

附录 A CCS 使用入门

CCS (Code Composer Studio) 是 TI 公司研发的一款具有环境配置、源文件编辑、程序调试、跟踪和分析等功能的集成开发环境, 能够帮助用户在一个软件环境下完成编辑、编译、链接、调试和数据分析等工作。其他一些集成环境如 IAR 公司的 EW430 也具有相似功能。本文以 CCSv9.3 为例, 说明它的基本使用方法, 9.3 版本后的 CCS 操作相似。

在 CCS 的控制下, 利用调试器通过单片机的 JTAG 接口可将在 PC 侧开发的单片机程序下载(俗称烧写, programming)到单片机上。JTAG (Joint Test Action Group 联合测试行动小组)是一种国际标准测试协议, 主要用于芯片内部测试。标准的 JTAG 接口有 4 线, 时钟 TMS、模式选择 TCK、数据输入 TDI 和数据输出 TDO。JTAG 最初是用来对芯片进行测试的, 其基本原理是在器件内部定义一个 TAP(Test Access Port 测试访问口), 通过专用的 JTAG 测试工具对内部节点进行测试。现在大部分高级器件都支持 JTAG 协议, 例如 DSP、FPGA 器件等。MSP430 单片机内部也具有 JTAG 接口, 可连接 JTAG 调试器, 与最初主要用于测试不同, 现在的 JTAG 接口不仅可以在系统下载程序(简称 ISP, In System Programming), 还可以在系统调试程序, 方便了用户的应用系统开发。图 A-0-0 为 JTAG 调试器与 PC 机和目标板的连接图。

如图 A-0-1, 同学们手上拿到的 msp430g253 单片机小板, 是一块学习用的实验板。该板将调试器和单片机电路做在了一个电路板上, 调试器和单片机通过简单的跳线块就可连接。将这种在实验板上的调试器称为板载仿真器。如图 A-0-1, 用一根 miniUSB 数据线将 PC 机与实验板连接, 就可利用调试器将程序下载到单片机内, 并对单片机进行在线调试。

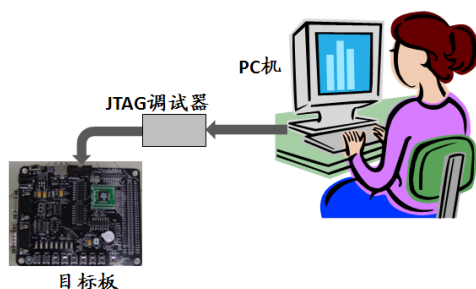


图 A-0-0 经 JTAG 调试器与目标板连接

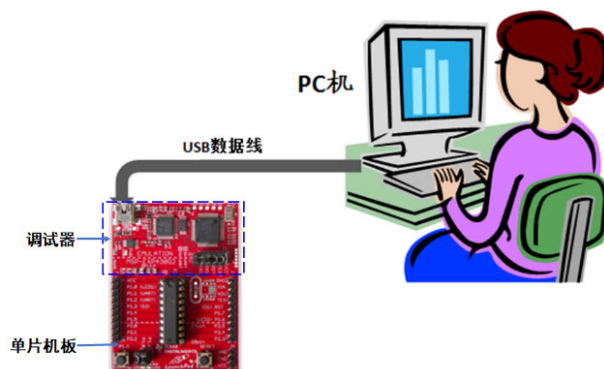


图 A-0-1 PC 机经板载仿真器与目标板单片机连接

本附录按照下面几个方面介绍 CCS 的基本使用:


- 一、创建 C 语言项目
- 二、编辑和添加源程序
- 三、编译和连接程序
- 四、连接相关硬件
- 五、下载程序到目标 MCU 中
- 六、运行程序及相关命令
- 七、查看当前单片机状态
- 八、退出 DEBUG
- 九、在一个工程空间中管理多个项目
- 十、CCS 提供的 Help 文档的使用

一、创建 C 语言项目

1. 创建工程空间文件夹

在自己的电脑硬盘上，创建一个存放 CCS 项目的文件夹，文件夹名字可自取，本例用 D:\0-CCS930 作为 CCS 的工作文件夹。注意：文件夹路径不要出现中文文字。如果使用实验室的电脑，只能将文件建在 F: 盘符下，C~E 盘均设有保护，重启系统，C~E 盘用户的内容(程序)不被保存，将丢失。

2. 启动 CCS

双击电脑桌面 Code Composer Studio 9.3.0 快捷图标，运行 CCS，将出现图 A-1 选择工作空间文件夹的界面，默认的文件夹是 C:\CCS\Projects\workspace。单击 Browse，将工作区间选择到所建文件夹 D:\0-CCS930 下。可勾选"Use this as the default and do not ask again"，此后启动 CCS 将默认该文件夹作为工作空间。点击 Launch，首次使用会进入到图 A-2 CCS 的 Getting Started 界面。在图 A-2 的 Getting Started 界面，可以点击 Getting Started 右侧的 X，关闭该界面；如有需要，可如图 A-3 选择 Help→Getting Started 恢复。

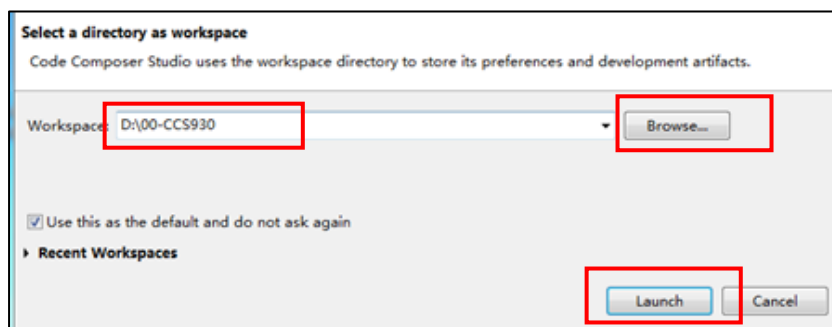


图 A-1 选择用户工作空间

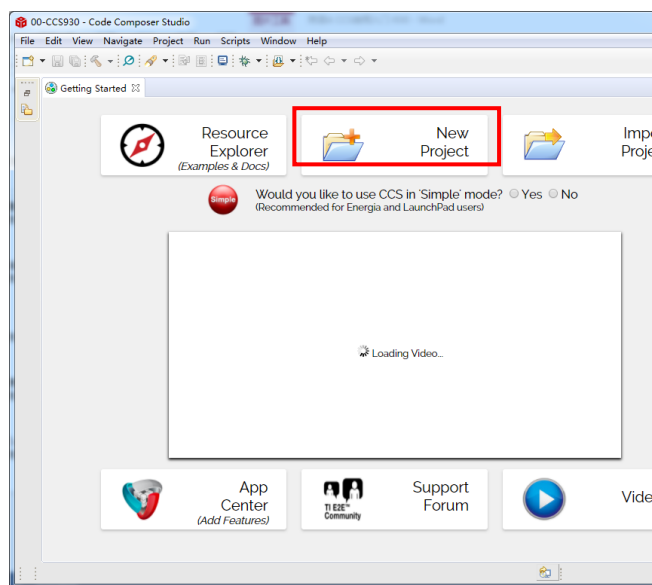


图 A-2 Getting Started 界面

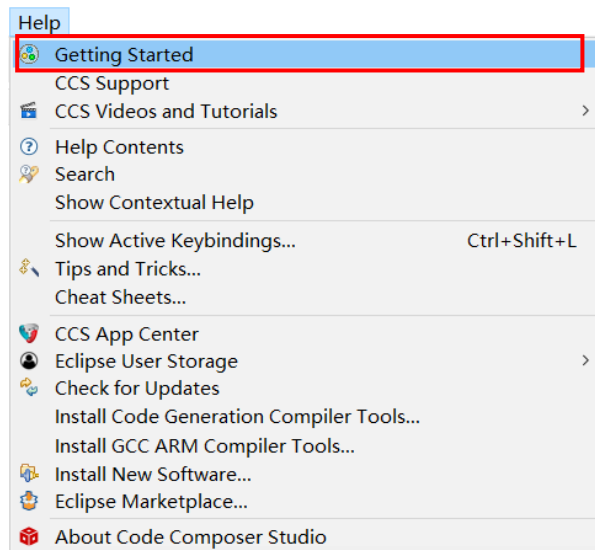


图 A-3 Help 栏中恢复 Getting Started 界面

3. 创建 C 语言项目

- 1) 可以点击图 A-2 中的 New Project 图标；或如图 A-4 选择 File → New → CCS Project；或如图 A-5 选择 Project → New CCS Project，出现如图 A-6-1 的界面。

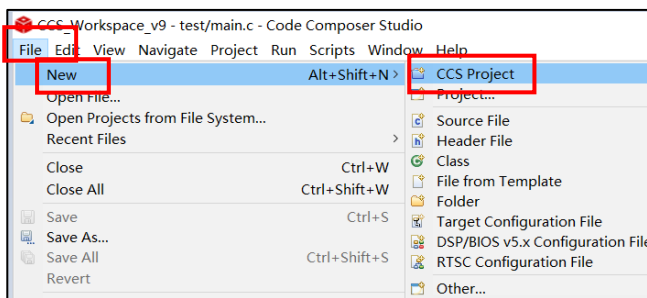


图 A-4 在菜单栏 File 中创建项目

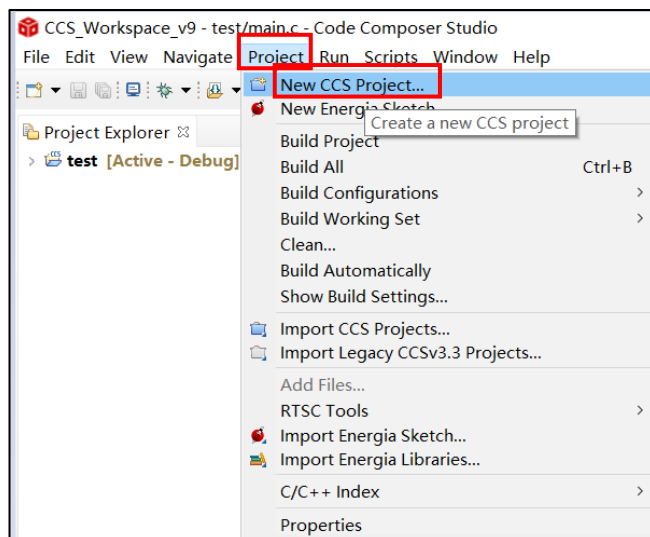


图 A-5 在菜单栏 Project 中创建项目

- 2) 如图 A-6-1, (1) 选择 Target 中的 MCU 系列: MSP430Gxxx Family; (2) 选择 Target 中的 MCU 型号: MSP430G2553; (3) 点击 Project name 框, 输入项目名, 如 test, 项目名可任取, 注意不用中文; (4) 在 Project templates and examples 中选 Empty project (with main.c); (5) 单击 Finish, 完成相关属性的设置, 进入到图 A-6-2 的项目编辑页面, 其中 main.c 是项目提供的模板文件, 如图 A-7。

