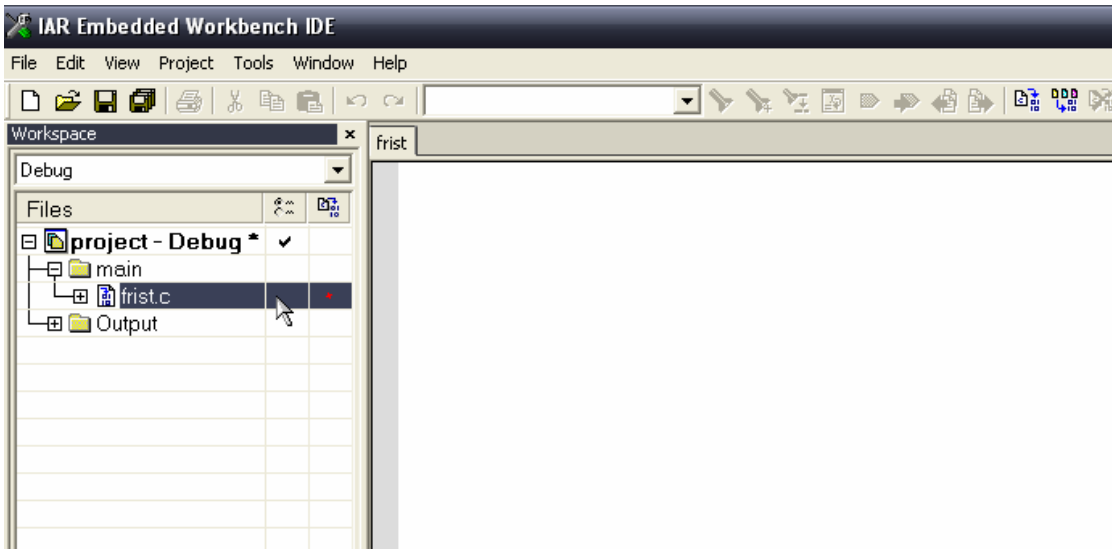
输入文件组名



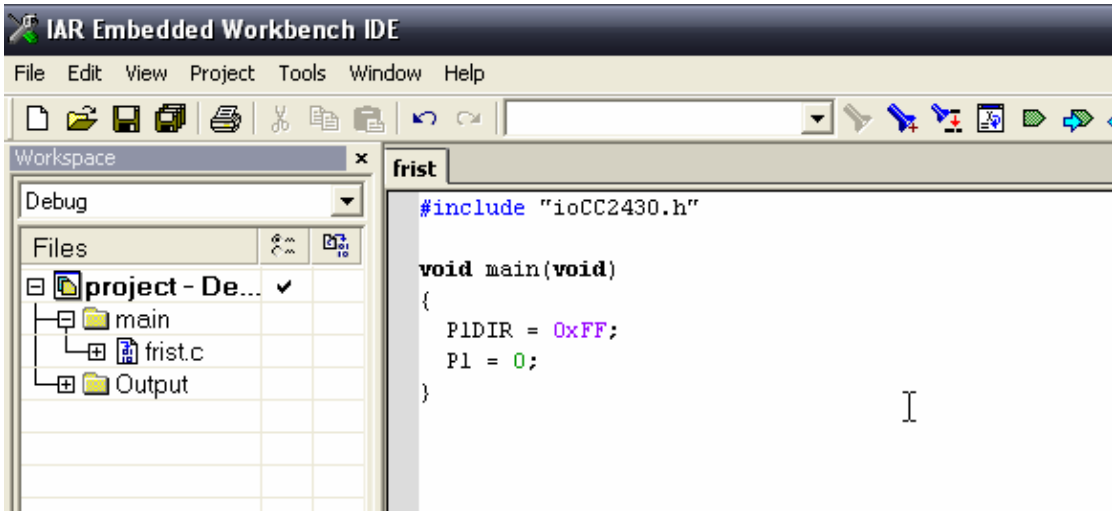
加入文件



选择新建的C文件

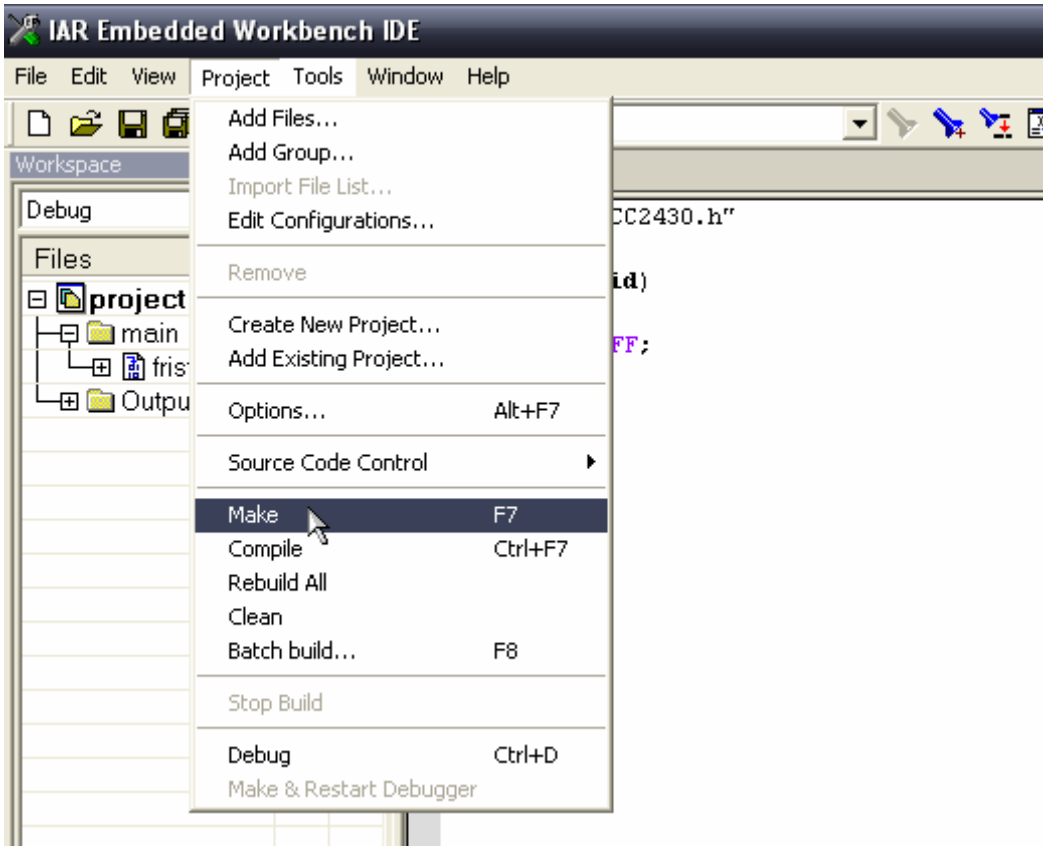


文件已经加入工程中，双击打开文件