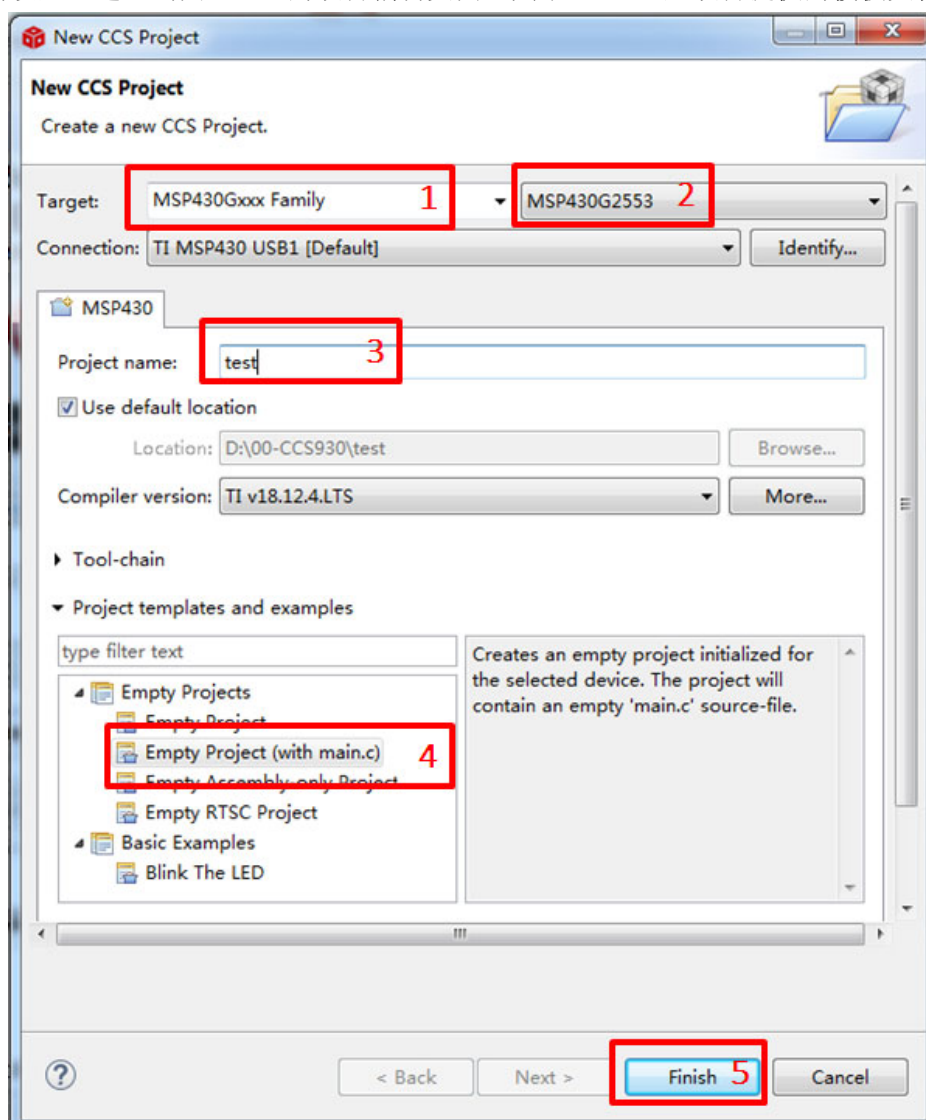


图 A-6-1 设置项目相关属性

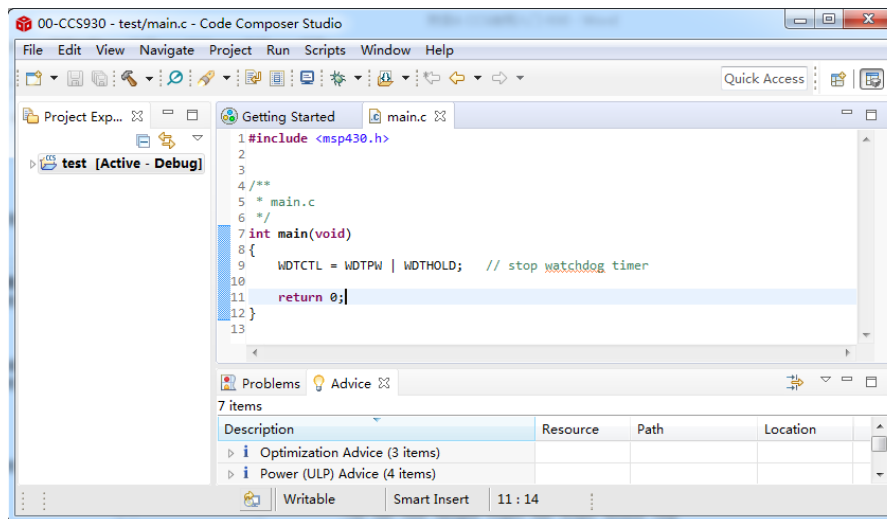


图 A-6-2 项目编辑页面

```
#include <msp430.h>
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    return 0;
}
```

图 A-7 CCS 创建项目后自带的模板文件 main.c

二、编辑源程序

方法 1: 可在模板文件 **main.c** 基础上编辑源文件，第一次实验时可将提供的 test_g2553.c 内容拷贝，粘贴到 main.c 中，替代原 main.c 全部内容，如图 A-8。test_g2553.c 提供电子版。

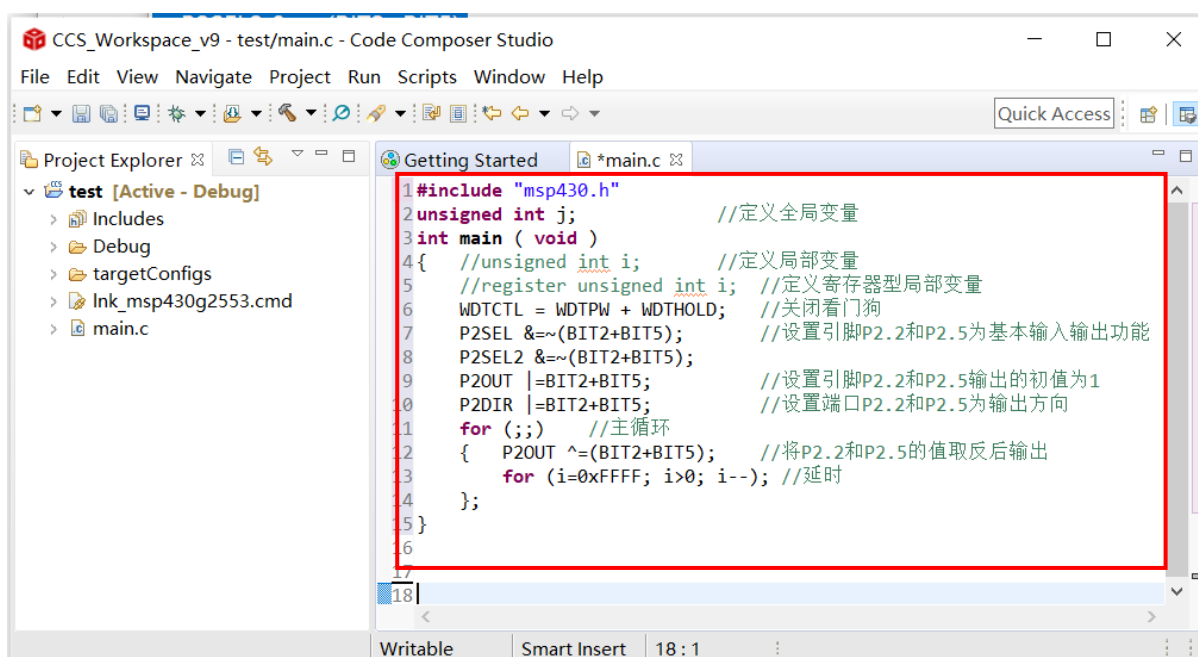


图 A-8 在模板文件基础上，编辑用户程序

```
// test_g2553.c (提供电子版)
#include "msp430.h"
unsigned int j;           //定义全局变量
int main ( void )
{
    //unsigned int i;      //定义局部变量
    //register unsigned int i; //定义寄存器型局部变量
    WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
    P2SEL &=~(BIT2+BIT5);    //设置引脚 P2.2 和 P2.5 为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;       //设置引脚 P2.2 和 P2.5 输出的初值为 1
    P2DIR |=BIT2+BIT5;       //设置端口 P2.2 和 P2.5 为输出方向
    for (;;) //主循环
    {
        P2OUT ^= (BIT2+BIT5); //将 P2.2 和 P2.5 的值取反后输出
        for (i=0xFFFF; i>0; i--); //延时
    }
};
}
```