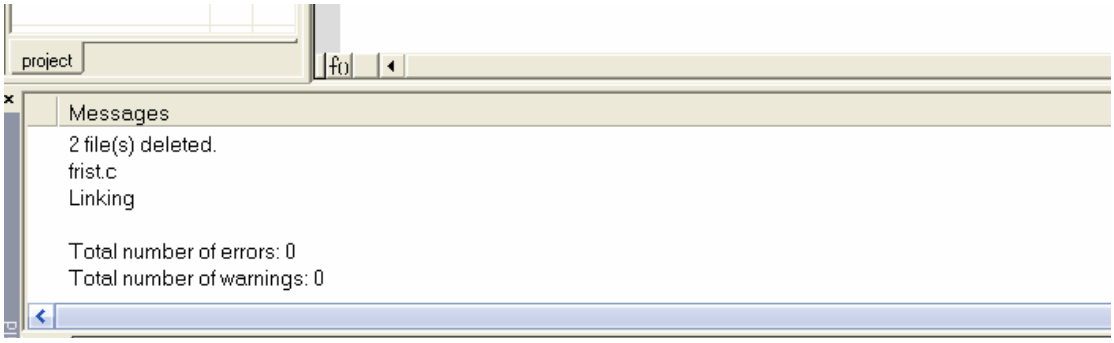


加入第一个代码，这个代码的意思是将P1口设置为输出，将P1口置0，无线龙模块和开发板中有小灯在P1口上，当执行这个代码的时候，小灯会点亮。

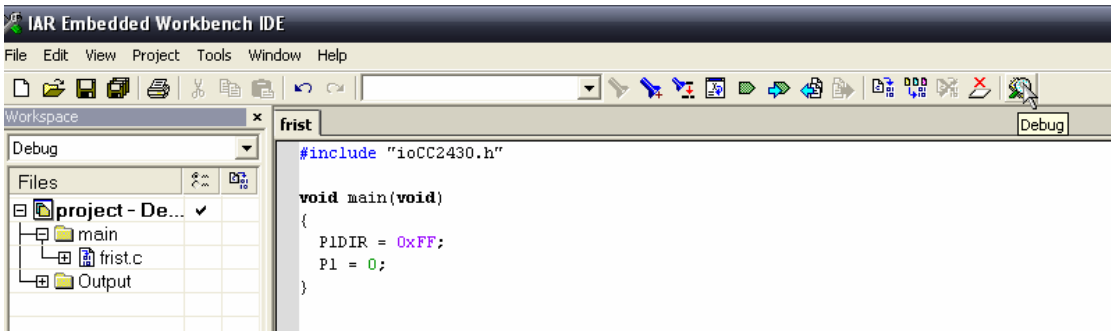
在实际的使用中如果IAR的工程路径有中文路径，有可能在调试的时候，设置断点的时候会不可见，所以我们将建立的工程拷贝到磁盘根目录中，这个我们将工程拷贝到D盘根目录。然后打开工程执行下面的步骤。



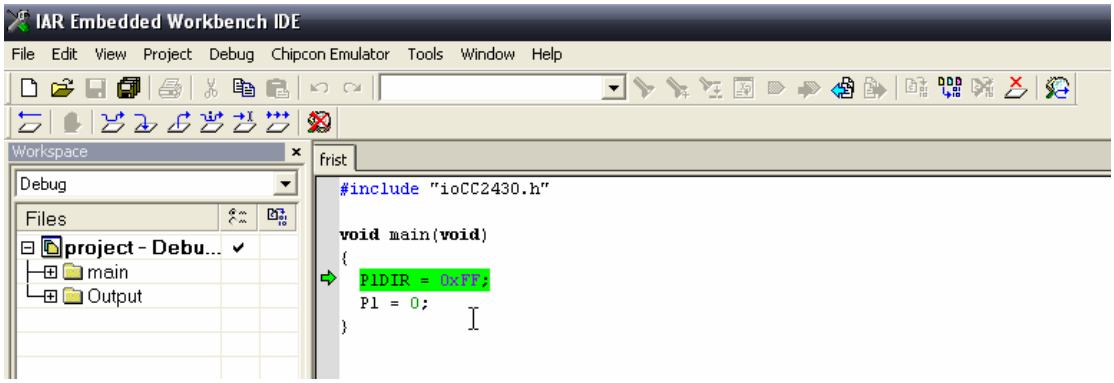
通过“make”编译，也可以通过Rebuild All全部编译，用make只会编译修改过的文件