方法 2：将编辑好的文件添加到项目中

CCS 有一个便捷操作，如图 A-9-1，把项目文件（如 test_g2553.c 文件）直接拷贝到项目所在文件夹下（如 D:\0-CCS740\test 下），即可自动将该文件加入进项目，如图 A-9，项目中出现了拷贝进来的文件，如 test_g2553.c。此时需要将 main.c 文件移出项目，因为该文件中也包含有一个 main 函数，而一个项目中可以有多个.c 源文件，但 main 函数只能有一个。如图 A-20-1，选择要移出的文件，如 main.c，单击右键，会出现如图 A-10-1 对该文操作的界面。注意：**移出文件时不建议选择 Delete，该操作会直接从硬盘上删除选中的文件**，且该过程不可逆，即文件一旦 delete，就不存在了。**建议选择 Exclude from Build**，将选中的文件移出项目，移出后文件还在原文件夹下，只是不在项目中了。移出后的文件如需要可点击 Exclude from Build 前的勾选项，重新将文件加入，如图 A-10-2。移出后的文件在项目中显示为灰色，如图 A-10-3。

如果拷贝到项目文件夹下的文件，没有出现在项目中，可以如图 A-11，右键出现的 Add files，找到要添加的文件，即可将文件加入项目中。

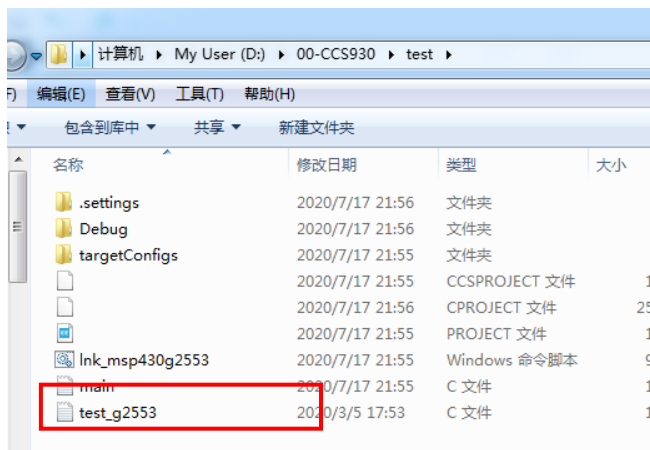


图 A-9-1 把文件拷贝到项目所在文件夹下

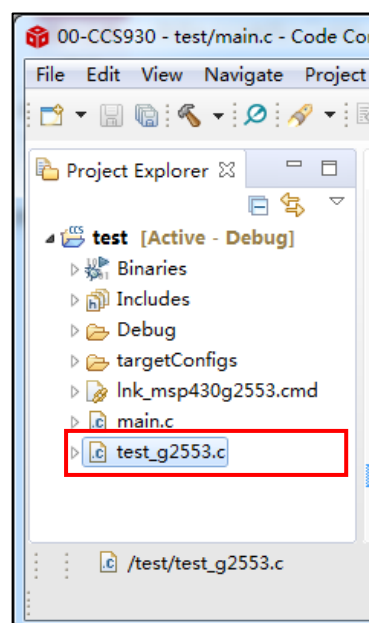


图 A-9-2 拷贝到项目所在文件夹下的文件自动加入到项目中

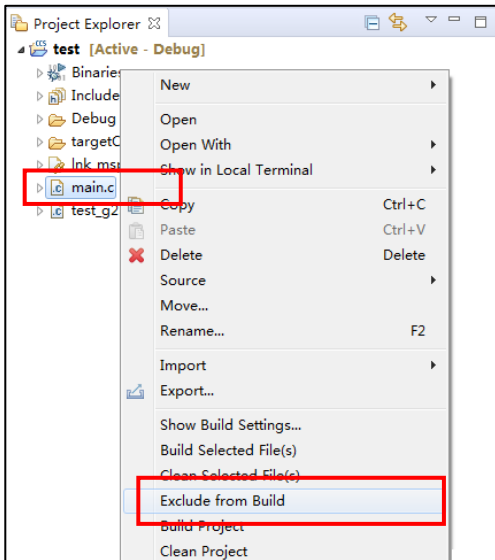


图 A-10-1 选择 Exclude from Build
将文件从项目移出

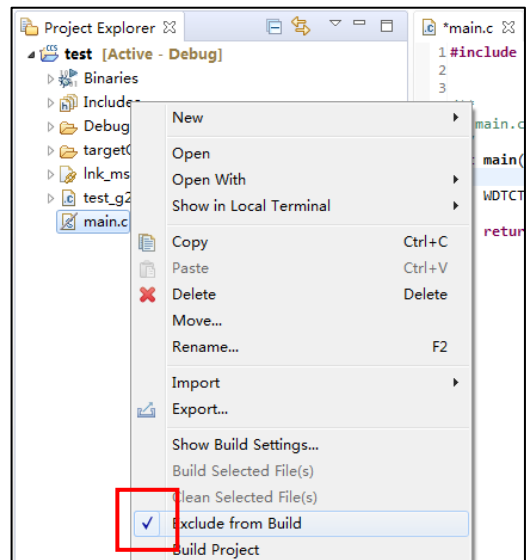


图 A-10-2 点击 Exclude from Build 前勾选选项，
可将文件重新加入项目

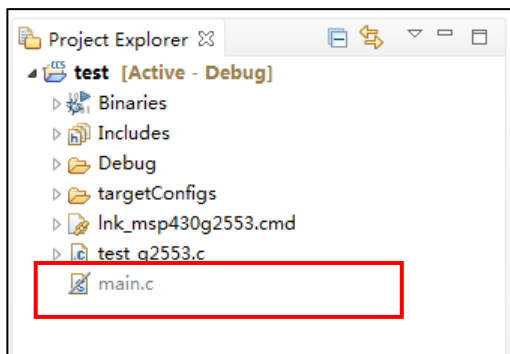


图 A-10-3 移出后文件在项目中显示为灰色
(如图中 main.c)

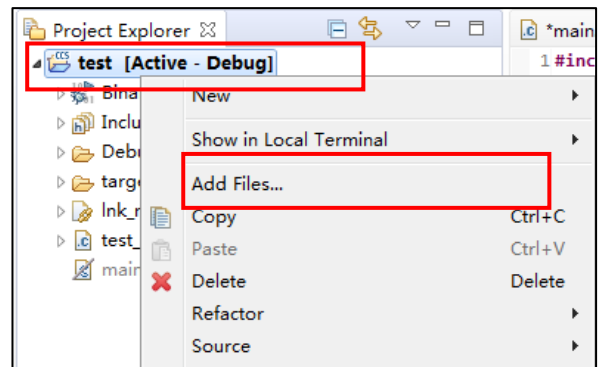


图 A-11 用 Add files 添加文件

三、编译和连接程序

在编译连接程序前，可根据用户对程序编译的需求，设置编译器的优化级别。如图 A-12-1，点击菜单栏 Project > Properties，如图 A-12-2 在出现的窗口中，点击左侧的 Build > MSP430 Compile > Optimization，将右侧的 Optimization level 从原来默认设置的 0-Register Optimization 改置为 Off，即关闭优化功能。建议本学期实验，无特别要求下，都选 off 级调试。

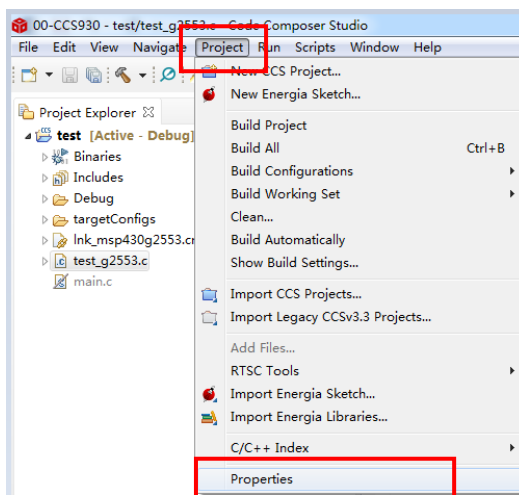


图 A-12-1 设置项目属性

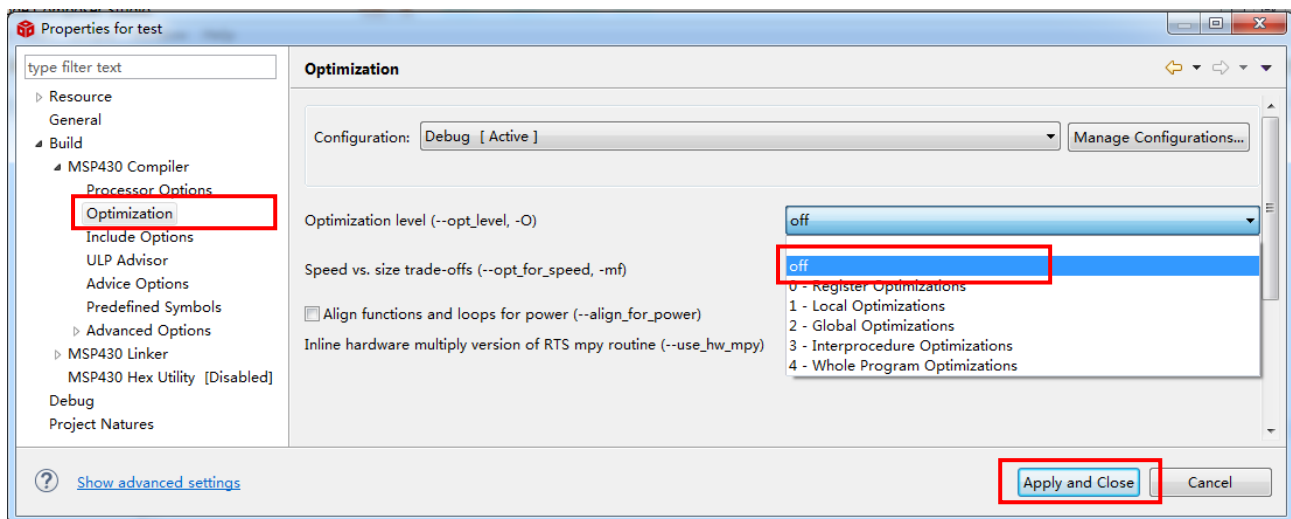



图 A-12-2 设置编译器的优化级别

然后开始编译和连接程序：可如图 A-13 单击菜单栏的快捷图标 ；或者如图 A-14 在 Project Explorer 下选中目标 Project（图中左侧黑体 **test[Active Debug]** 处），点击鼠标右键，在出来的窗口中选择 Build Project；也可以如图 A-15，在菜单栏中点击 Project > Build Project。

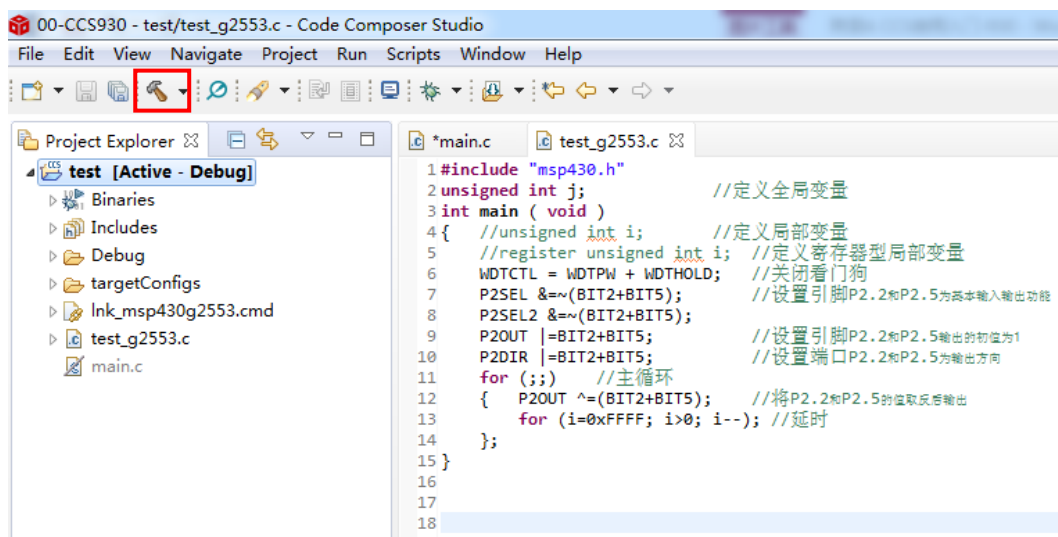


图 A-13 单击菜单栏 Build 快捷图标

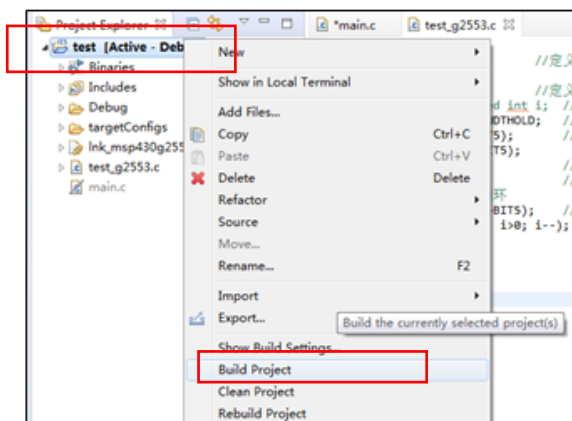


图 A-14 在项目操作框中选择 Build Project

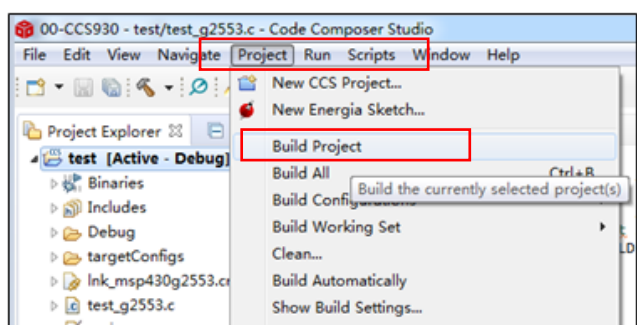


图 A-15 单击菜单栏中的 Project 中的 Build Project

图 A-16-1 是编译后给出的信息，一般会在源文件中用醒目的图标指示程序可能有问题的语句，并在 Console 和 Problems 窗口中给出编译过程中的信息，含错误信息。Console 和 Problems 窗口可如图 A-16-2 点击菜单栏 Windows>Show View 选择打开。根据 Console 窗口中的提示，改正程序中的错误。一般注意错误提示信息中其中提到的变量名、或函数名，看看与它们相关联的问题。找到问题，修改程序，重新 Build 项目。直到如图 A-16-3 形成以可执行的项目名 xxx.out 文件(如 test.out)，表示编译完成。编译系统会给一些低功耗的建议，可以暂时忽略。