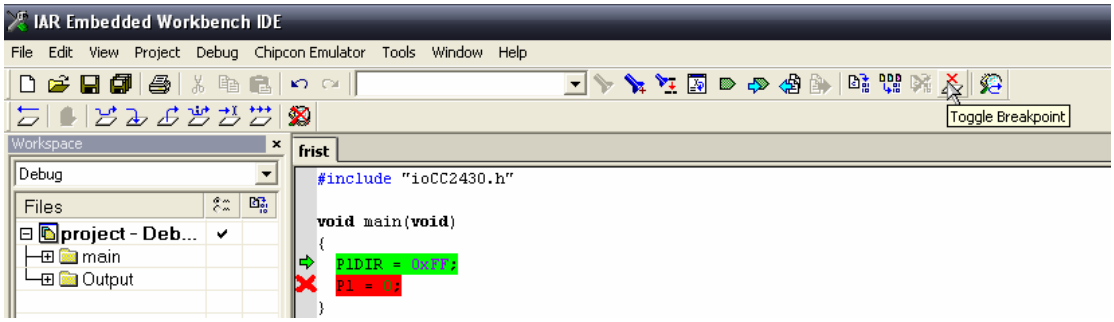
编译后只要没有错误就可以使用了，一般警告我们可以放过



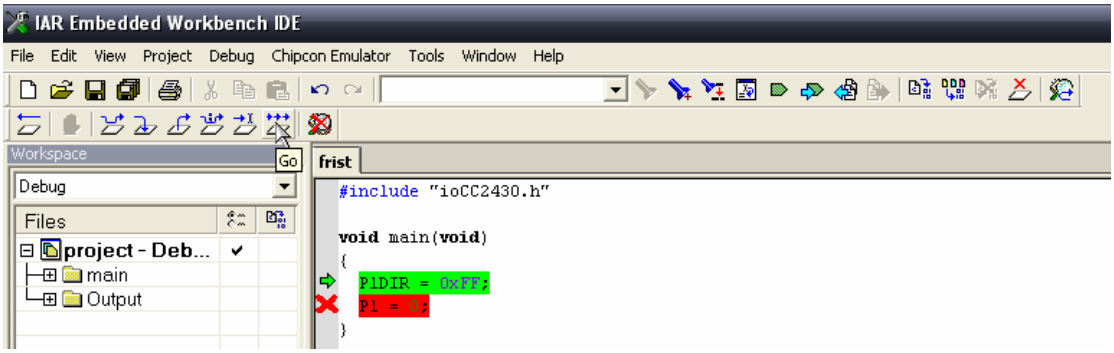
在编译没有错误后，就可以下载程序了，点击Debug，就现在程序了，下载程序后，软件进入在线仿真模式。



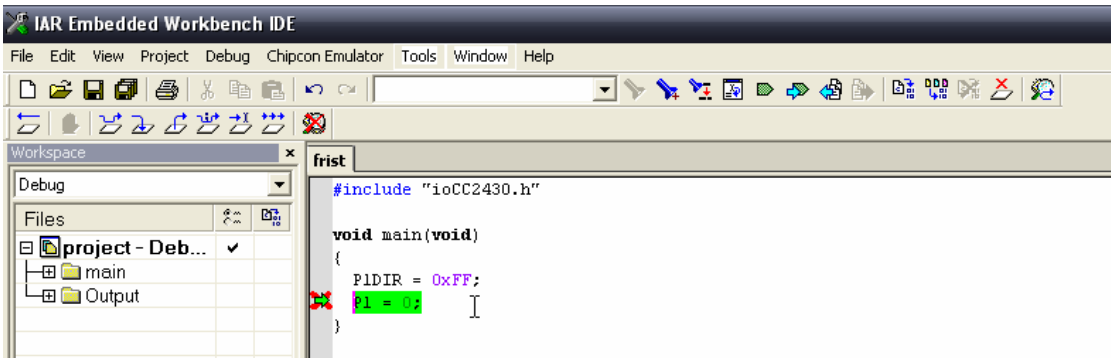
在仿真模式中，可以对这个文件设置断点，断点的设置方法是首先选择需要设置断点的行，然后单击Toggle Breakpoint设置断点。