注意：提供的 test_g2553.c 故意留有一个简单的程序小错误，编译时在 Console 窗口会有错误的提示，请利用错误提示，修改程序，然后重新编译。

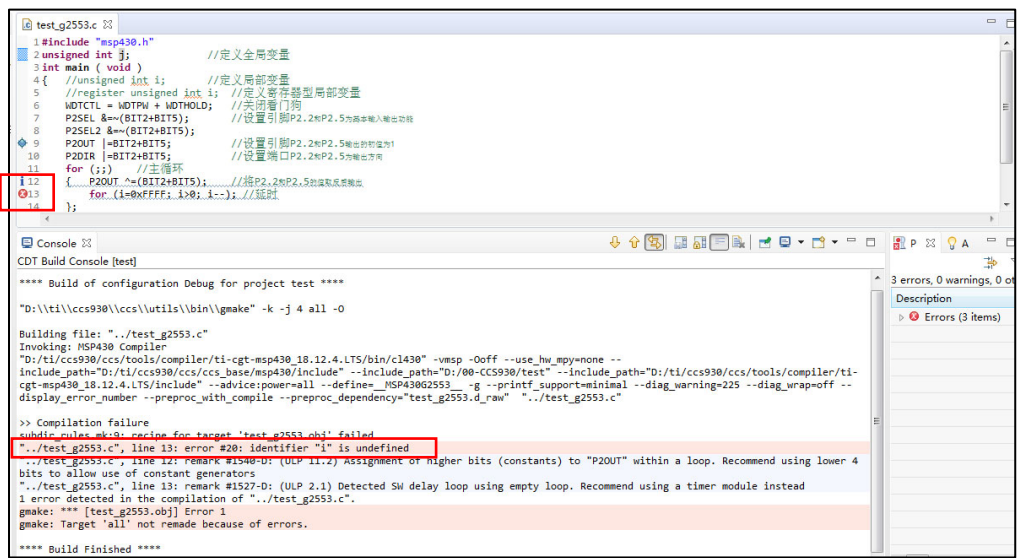


图 A-16-1 编译后给出相关信息

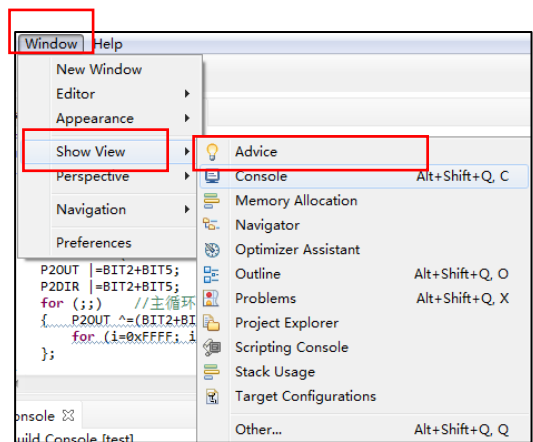


图 A-16-2 选择 Build 时需要打开的观察窗口

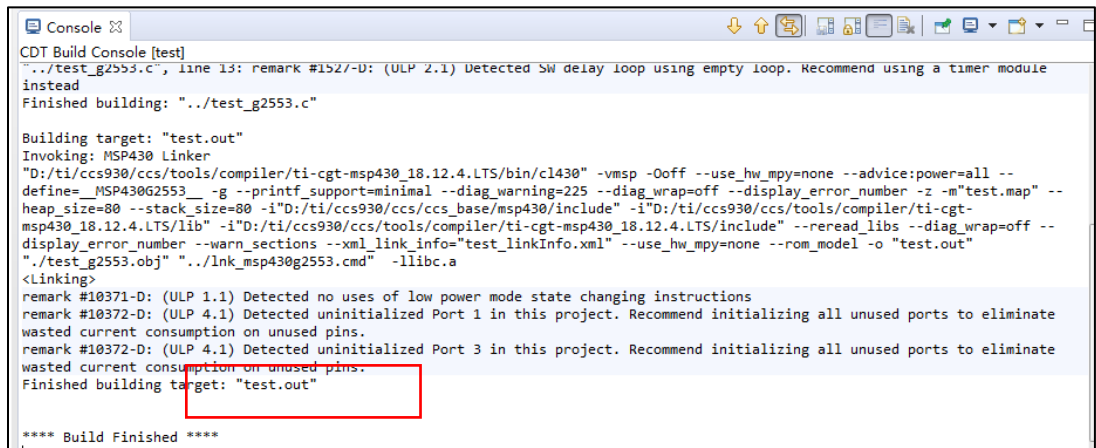


图 A-16-3 编译通过后形成.out 可执行文件

有时即使程序无错，但也会编译不通过，显示如图 A-16-4 的错误提示信息。此时可能是一些杀毒软件阻止了编译操作。只要关闭相应的杀毒功能即可。

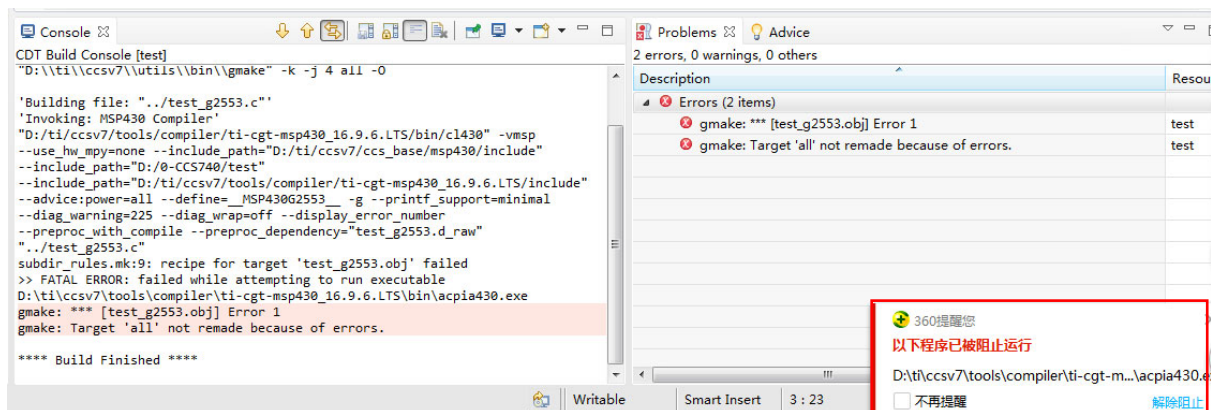


图 A-16-4 因杀毒软件使编译不能进行

四、连接相关硬件

1. MSP430G2553 实验系统简介

打开存放实验系统的储物小盒，可以看到整个实验系统由单片机板一块、扩展板一块、mini USB 数据线一根、和一些短线块及杜邦线构成，如图 A-17。单片机板上主要有一片 msp430g2553 单片机和板载调试器，扩展板上有发光二极管 LED、按键、蜂鸣器、数码管等可用于做实验用的被控对象。详情可以参看附录 B MSP430G2xxx 实验系统简介。

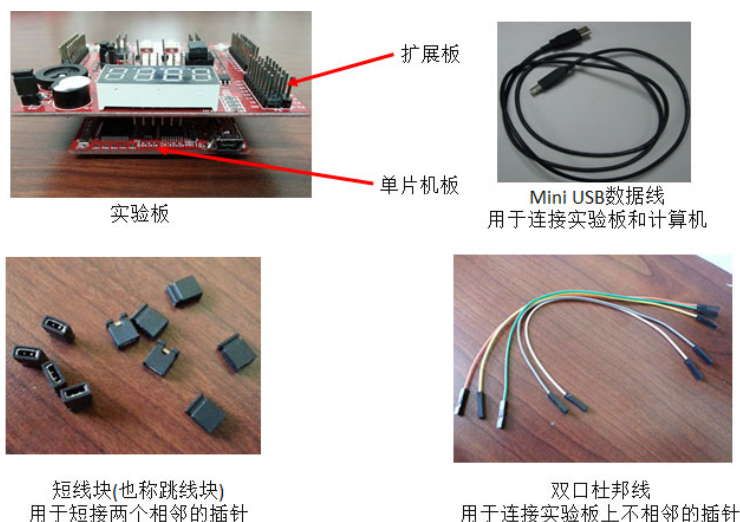


图 A-17 MSP430G2553 实验系统

2. 连接实验板上相关电路

1) 检查下面五处跳线连接情况

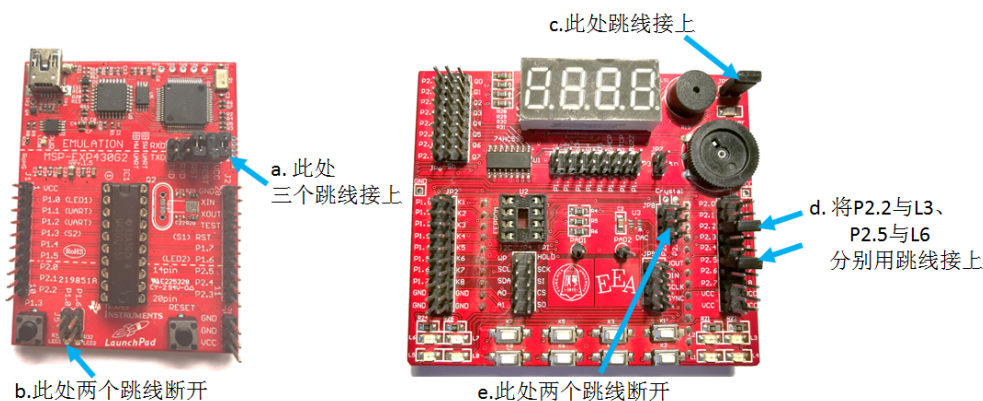


图 A-18-1 实验板上连接的跳线

2) 单片机板与扩展板对接

确认扩展板的插座与单片板上的排针对齐，连接良好，且方向正确，即扩展板上的数码管与单片机板上的 USB 接口在同一侧，如图 A-18-2。

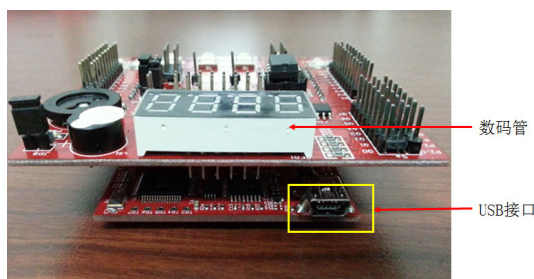


图 A-18-2 单片机板与扩展板的连接

3. 用 mini 数据线将实验板与电脑的 USB 口连接，如图 A-18-3。

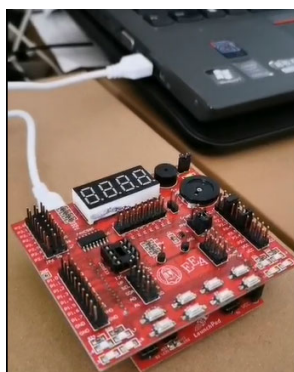








图 A-18-3 将实验板与电脑连接

五、下载程序到目标 MCU 中

点击菜单栏 Debug 快捷图标 ; 或点击菜单栏 Run > Debug, 会出现图 A-19 超低功耗建议(ULP Advisor) 信息提示窗口, 可勾选 “Do not show this message again”, 下次不再显示。接着会显示图 A-20 调试配置 Configuring Debug 信息提示。之后进入图 A-21 的 **Debug 调试界面**, 注意此时 PC 指针指向 main 函数的入口。PC 指针的位置在最左侧用图标  指示, 对应的语句呈现加亮色, 如图 A-21-1 中 PC 指针目前是指向语句 WDTCTL=WDTPW+WDTHOLD。此时单片机系统处于暂停运行状态。当发出运行命令后, CPU 将从 PC 指针指向的语句开始执行程序。运行命令将在下面部分介绍。点击图 A-21-1 右上角的图标  或 , 可以在 Debug 调试界面  和 Edit 编辑界面  切换。Debug 调试界面可对程序的运行做控制、监视; Edit 编辑界面可修改源程序。修改过的程序需要重新编译连接, 形成相应的执行文件, 再下载到单片机中, 才可被 Debug 调试。否则调试的是未修改前的程序。如果没有连接硬件调试器或实验板, 将出现图 A-22 错误提示, 将实验板连上电脑, 再 DEBUG。