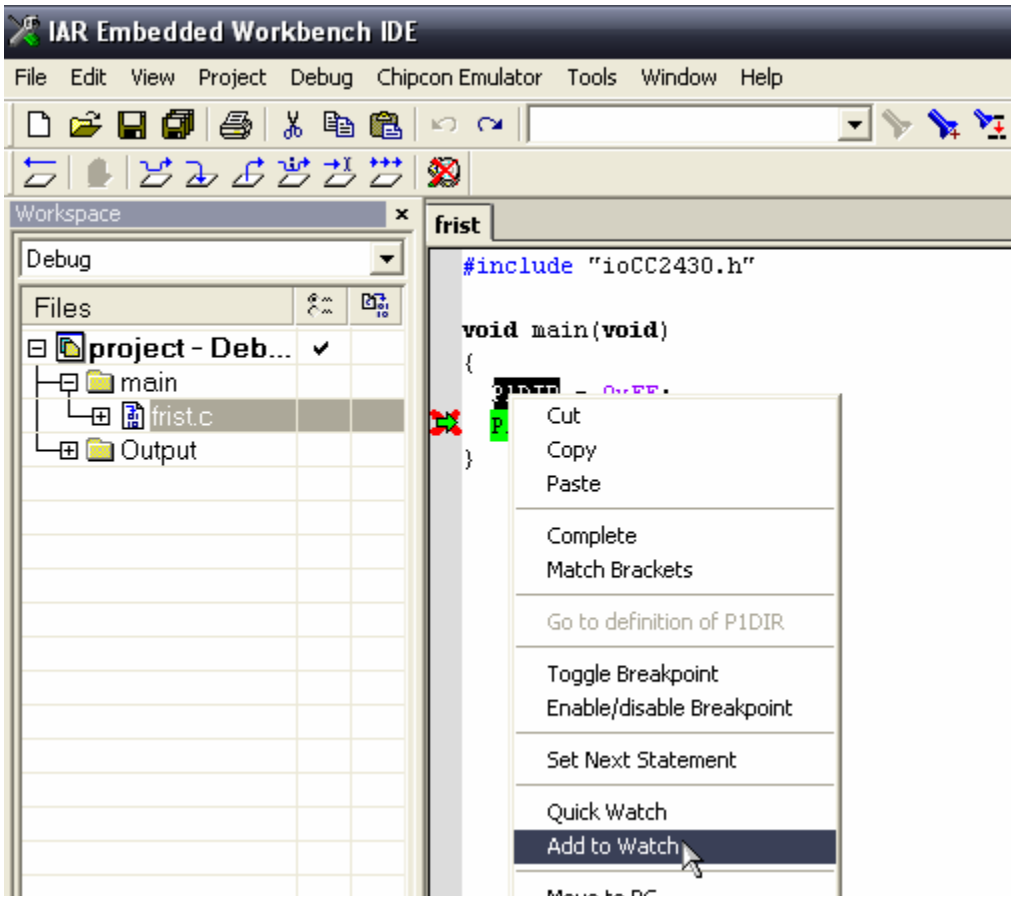
设置好以后，这行代码会变为红色，这样就表示断点设置已经完成。



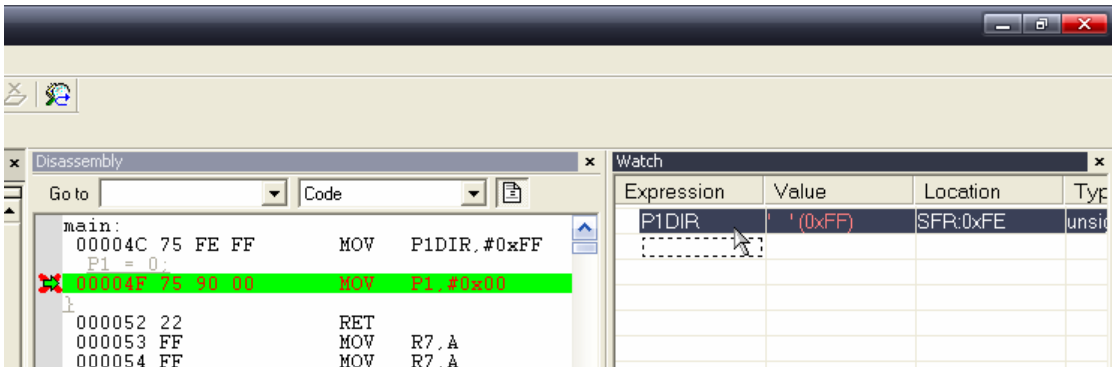
然后执行全速运行，当执行到断点出会停止在断点处。



然后双击P1DIR，单击右键，选择Add to Watch或者Quick Watch我们这里选择Add to Watch。



这个步骤的作用是查看这个寄存器中的值，如果是一个变量的话，就是查看一个变量的值。该值在Watch中可以看到。（请放大图片观看）



2.5.4 建立自己的模块设备

在上面的介绍中，大家对IAR的使用方法已经有了一定的认识，但是在一个项目中，尤其是无线项目中，涉及到的设备不会仅仅只有一个，如在zigbee设备中的协调器、路由器和终端，虽然设备不相同但它们的功能和协议栈底层却基本相同，所以在里面只需要定义些条件编译就可以设定好，如果将每一个设备都重新建立一个工程的话，这样寄浪费空间，代码也很混乱，所以我们就需要使用IAR的模块设备功能（自定义的一个名称，不准确请大家指正）。

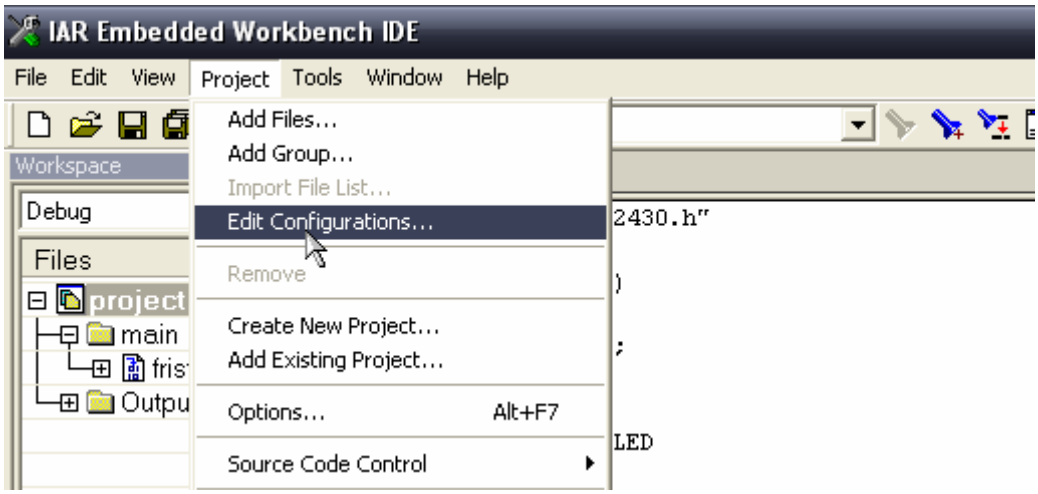
我们同样通过上面的例子来说明这个功能的使用方法。

首先修改工程的代码，在这里我们要实现两个设备在一个工程中实现不同的功能，两个模块的名称分别定义为：Blind_LED和Open_LED。实现的功能是闪烁小灯和打开小灯。将

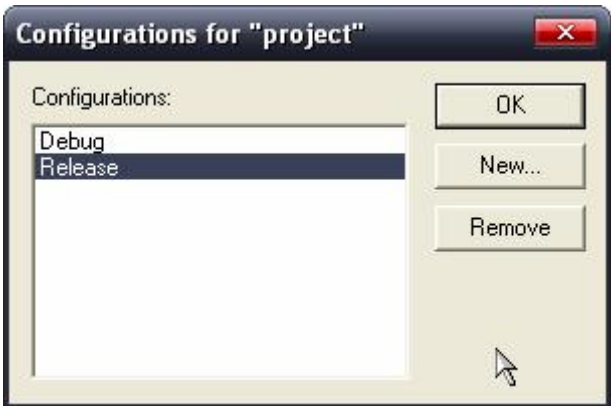
Project工程的frist.c文件中的主函数修改为下面的代码。

```
void main(void)
{
    P1DIR = 0xFF;
    while(1)
   {
#ifndef Blind_LED
    P1 = 0;
#else
    P1 = ~P1;
    for(int i=0;i<1000;i++)
    for(int j=0;j<1000;j++);
#endif
    }
}
```