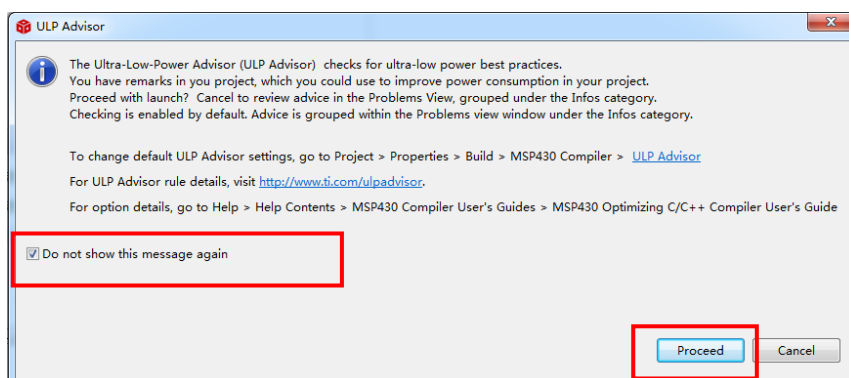


图 A-19 ULP Advisor 信息提示

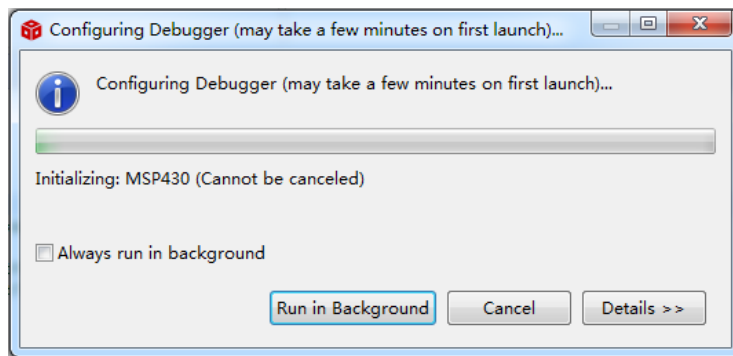


图 A-20 Configuring Debug 信息显示

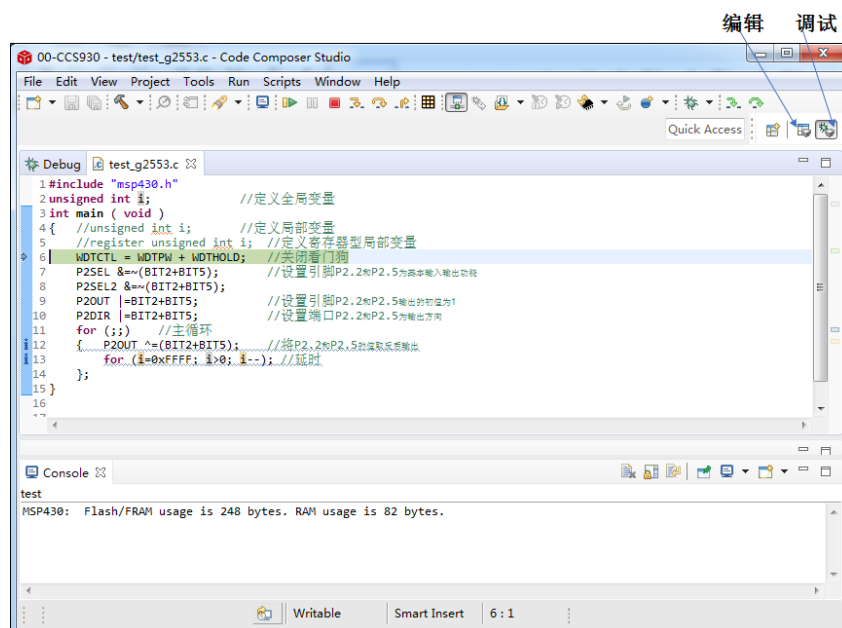


图 A-21-1 Debug 调试界面



图 A-21-2 PC 指向暂停的语句

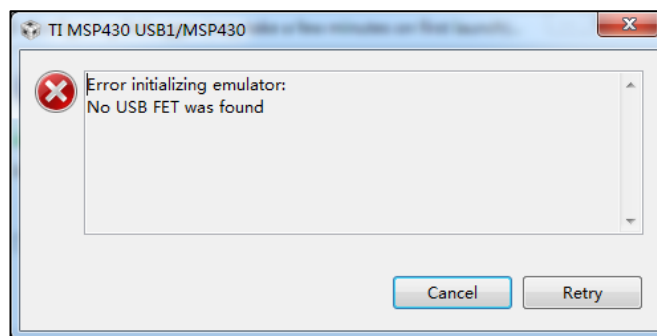


图 A-22 无调试器硬件错误信息

六、运行程序及相关命令

按照计算机“存储程序控制”的工作原理，CCS 下的下载操作只是完成了把程序存储在单片机 Flash ROM 的功能，程序并未运行。需要使用 Debug 下的运行命令，来控制单片机内程序的执行。点击图 A-23 Debug 调试界面菜单栏中的快捷图标，可通过单步、断点、连续运行、暂停、复位、重新运行等操作，灵活实时控制程序的执行；也可点击图 A-24 菜单栏 Run 下的各种命令，完成相应操作。表 1 简述了 Debug 下各运行命令的功能。

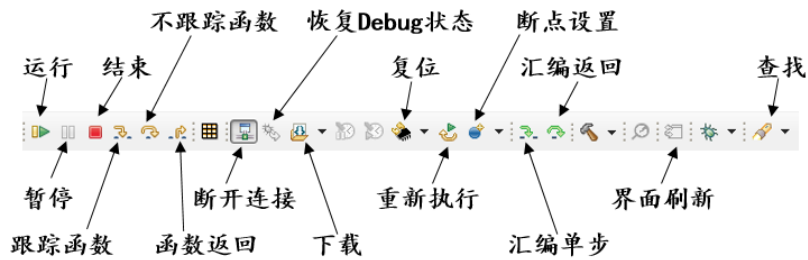


图 A-23 Debug 调试界面的各种操作命令快捷图标

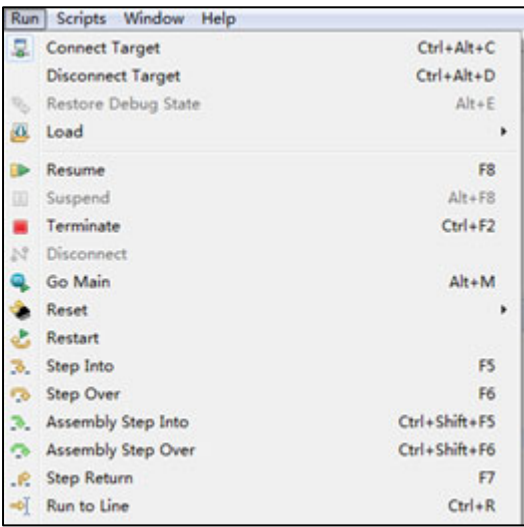




图 A-24 Run 下的命令操作

表 1 Run 下常用命令功能简介

命令	功能
Resume	执行程序
Suspend	暂停
Terminate	结束运行，退出 Debug
Reset (Soft Reset)	复位 MCU，使 PC 重新指向 main 函数的入口处
Reset (Hard Reset)	复位 MCU，使 PC 重新指向上电复位的第一条指令处
Restart	复位 MCU
Step Into	单步执行，如执行的语句 CALL 或是一个 C 函数，跟踪到函数内部
Step Over	执行的语句 CALL 或是一个 C 函数，一步将函数执行完，不跟踪到函数内部
Step Return	跳出被单步跟踪的函数，用于结束对函数的跟踪。
Run to Line	运行到当前光标指向的语句，暂停下来。 用户可事先将光标设置在想要停止下来的地方。
New breakpoint	设置一个断点，使 CPU 运行到此处时，暂停下来

请在当前的 Debug 调试界面下，点击**运行图标** ，CPU 开始执行程序。观察实验板，会看到相应的实验现象。例程 test g2553 的正常现象应该是 L3、L6 两个发光二极管在闪烁。点击**暂停图标** ，程序将暂停运行，此时 L3、L6 停止闪烁，停在某种状态之下，此时 PC 指针指向程序暂停的地方。暂停状态下，可用后面介绍的查看命令查看单片机内部的状态，包括各寄存器、存储单元的内容等。调试程序时，


可根据需要使用运行、暂停命令，也可以点击**复位(restart)图标**，使 PC 指针重新指向 main 函数的入口处，再发运行命令可让 CPU 从头开始运行程序。在 CPU 运行程序的过程中，随时可以使用暂停命令，但使用暂停命令是不能控制 CPU 准确停在何处。

想让 CPU 停在指定的位置，可用设置断点的方法。在程序暂停状态，通过设置断点，使 CPU 在执行程序的过程中，如果执行遇到该语句，会自动暂停下来，以便可以查看当前程序执行的中间状态，包括单片机状态、程序变量等。设置断点的方法是，将鼠标放置在要设置断点语句的最左端，如图 A-25-1 在语句 P2OUT|=BIT2+BIT5 的最左端，双击鼠标左键，就会在该语句前显示**断点图标**，表示此处有一个断点。可设置多个断点，图中语句 P2OUT^=(BIT2+BIT5)前也设置有一个断点。设置断点后，再使用运行命令时，CPU 在执行程序的过程中，如果执行到断点的语句处，就会自动停下来。图 A-25-2 和图 A-25-3 分别是 CPU 在所设的两个断点处暂停的情形。取消断点的方法是用鼠标**双击断点图标**所在的地方，此时断点图标消失。



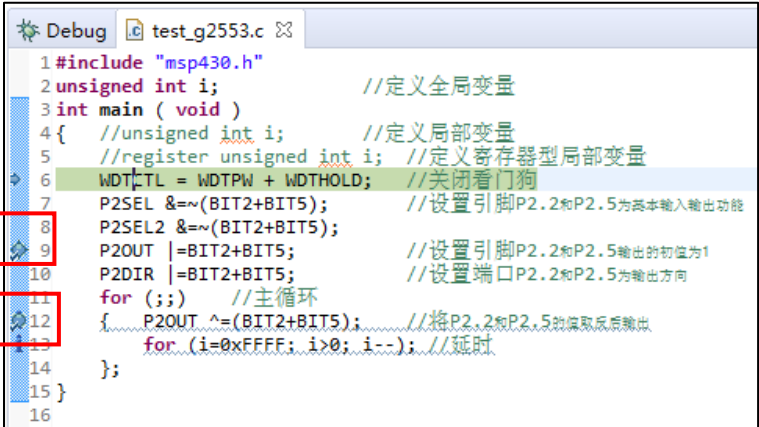
```
1#include "msp430.h"
2unsigned int i;           //定义全局变量
3int main ( void )
4{ //unsigned int i;       //定义局部变量
5  //register unsigned int i; //定义寄存器型局部变量
6  WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
7  P2SEL &=~(BIT2+BIT5);    //设置引脚P2.2和P2.5为基本输入输出功能
8  P2SEL2 &=~(BIT2+BIT5);
9  P2OUT |=BIT2+BIT5;       //设置引脚P2.2和P2.5输出的初值为1
10 P2DIR |=BIT2+BIT5;       //设置端口P2.2和P2.5为输出方向
11 for (;;) //主循环
12 { P2OUT ^= (BIT2+BIT5);   //将P2.2和P2.5的取值取反后输出
13   for (i=0xFFFF; i>0; i--); //延时
14 }
15 }
16
```

图 A-25-1 在指定的语句前设置断点



```
1#include "msp430.h"
2unsigned int i;           //定义全局变量
3int main ( void )
4{ //unsigned int i;       //定义局部变量
5  //register unsigned int i; //定义寄存器型局部变量
6  WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
7  P2SEL &=~(BIT2+BIT5);    //设置引脚P2.2和P2.5为基本输入输出功能
8  P2SEL2 &=~(BIT2+BIT5);
9  P2OUT |=BIT2+BIT5;       //设置引脚P2.2和P2.5输出的初值为1
10 P2DIR |=BIT2+BIT5;       //设置端口P2.2和P2.5为输出方向
11 for (;;) //主循环
12 { P2OUT ^= (BIT2+BIT5);   //将P2.2和P2.5的取值取反后输出
13   for (i=0xFFFF; i>0; i--); //延时
14 }
15 }
16
```

图 A-25-2 运行至断点处暂停



```
1#include "msp430.h"
2unsigned int i;           //定义全局变量
3int main ( void )
4{ //unsigned int i;       //定义局部变量
5  //register unsigned int i; //定义寄存器型局部变量
6  WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
7  P2SEL &=~(BIT2+BIT5);    //设置引脚P2.2和P2.5为基本输入输出功能
8  P2SEL2 &=~(BIT2+BIT5);
9  P2OUT |=BIT2+BIT5;       //设置引脚P2.2和P2.5输出的初值为1
10 P2DIR |=BIT2+BIT5;       //设置端口P2.2和P2.5为输出方向
11 for (;;) //主循环
12 { P2OUT ^= (BIT2+BIT5);   //将P2.2和P2.5的取值取反后输出
13   for (i=0xFFFF; i>0; i--); //延时
14 }
15 }
16
```

图 A-25-3 可设置多个断点

七、查看当前单片机状态

进入 Debug 后，在 Debug 界面下，如图 A-26 点击菜单栏 View 中的各命令，可查看当前单片机内部各寄存器、存储单元、变量的值，并可进行反汇编等操作。表 2 对常用的一些操作功能做了说明。

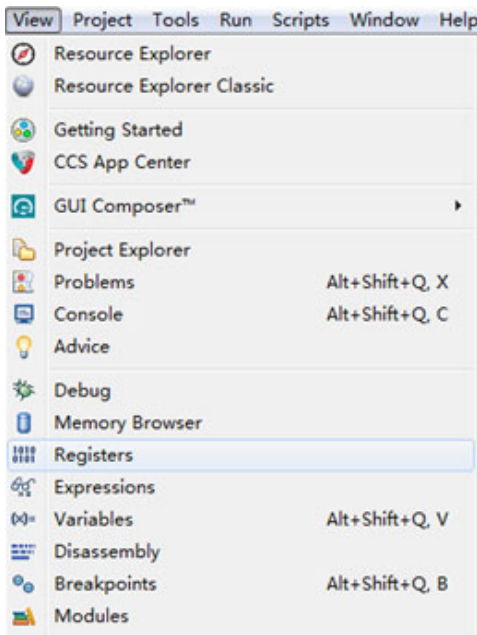








图 A-26 View 下的各种操作

表 2 View 查看操作功能

命令	功能
Registers	查看、修改寄存器、外围模块 I/O 寄存器
Memory Browser	查看、修改存储器
Expressions	查看表达式
Variables	查看变量
Disassembly	反汇编程序
Breakpoints	查看断点

1. View>Registers 查看寄存器

图A-27是寄存器查看窗口，可以查看单片机内各模块内部的寄存器值，包括CPU的寄存器、外围模块的I/O寄存器等。选中要查看的模块，点击模块前的图标 ，可将该模块内部各寄存器的值将显示出来。如点击 Core Register 前的 ，显示图A-28 CPU内部寄存器的值；点击Port_1_2前的 ，显示图A-29 端口P1和P2内部各I/O寄存器的值。点击寄存器前的图标 ，可以进一步显示该寄存器各相应位的值，如图 A-30 显示状态寄存器SR各位的值，图A-31显示P2DIR各位的值。对于可写的寄存器，在窗口 Value栏下，输入新值，可修改相应寄存器或寄存器位的值，如图A-32 修改P2DIR的值，图A-33 修改P2DIR中BIT2的值。如果显示的内容没有发生变化，可点击Registers窗口右上角的刷新按钮  或 。

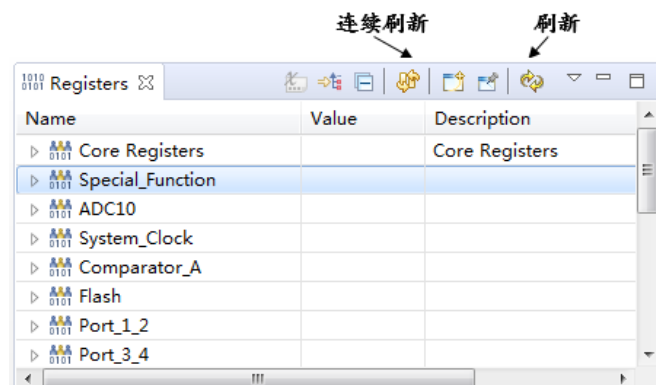


图 A-27 Registers 窗口

Name	Value
Core Registers	
PC	0xC000
SP	0x03FE
SR	0x0000
R3	0x0000
R4	0x2FEF
R5	0xFF7C
R6	0xB6FB
R7	0xCDEF
R8	0xFF7F

图 A-28 显示 CPU 内部寄存器

Name	Value
Port_1_2	
P1IN	0x87
P1OUT	0xFF
P1DIR	0x00
P1IFG	0x71
P1IES	0xFF
P1IE	0x00
P1SEL	0x00
P1SEL2	0x00

图 A-29 显示端口 P1 和 P2 内部寄存器

Name	Value
SR	0x0000
V	0
SCG1	0
SCG0	0
OSCOFF	0
CPUOFF	0
GIE	0
N	0
Z	0
C	0

图 A-30 显示 SR 各位的值

Name	Value
P2DIR	0x00
P7	0
P6	0
P5	0
P4	0
P3	0
P2	0
P1	0
P0	0
P2IFG	0x0F

图 A-31 显示 P2DIR 各位的值

Name	Value
P2IN	0x3F
P2OUT	0xDB
P2DIR	0xFF
P2IFG	0x00
P2IES	0xFF
P2IE	0x00
P2SEL	0xC0
P2SEL2	0x00
P2REN	0x00

图 A-32 修改 P2DIR 的值

Name	Value
P2DIR	0xFF
P7	1
P6	1
P5	1
P4	0
P3	1
P2	1
P1	1
P0	1

图 A-33 修改 P2DIR 中 D4 一位的值

2. View/Expressions 或 View/Variables 变量或表达式查看窗口

图 A-34-1 和图 A-34-2 是变量显示窗口，用于显示程序中定义的变量内容，变量包括局部变量和全局变量，显示信息包括变量的名称、类型、值和所在位置，如果变量是 CPU 内部的寄存器，就显示寄存器名称；如果是存储器单元，则显示单元的地址。在 Expression 或 Name 下输入变量名，即可查看到该变量的信息。或者如图 A-35，在源程序中将光标放在变量的位置上，则会自动显示该变量的信息。

Expression	Type	Value	Address
(x)= i	unsigned int	40911	0x03FC
+ Add new expression			

图 A-34-1 View/Expressions 变量显示窗口

Name	Type	Value	Location
(x)= i	unsigned int	40911	0x03FC

图 A-34-2 View/Variables 变量显示窗口

U-CCS/40 - CCS Debug - Lab_2/main.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

Debug

Lab_2 [Code Composer Studio - Device Debugging] TI MSP430 USB1/MSP430 (Suspended) main() at main.c:12 0xC0FE

main.c

```
1 #include "msp430.h"
2 unsigned int i; //定义全局变量
3 int main ( void )
4 { //register unsigned int i; //定义寄存器型局部变量
5   WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
6   P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为双向
7   P2SEL &=~(BIT2+BIT5);
8   P2OUT |=BIT2+BIT5;
9   P2DIR |=BIT2+BIT5;
10  for (;;) //主循环
11  { P2OUT ^= (BIT2+BIT5);
12    for (i=0xFFFF; i>0; i--)
13    ;
14  }
```

Problems

0 items

Description

Name : i
Default:40911
Hex:0x9FCF
Decimal:40911
Octal:0117717
Binary:1001111111001111b

Expressions

Expression	Type	Value	Address
(x)= i	unsigned int	40911	0x0200
+ Add new expression			

Expression	Type	Value
(x)= i	unsigned int	40911

图 A-35 将光标置于源程序中变量所在位置自动显示变量信息

3. View>Memory Browser 存储器窗口

图 A-36 是存储器显示窗口，可显示存储系统各单元的值。在窗口中地址输入栏（Enter location here），输入存储单元地址、或变量名称等，可查看或修改其对应单元的内容。窗口最左侧显示的是存储单元地址，其后依次显示从该存储单元地址开始连续的若干个存储单元的内容。显示个数与 Memory Browser 窗口大小有关，窗口大，显示的单元数多；窗口小，显示的单元数就少。下一行行首显示该行存储单元首地址，然后是从该地址开始的各单元的内容。如果存储单元是在 RAM 区，如 msp430G2553 中 0x200~0x3FF 的单元，如图 A-37，点击对应单元的内容，输入新的内容，即可改变该单元的内容，图中 0x200 单元的内容从 00，变成了 12。如果不是 RAM 单元，而是 FLASHROM 单元，如 msp430G2553 中 0xC000~0xFFFF 的单元，其内容则不能改变。如图 A-38，点击格式栏中的下拉按钮，在出现的格式中，可以选择想要的显示格式，可以按 8 位、16 位二进制或十六进制显示，也可以把内容当成字符的 ASCII 编码，以字符方式显示，如果内容没有对应的可显示字符，则显示为“.”。