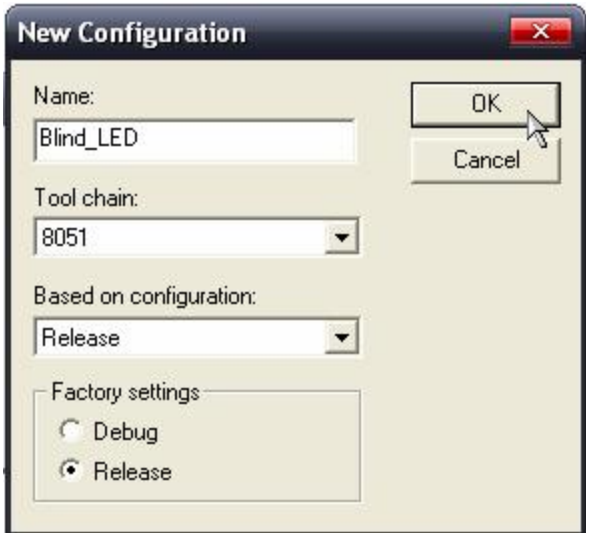
在代码中涉及到有两个条件编译，它们就是用来选择不同设备功能的。



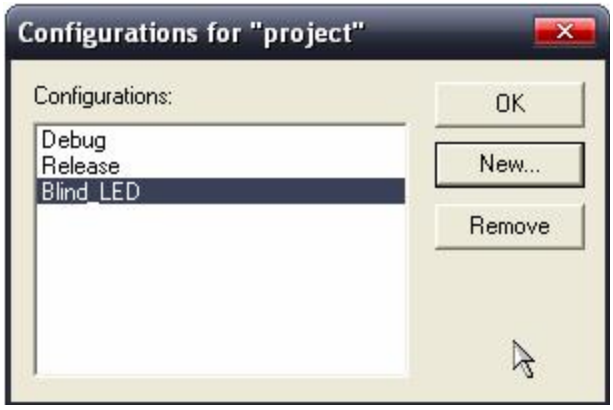
执行Project->Edit Configurations，这时候，系统会检测出项目中存在的模块设备。



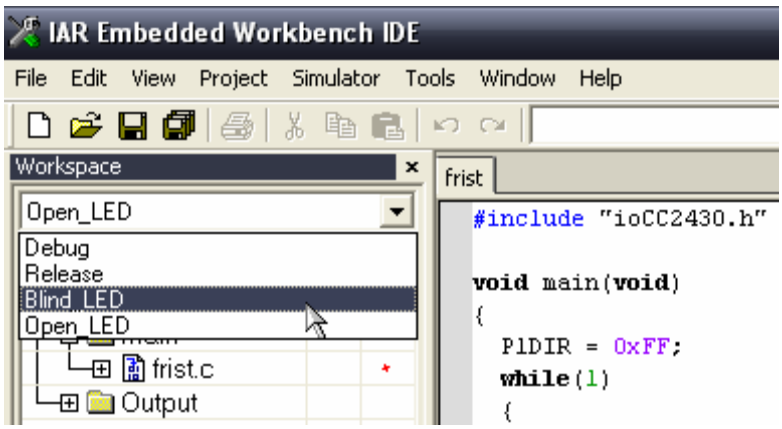
我们可以看到系统默认有两个设备，这这里我们需要生成我们需要的设备，也那就是Debug和Release，现在添加我们自己的设备，单击New。



输入名称，Blind_LED，在Factory settings中选择需要复制的设备，我们选择Release，这样Blind_LED就拥有了Release的全部功能，其他保持默认，单击OK，完成添加。



在工程配置选框中出现了我们添加的设备名称，按照刚才的方法继续添加Open_LED，然后单击OK，退出配置。



这时我们的Workspace就已经存在了我们添加的模块，从设备上看有两个设备我们不需要，我们可以将它删除，但是注意在这里激活的模块是不允许被删除的，所以我们要选择一个不删除的模块激活，然后再根据添加的步骤，选择Remove删除模块，这里就不多介绍了。