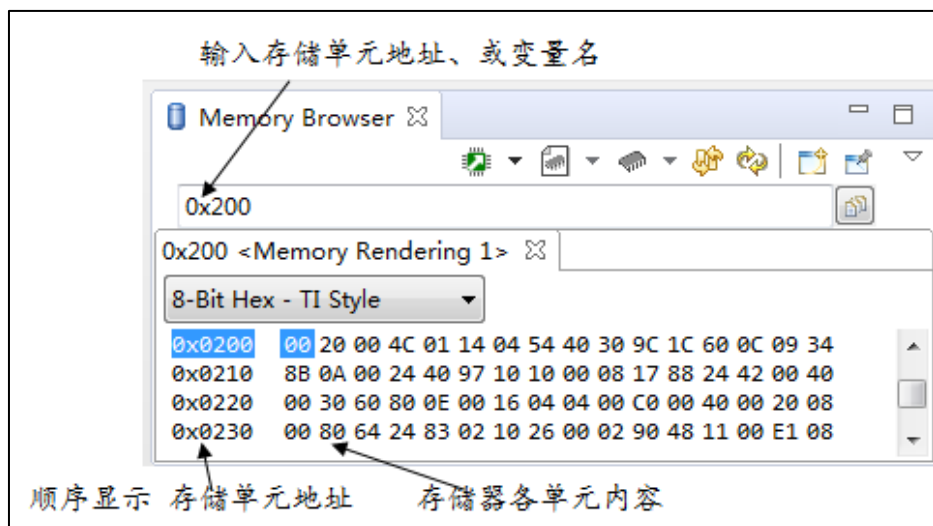


图 A-36 存储器显示窗口

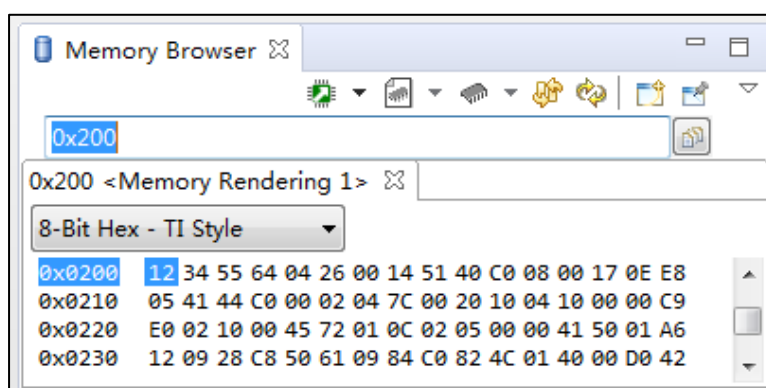


图 A-37 在存储器显示窗口中修改存储单元内容

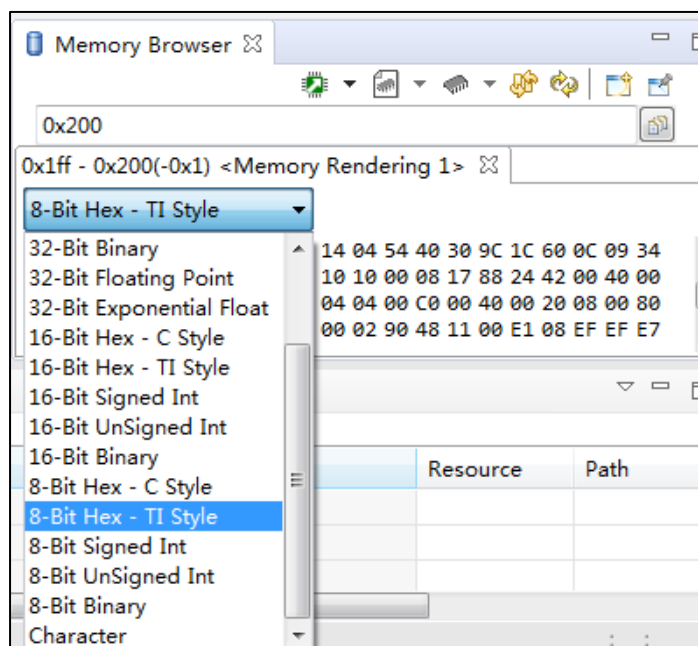
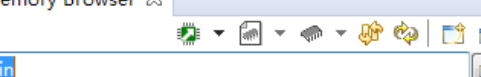


图 A-38 更修 RAM 区中内容的显示格式



Memory Browser

main

0xc000 - main <Memory Rendering 1>

8-Bit Hex - TI Style

0xC000 main

0xC008 B2 40 80 5A 20 01 F2 F0 DB 00 2E 00 F2 F0 DB 00

0xC010 2E 00 F2 D0 24 00 29 00 F2 D0 24 00 2A 00

0xC01E \$C\$L1

[illegible]

Memory Browser

0x200

0x200 <Memory Rendering 2>

Character

0x0200	A	B	X	c	d	&	.	.	.	@
0x0210	.	A	D	\
0x0220	E	r	A	P	.	.	.
0x0230	.	.	.	P	A	L	@	.	.	.	B
0x0240	<	.	}

4. View/Disassembly 反汇编窗口


通常打开反汇编窗口时，是以 PC 指针的值作为反汇编存储单元的首地址，依次显示从此地址开始的反汇编结果。其中最左侧一列是每条指令存放的首地址；中间一列是指令对应的机器码，不同指令的机器码内容和长度不同，指令长度为 1~3 个字，分别占 2~6 个存储单元；最右侧一列是指令对应的汇编语言指令。

在地址栏中可以输入想反汇编的地址、或是函数名称，也可以是寄存器的名称(如 PC)，将显示对应的存储单元反汇编的结果。改变了反汇编开始的地址后，可以按刷新按钮进行刷新。窗口显示了每条指令存放在存储系统的首地址、指令对应的机器码和以及该指令对应的汇编格式指令。同时为了方便查看，窗口中提供了源程序显示控制，用于选择显示或不显示源程序代码。



图 A-41 反汇编窗口

八、退出 DEBUG

Debug 调试状态下, 点击结束图标 , 可以结束调试, 退出 Debug 状态, 退回到只有编辑的状态。此时程序已烧写在单片机的 Flash ROM 中, 给单片机断电, 再重新上电, 会看到烧写在单片机里的程序的运行现象。在 test_g2553 的例程中, 可以看到 L3、L6 仍在闪烁。

九、在一个工程空间中管理多个项目

可在一个工程空间中建立多个项目, 图 A-42 中, 在工程空间 0_CCS740 文件夹下建立了 test、Lab_1、Lab_2 三个项目。3 个项目均放在以 0_CCS740 文件夹中, 为方便查看, 每个项目均放在以项目名为文件夹名的子文件夹下, 文件夹结构图参看图 A-43。每个项目文件夹下可以存放多个文件, 根据需要在项目中添加或移出。用工程空间管理整个学期的实验项目和任务, 比较方便查看和学习。

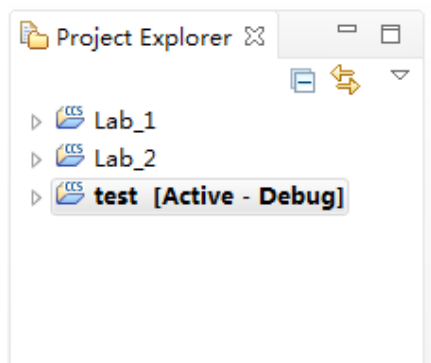


图 A-42 一个工程空间下可建多个项目

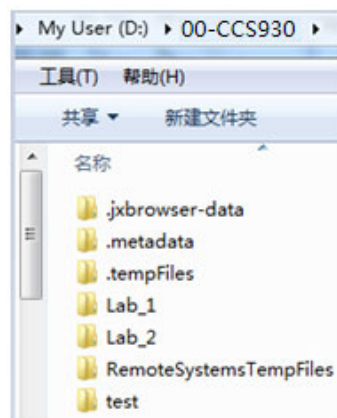


图 A-43 项目空间和项目文件夹结构图

十、CCS 提供的 Help 文档的使用

CCS 提供了很多的帮助文档，用户可利用这些文档对 CCS 的使用、及其编译、连接等进行详细了解。如图 A-45，点击菜单栏的 Help>Help contents，在打开的图 A-46 中输入要查找的内容。表 3 是帮助信息里提供的关于 Debug 调试状态下的一些命令说明。

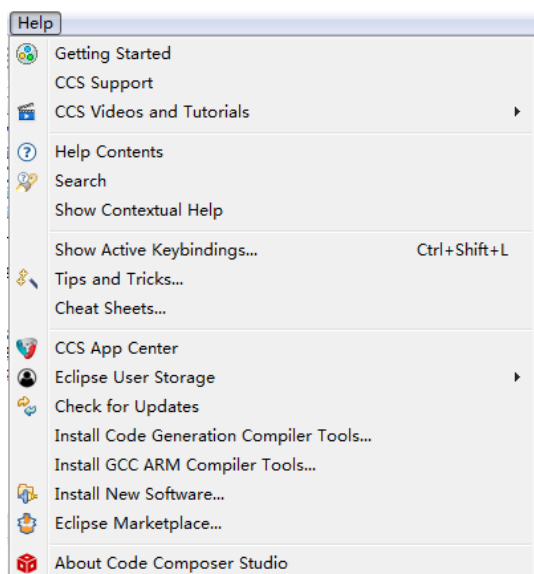


图 A-45 利用 CCS 的 help 获取帮助信息

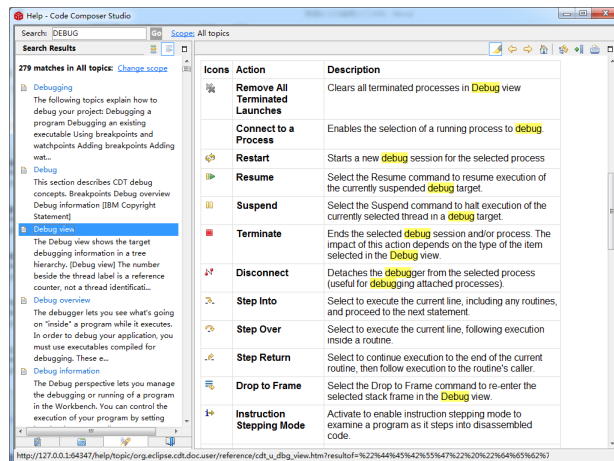












图 A-46 输入需要查找命令的关键字

表 3 Program execution

Image	Name	Description
	Suspends	Suspends the selected target. The rest of the debug views will update automatically with most recent target data.
	Run	Resumes the execution of the currently loaded program from the current PC location. Execution continues until a breakpoint is encountered.
	Run to Line	Resumes the execution of the currently loaded program from the current PC location. Execution continues until the specific source/assembly line is reached.
	Go to Main	Runs the programs until the beginning of function main in reached.
	Step Into	Steps into the highlighted statement.
	Step Over	Steps over the highlighted statement. Execution will continue at the next line either in the same method or (if you are at the end of a method) it will continue in the method from which the current method was called. The cursor jumps to the declaration of the method and selects this line.
	Step Return	Steps out of the current method.
	Reset	Resets the selected target. The drop-down menu has various advanced reset options, depending on the selected device.
	Restart	Restores the PC to the entry point for the currently loaded program. If the debugger option "Run to main on target load or restart" is set the target will run to the specified symbol, otherwise the execution state of the target is not changed.
	Assembly Step Into	The debugger executes the next assembly instruction, whether source is available or not.