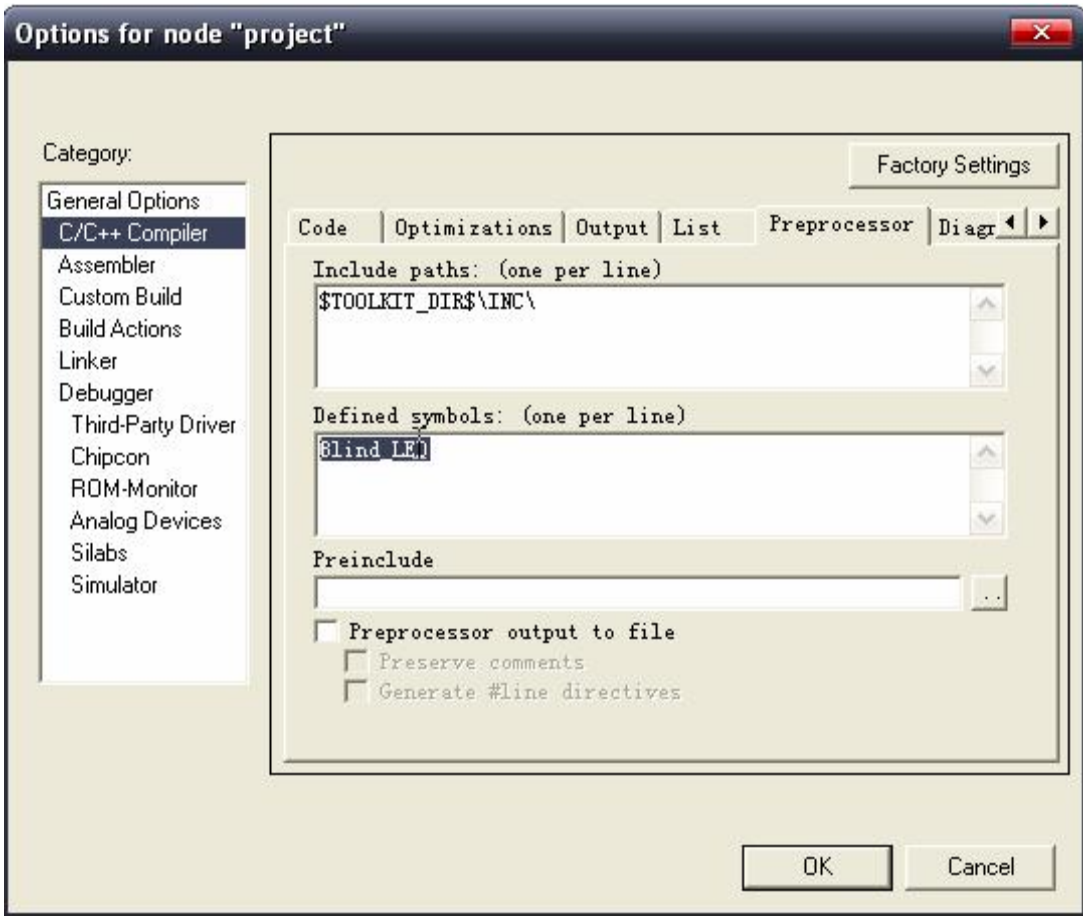
2.5.5 使用自己的模块

下面我们来通过模块的调用实现不同的功能，首先看看这几行代码实现的功能，

```
void main(void)
{
    P1DIR = 0xFF;
    while(1)
    {
#ifndef Blind_LED
      P1 = 0; //打开小灯
#else //否则，小灯闪烁
      P1 = ~P1;
      for(int i=0;i<1000;i++)
      for(int j=0;j<1000;j++);
#endif
      }
}
```

从代码中可以看出，根据条件的不同得到的结果也不相同，在这里实现的是一个闪灯的和一个打开小灯的程序，根据下面的图片进行配置，首先是打开小灯，打开小灯的是在没有定义Blind_LED的情况下实现的，所以我们只需要直接选择Open_LED模块，不需要任何修改就可以完成该功能。

我们着重介绍闪烁小灯，闪烁小灯是通过定义Blind_LED实现，由于我们要公用代码，所以定义最好不要在代码中，我们将他设置在C/C++ Compile->Preprocessor选项中的Defined symbols中，具体实现的方法如图解。



这样就定义好了，然后根据上面介绍的步骤编译现在，就可以看到小灯闪烁了。这个功能在

后面的例子中有广泛的应用。